## Contents

### 1 Basics
1.1 Introduction ......................................................... 3
1.2 Main Features ...................................................... 4
1.3 Running existing negotiators/negotiation protocols ................................. 4
1.4 Developing a negotiator ............................................. 5
1.5 Developing a negotiation protocol .................................. 5
1.6 Running a world simulation ........................................ 5
1.7 Acknowledgement .................................................... 6

### 2 Installation
2.1 Stable release ...................................................... 7
2.2 Latest Commit ..................................................... 7
2.3 From sources ...................................................... 8
2.4 [Optional] Post Installation ....................................... 8

### 3 Overview
3.1 Organization ..................................................... 9
3.2 Issues, Outcomes, and Responses .................................. 10
3.3 Utilities .......................................................... 14
3.4 Other utility function types ....................................... 19
3.5 Negotiators ....................................................... 20
3.6 Mechanisms (Negotiations) ......................................... 21

### 4 Tutorials
4.1 Running a Negotiation ............................................. 23
4.2 Develop a new negotiator .......................................... 27
4.3 Develop a new mechanism (protocol) ................................ 27
4.4 Develop a new simulation (world) ................................ 28
4.5 Develop a new agent (for your simulation) ........................... 28
4.6 Run a session of the SCML world .................................. 28
4.7 Develop a factory manager (agent) for the SCM world ...................... 32

### 5 Base Modules
5.1 negmas.outcomes Module ......................................... 37
5.2 negmas.utilities Module .......................................... 56
5.3 negmas.mechanisms Module ....................................... 180
5.4 negmas.common Module ........................................... 188
5.5 negmas.negotiators Module ........................................ 193
5.6 negmas.sao Module ............................................... 206
5.7 negmas.genius Module ............................................. 286
5.8 negmas.situated Module ........................................... 294
11.32 Release 0.1.3 ................................................................. 598
11.33 Release 0.1.2 ................................................................. 598

12 Reference ............................................................... 599
12.1 DEFAULT_NEGOTIATOR ...................................................... 599
12.2 INVALID_STEP ............................................................... 599
12.3 INVALID_Utility ............................................................... 599
12.4 NO_PRODUCTION ............................................................. 600
12.5 GenericMapping ............................................................. 600
12.6 IterableMapping ............................................................ 600
12.7 gget .............................................................................. 600
12.8 gmap .............................................................................. 601
12.9 enumerate ..................................................................... 601
12.10 iget ............................................................................. 602
12.11 iitems ............................................................................ 602
12.12 ikeys .......................................................................... 603
12.13 ivvalues ...................................................................... 604
12.14 DEFAULT_DUMP_EXTENSION ............................................. 604
12.15 PYTHON_CLASS_IDENTIFIER ........................................... 604
12.16 Outcome ...................................................................... 605
12.17 INVALID_Utility ............................................................. 605
12.18 UtilityValue ................................................................. 605

13 Indices and tables .................................................. 607

Index ................................................................. 609
NegMAS is a python library for developing autonomous negotiation agents embedded in simulation environments. The name `negmas` stands for either NEGotiation MultiAgent System or NEGotiations Managed by Agent Simulations (your pick). The main goal of NegMAS is to advance the state of the art in situated simultaneous negotiations. Nevertheless, it can; and was used; in modeling simpler bilateral and multi-lateral negotiations, preference elicitation, etc.
NegMAS is a python library for developing autonomous negotiation agents embedded in simulation environments. The name `negmas` stands for either NEGotiation MultiAgent System or NEGotiations Managed by Agent Simulations (your pick). The main goal of NegMAS is to advance the state of the art in situated simultaneous negotiations. Nevertheless, it can; and was used; in modeling simpler bilateral and multi-lateral negotiations, preference elicitation, etc.

**Note:** A YouTube playlist to help you use NegMAS for ANAC2019 SCM league can be found [here](#).

### 1.1 Introduction

This package was designed to help advance the state-of-art in negotiation research by providing an easy-to-use yet powerful platform for autonomous negotiation targeting situated simultaneous negotiations. It grew out of the NEC-AIST collaborative laboratory project.

By *situated* negotiations, we mean those for which utility functions are not pre-ordained by fiat but are a natural result of a simulated business-like process.

By *simultaneous* negotiations, we mean sessions of dependent negotiations for which the utility value of an agreement of one session is affected by what happens in other sessions.

The documentation is available at: [documentation](#)
1.2 Main Features

This platform was designed with both flexibility and scalability in mind. The key features of the NegMAS package are:

1. The public API is decoupled from internal details allowing for scalable implementations of the same interaction protocols.
2. Supports agents engaging in multiple concurrent negotiations.
3. Provides support for inter-negotiation synchronization either through coupled utility functions or through central control agents.
4. The package provides sample negotiators that can be used as templates for more complex negotiators.
5. The package supports both mediated and unmediated negotiations.
6. Supports both bilateral and multilateral negotiations.
7. Novel negotiation protocols and simulated worlds can be added to the package as easily as adding novel negotiators.
8. Allows for non-traditional negotiation scenarios including dynamic entry/exit from the negotiation.
9. A large variety of built in utility functions.
10. Utility functions can be active dynamic entities which allows the system to model a much wider range of dynamic ufuns compared with existing packages.
11. A distributed system with the same interface and industrial-strength implementation is being created allowing agents developed for NegMAS to be seemingly employed in real-world business operations.

To use negmas in a project

```python
import negmas
```

The package was designed for many uses cases. On one extreme, it can be used by an end user who is interested in running one of the built-in negotiation protocols. On the other extreme, it can be used to develop novel kinds of negotiation agents, negotiation protocols, multi-agent simulations (usually involving situated negotiations), etc.

1.3 Running existing negotiators/negotiation protocols

Using the package for negotiation can be as simple as the following code snippet:

```python
from negmas import SAOMechanism, AspirationNegotiator, MappingUtilityFunction
session = SAOMechanism(outcomes=10, n_steps=100)
negotiators = [AspirationNegotiator(name=f'a{index}') for index in range(5)]
for negotiator in negotiators:
    session.add(negotiator, ufun=MappingUtilityFunction(lambda x: random.random() - x[0]))
session.run()
```

In this snippet, we created a mechanism session with an outcome-space of 10 discrete outcomes that would run for 10 steps. Five agents with random utility functions are then created and added to the session. Finally the session is run to completion. The agreement (if any) can then be accessed through the state member of the session. The library provides several analytic and visualization tools to inspect negotiations. See the first tutorial on Running a Negotiation for more details.
1.4 Developing a negotiator

Developing a novel negotiator slightly more difficult by is still doable in few lines of code:

```python
from negmas.sao import SAONegotiator
from negmas import ResponseType

class MyAwesomeNegotiator(SAONegotiator):
    def __init__(self):
        # initialize the parents
        super().__init__(self)

    def respond(self, offer, state):
        # decide what to do when receiving an offer
        return ResponseType.ACCEPT_OFFER

    def propose(self, state):
        # proposed the required number of proposals (or less)
        pass
```

By just implementing `respond()` and `propose()`. This negotiator is now capable of engaging in alternating offers negotiations. See the documentation of `Negotiator` for a full description of available functionality out of the box.

1.5 Developing a negotiation protocol

Developing a novel negotiation protocol is actually even simpler:

```python
from negmas.mechanisms import Mechanism

class MyNovelProtocol(Mechanism):
    def __init__(self):
        super().__init__()

    def round(self):
        # one step of the protocol
        pass
```

By implementing the single `round()` function, a new protocol is created. New negotiators can be added to the negotiation using `add()` and removed using `remove()`. See the documentation for a full description of `Mechanism` available functionality out of the box [Alternatively you can use `Protocol` instead of `Mechanism`].

1.6 Running a world simulation

The *raison d'être* for NegMAS is to allow you to develop negotiation agents capable of behaving in realistic business like simulated environments. These simulations are called *worlds* in NegMAS. Agents interact with each other within these simulated environments trying to maximize some intrinsic utility function of the agent through several possibly simultaneous negotiations.

The situated module provides all that you need to create such worlds. An example can be found in the `scml` package. This package implements a supply chain management system in which factory managers compete to maximize their profits in a market with only negotiations as the means of securing contracts.
1.7 Acknowledgement

NegMAS tests use scenarios used in ANAC 2010 to ANAC 2018 competitions obtained from the Genius Platform. These domains can be found in the tests/data and notebooks/data folders.
CHAPTER 2

Installation

It is always a good idea to install your packages to a virtual environment. This is a reminder of how to create one using the standard `venv` module in python 3 (inside a folder called `workspace`):

```
$ mkdir workspace; cd workspace
$ python -m venv venv
$ source venv/bin/activate
```

To check that you are in your newly created environment run the following commands depending on your OS. In windows machines:

```
$ where python
```

In *nix/macos machines:

```
$ which python
```

In both cases you should find that python is running from `venv/bin` within your workspace folder.

After creating your venv, it is recommended to update pip:

```
$ pip install -U pip
```

### 2.1 Stable release

To install negmas, run this command in your terminal:

```
$ pip install negmas
```

This is the preferred method to install negmas, as it will always install the most recent stable release.

If you don’t have pip installed, this Python installation guide can guide you through the process.

### 2.2 Latest Commit

To installing the latest commit of negmas from github, you can run this command in your terminal:
$ pip install git+https://github.com/yasserfarouk/negmas

Please note that this may not be a stable version. As always make sure that you are running the command within the correct virtual environment.

### 2.3 From sources

The sources for negmas can be downloaded from the Github repo. You can either clone the public repository:

```bash
$ git clone git://github.com/yasserfarouk/negmas
```

Or download the tarball:

```bash
$ curl -OL https://github.com/yasserfarouk/negmas/tarball/master
```

Once you have a copy of the source, you can install it with:

```bash
$ python setup.py install
```

or, better yet, if you do not have Poetry, install it as explained in Poetry_install, and then just run:

```bash
$ poetry install
```

We recommend the use of Poetry.

### 2.4 [Optional] Post Installation

After installation, some new commands will be added to your environment (hopefully it is a virtual environment). Among them there is a script called: `rungenius`.

To test your installation, run the following command (note that test_genius tests will be skipped):

```bash
$ python -m pytest --pyargs negmas
```

If you want to test the Genius bridge, you need to download the Genius-NegMAS-Bridge Once you have the file save it to some path in your machine and run the following command (note that it will run in the foreground until you press Ctrl-C to close it):

```bash
$ negmas genius --path=path-to-genius-bridge.jar
```

This will start a service that allows NegMAS to use Genius. After this process starts, you can run the tests involving genius using:

```bash
$ python -m pytest --pyargs negmas/tests/test_genius
```

Notice that this test will report coverage for test files as well. That is not ideal. To exclude such files from the report you will need to use a .coveragerc file as described in Coverage.
negmas was designed mainly to support multi-strand multilateral multi-issue negotiations with complex utility functions. This section gives an introduction to the main concepts of the public interface.

In order to use the library you will need to import it as follows (assuming that you followed the instructions in the installation section of this document):

```python
# This is to make the results reproducible if you are using the Jupyter notebook
from random import seed
seed(0)
import negmas
```

### 3.1 Organization

The package is organized into a set of modules the combine together related functionality. In general there are base modules that implement the most general abstractions and concepts and then specialized modules that implement the computational structures needed for a specific application or research domain:

- **Base Modules** These are the most general modules and all other specialized modules use the computational resources defined here. The base modules provided in this version are:
  1. **outcomes** This module represents issues, outcomes and responses and provides basic functions and methods to operate with and on them.
  2. **utilities** This module represents the base type of all utilities and different widely used utility function types including linear and nonlinear utilities and constraint-based utilities.
  3. **negotiators** This module represents basic negotiation agent implementation and provides basic interfaces to be overridden (implemented) by higher specialized modules.
  4. **mechanisms** This module represents the most basic conceptual view of a negotiation protocol supporting both mediate and unmediated mechanisms. The term mechanism was used instead of the more common protocol to stress the fact that this mechanism need not be a standard negotiation protocol. For example auction mechanisms (like second-price auctions) can easily be implemented in this package.
  5. **opponent_models** This module provides the basic interface for all opponent models.
6. **situated** This module implements world simulations within which agents with intrinsic utility functions can engage in simultaneous connected situated negotiations. It is the most important module for the goals of this library.

7. **Helper Modules** These modules provide basic activities that is not directly related to the negotiation but that are relied upon by different base modules. The end user is not expected to interact directly with these modules.

   - **common** Provides common interfaces that are used by all other modules.
   - **generics** Provides a set of types and interfaces to increase the representation flexibility of different base modules.
   - **helpers** Various helper functions and classes used throughout the library including mixins for logging.

• **App Modules** This namespace provides the modules needed to run different apps that represent worlds within which negotiations take place.

1. **scml** The Supply Chain Management App as defined for the SCM league of ANAC 2019 competition.

To simplify the use of this library, all classes and functions from all base modules are aliased in the root package (except generics and helpers). This is an example of importing just `Outcome`:

```python
from negmas import Outcome
```

It is possible to just import everything in the package using:

```python
from negmas import *
```

As usual you can just import everything in a separate namespace using:

```python
import negmas
```

### 3.2 Issues, Outcomes, and Responses

Negotiations are conducted between multiple agents with the goal of achieving an agreement (usually called a contract) on one of several possible outcomes. Each outcome is in general an assignment to some value to a set of issues. Each issue is a variable that can take one of a probably infinite set of values from some predefined domain.

The classes and functions supporting management of issues, outcomes and responses are combined in the `outcomes` module.

To directly handle issues, outcomes and responses; you need to import the `outcomes` modules. To simplify the code snippets in this overview, we will just import everything in this module but you can of course be selective.

#### 3.2.1 Issues

Issues are represented in `negmas` using the `issue` class. An issue is defined by a set of values and a name. It can be created as follows:

- Using a set of strings:

  ```python
  # an issue with randomly assigned name
  issue1 = Issue(values=['to be', 'not to be'])
  print(issue1)
  # an issue with given name:
  issue2 = Issue(values=['to be', 'not to be'], name='The Problem')
  print(issue2)
  ```
The Problem: ['to be', 'not to be']

- Using a single integer to give an issue which takes any value from 0 to the given integer minus 1:

```python
issue3 = Issue(values=10, name='number of items')
print(issue3)
```

number of items: 10

- Using a tuple with a lower and upper real-valued boundaries to give an issue with an infinite number of possibilities (all real numbers in between)

```python
issue4 = Issue(values=(0.0, 1.0), name='cost')
print(issue4)
```

cost: (0.0, 1.0)

The `Issue` class provides some useful functions. For example you can find the cardinality of any issue using:

```python
[issue2.cardinality(), issue3.cardinality(), issue4.cardinality()]
```

[2, 10, -1]

It is also possible to check the type of the issue and whether it is discrete or continuous:

```python
[issue2.type, issue2.is_discrete(), issue2.is_continuous()]
```

['discrete', True, False]

It is possible to check the total cardinality for a set of issues (with the usual -1 encoding infinity):

```python
[Issue.n_outcomes([issue1, issue2, issue3, issue4]), # expected -1 because of issue4
 Issue.n_outcomes([issue1, issue2, issue3])] # expected 40 = 2 * 2 * 4
```

[-1, 40]

You can pick random valid or invalid values for the issue:

```python
[
    [issue1.rand_valid(), issue1.rand_invalid()],
    [issue2.rand_valid(), issue2.rand_invalid()],
    [issue3.rand_valid(), issue3.rand_invalid()],
    [issue4.rand_valid(), issue4.rand_invalid()],
]
```

[['not to be', '20190203-085645wL56nGisto be20190203-085645WgNZq6IT'],
 ['to be', '20190203-085645tgUe52Rvnot to be20190203-085645JgwBuNO6'],
 [3, 19],
 [0.47700977655271704, 1.86630992777164]]

You can also list all valid values for an issue using `all`. Notice that this property is a generator so it is memory efficient for the case when an issue has many values.

```python
print(list(issue1.all))
print(list(issue2.all))
print(list(issue3.all))
```

(continues on next page)
```python
try:
    print(list(issue4.all))
except ValueError as e:
    print(e)
```

['to be', 'not to be']
['to be', 'not to be']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Cannot return all possibilities of a continuous issue

### 3.2.2 Outcomes

Now that we know how to define issues, defining outcomes from a negotiation is even simpler. An outcome can be any python mapping or iterable with a known length. That includes dictionaries, lists, tuples among many other.

Here is how to define an outcome for the last three issues mentioned above:

```python
valid_outcome = {'The Problem': 'to be', 'number of items': 5, 'cost': 0.15}
invalid_outcome = {'The Problem': 'to be', 'number of items': 10, 'cost': 0.15}
```

Notice that the invalid_outcome is assigning a value of 10 to the number of items issue which is not an acceptable value (cost ranges between 0 and 9).

Because outcomes can be represented with many builtin collection classes, the only common ancestor of all outcome objects is the object class. Nevertheless, the outcomes module provide a type-alias Outcome that can be used for static type checking if needed. The outcomes module also provides some functions for dealing with outcome objects in relation to Issues. These are some examples:

```python
[ outcome_is_valid(valid_outcome, [issue2, issue3, issue4]) # valid giving True,
  outcome_is_valid(invalid_outcome, [issue2, issue3, issue4])] # invalid giving False
```

It is not necessary for an outcome to assign a value for all issues to be considered valid. For example the following outcomes are all valid for the last three issues given above:

```python
[ outcome_is_valid({'The Problem': 'to be'}, [issue2, issue3, issue4]) # valid giving True,
  outcome_is_valid({'The Problem': 'to be', 'number of items': 5}, [issue2, issue3, issue4])] # valid giving True
```

It is also important for some applications to check if an outcome is complete in the sense that it assigns a valid value to every issue in the given set of issues. This can be done using the outcome_is_complete function:

```python
[ outcome_is_complete(valid_outcome, [issue2, issue3, issue4]) # complete -> True,
  outcome_is_complete(invalid_outcome, [issue2, issue3, issue4]) # invalid -> incomplete -> False,
  outcome_is_complete({'The Problem': 'to be'}, [issue2, issue3, issue4]) # incomplete -> False
]```
It is sometimes tedious to keep track of issue names in dictionaries. For this reason, the library provides a type called `OutcomeType`. Inheriting your dataclass from an `OutcomeType` will allow it to act both as a dict and a normal dot accessible object:

```python
from dataclasses import dataclass
class MyOutcome(OutcomeType):
    problem: bool
    price: float
    quantity: int
```

Now you can use objects of `MyOutcome` as normal outcomes:

```python
issues = [Issue(['to be', 'not to be'], name='problem'),
          Issue((0.0, 3.0), name='price'),
          Issue(5, name='quantity')]
outcomes = Issue.sample(issues, n_outcomes = 5, astype=MyOutcome)
for _ in outcomes:
    print(_)

MyOutcome(problem='to be', price=1.0848388916904823, quantity=0)
MyOutcome(problem='to be', price=1.8906644944040263, quantity=0)
MyOutcome(problem='not to be', price=1.2102407956353904, quantity=0)
MyOutcome(problem='not to be', price=2.957644296190988, quantity=1)
MyOutcome(problem='not to be', price=2.847064181581488, quantity=0)
```

The `sample` function created objects of type `MyOutcome` that can be accessed using either the dot notation or as a dict:

```python
print(outcomes[0].price)
print(outcomes[0]['price'])
print(outcomes[0].get('price', None))
```

```
1.0848388916904823
1.0848388916904823
1.0848388916904823
```

OutcomeType is intended to be used as a syntactic sugar around your outcome objects but it provides almost no functionality above a dict.

### 3.2.3 Outcome Ranges and constraints

Sometimes, it is important to represent not only a single outcome but a range of outcomes. This can be represented using an `OutcomeRange`. Again, an outcome range can be almost any mapping or iterable in python including dictionaries, lists, tuples, etc with the only exception that the values stored in it can be not only int, str, float but also tuples of two of any of them representing a range. This is easier shown:

```python
range1 = {'The Problem': ['to be', 'not to be'], 'number of items': 5, 'cost': (0.1, 0.2)}
```

`range1` represents the following range of outcomes:

- **The Problem**: accepts both `to be` and `not to be`
- **number of items**: accepts only the value 5
- **cost**: accepts any real number between 0.1 and 0.2 up to representation error
It is easy to check whether a specific outcome is within a given range:

```python
outcome1 = {'The Problem': 'to be', 'number of items': 5, 'cost': 0.15}
outcome2 = {'The Problem': 'to be', 'number of items': 10, 'cost': 0.15}
[ outcome_in_range(outcome1, range1)  # True,
  outcome_in_range(outcome2, range1)  # False
]
```

In general outcome ranges constraint outcomes depending on the type of the constraint:

- **tuple** The outcome must fall within the range specified by the first and second elements. Only valid for values that can be compared using `__lt__` (e.g. int, float, str).
- **single value** The outcome must equal this given value.
- **list of values** The outcome must be within the list.
- **list of tuples** The outcome must fall within one of the ranges specified by the tuples.

### 3.2.4 Responses

When negotiations are run, agents are allowed to respond to given offers for the final contract. An offer is simple an outcome (either complete or incomplete depending on the protocol but it is always valid). Agents can then respond with one of the values defined by the `Response` enumeration in the `outcomes` module. Currently these are:

- **ACCEPT_OFFER** Accepts the offer.
- **REJECT_OFFER** Rejects the offer.
- **END_NEGOTIATION** This implies rejection of the offer and further more indicates that the agent is not willing to continue with the negotiation. The protocol is free to handle this situation. It may just end the negotiation with no agreement, may just remove the agent from the negotiation and keep it running with the remaining agents (if that makes sense) or just gives the agent a second chance by treating it as just a `REJECT_OFFER` case. In most case the first response (just end the negotiation) is expected.
- **NO_RESPONSE** Making no response at all. This is usually not allowed by negotiation protocols and will be considered a protocol violation in most cases. Nevertheless, negotiation protocols are free to handle this response when it arise in any way.

### 3.3 Utilities

Agents engage in negotiations to maximize their utility. That is the central dogma in negotiation research. `negmas` allows the user to define their own utility functions based on a set of predefined base classes that can be found in the `utilities` module.

#### 3.3.1 Utility Values

In most applications, utility values can be represented by real numbers. Nevertheless, some applications need a more complicated representation. For example, during utility elicitation (the process of learning about the utility function of the human being represented by the agent) or opponent modeling (the process of learning about the utility function of an opponent), the need may arise to represent a probability distribution over utilities.

`negmas` allows all functions that receive a utility value to receive a utility distribution. This is achieved through the use of two basic type definitions:
• **UtilityDistribution** That is a probability Distribution class capable of representing probabilistic variables having both continuous and discrete distributions and applying basic operations on them (addition, subtraction and multiplication). Currently we use scipy.stats for modeling these distributions but this is an implementation detail that should not be relied upon as it is likely that the probabilistic framework will be changed in the future to enhance the flexibility of the package and its integration with other probabilistic modeling packages (e.g. PyMC3).

• **UtilityValue** This is the input and output type used whenever a utility value is to be represented in the whole package. It is defined as a union of a real value and a UtilityDistribution(Union[float, UtilityDistribution]). This way, it is possible to pass utility distributions to most functions expecting (or returning) a utility value including utility functions.

This means that both of the following are valid utility values

```
u1 = 1.0
u2 = UtilityDistribution(dtype='norm')  # standard normal distribution
print(u1)
print(u2)
1.0
norm(loc:0.0, scale:1.0)
```

### 3.3.2 Utility Functions

Utility functions are entities that take an Outcome and return its UtilityValue. There are many types of utility functions defined in the literature. In this package, the base of all utility functions is the UtilityFunction class which is defined in the utilities module. It behaves like a standard python Callable which can be called with a single Outcome object (i.e. a dictionary, list, tuple etc representing an outcome) and returns a UtilityValue. This allows utility functions to return a distribution instead of a single utility value.

Utility functions in negmas have a helper property called type which returns the type of the utility function and a helper function eu for returning the expected utility of a given outcome which is guaranteed to return a real number (float) even if the utility function itself is returning a utility distribution.

To implement a specific utility function, you need to override the single __call__ function provided in the UtilityFunction abstract interface. This is a simple example:

```
class ConstUtilityFunction(UtilityFunction):
    def __call__(self, offer):
        try:
            return 3.0 * offer['cost']
        except KeyError:
            # No value was given to the cost
            return None

def xml(self):
    return '<ufun const=True value=3.0></ufun>'

f = ConstUtilityFunction()
[f({'The Problem': 'to be'}), f({'cost': 10})]

[None, 30.0]
```

Utility functions can store internal state and use it to return different values for the same outcome over time allowing for dynamic change or evolution of them during negotiations. For example this silly utility function responds to the mood of the user:

```
class MoodyUtilityFunction(UtilityFunction):
    def __init__(self, mood='good'):
        super().__init__()
        self.mood = mood

[f({'The Problem': 'to be'}), f({'cost': 10})]
```

(continues on next page)
def __call__(self, offer):
    return float(offer['cost']) if self.mood == 'good'
    else 0.1 * offer['cost'] if self.mood == 'bad'
    else None

def set_mood(self, mood):
    self.mood = mood

def xml(self):
    pass

offer = {'cost': 10.0}
f = MoodyUtilityFunction()
# I am in a good mode now
print(f'Utility in good mood of {offer} is {f(offer)}')
f.set_mood('bad')
print(f'Utility in bad mood of {offer} is {f(offer)}')
f.set_mood('undecided')
print(f'Utility in good mood of {offer} is {f(offer)}')

Utility in good mood of {'cost': 10.0} is 10.0
Utility in bad mood of {'cost': 10.0} is 1.0
Utility in good mood of {'cost': 10.0} is None

Notice that (as the last example shows) utility functions can return `None` to indicate that the utility value cannot be inferred for this outcome/offer.

The package provides a set of predefined utility functions representing most widely used types. The following subsections describe them briefly:

### 3.3.3 Linear Aggregation Utility Functions

The `LinearAggregationUtilityFunction` class represents a function that linearly aggregate utilities assigned to issues in the given outcome which can be defined mathematically as follows:

\[
U(o) = \sum_{i=0}^{\mid o \mid} w_i \times g_i(o_i)
\]  

where \( o \) is an outcome, \( w \) is a real-valued weight vector and \( g \) is a vector of functions each mapping one issue of the outcome to some real-valued number (utility of this issue).

Notice that despite the name, this type of utility functions can represent nonlinear relation between issue values and utility values. The linearity is in how these possibly nonlinear mappings are being combined to generate a utility value for the outcome.

For example, the following utility function represents the utility of a `buyer` who wants low cost, many items, and prefers delivery:

```python
buyer_utility = LinearUtilityAggregationFunction({'price': lambda x: - x,
                                                      'number of items': lambda x: 0.5 * x,
                                                      'delivery': {'delivered': 1.0, 'not delivered': 0.0}})
```

Given this definition of utility, we can easily calculate the utility of different options:

```python
print(buyer_utility({'price': 1.0, 'number of items': 3, 'delivery': 'not delivered --'}))
```
Now what happens if we offer to deliver the items:

```python
print(buyer_utility({'price': 1.0, 'number of items': 3, 'delivery': 'delivered'}))
```

And if delivery was accompanied with an increase in price

```python
print(buyer_utility({'price': 1.8, 'number of items': 3, 'delivery': 'delivered'}))
```

It is clear that this buyer will still accept that increase of price from '1.0' to '1.8' if it is accompanied with the delivery option.

### 3.3.4 Nonlinear Aggregation Utility Functions

A direct generalization of the linear aggregation utility functions is provided by the `NonLinearAggregationUtilityFunction` which represents the following function:

\[
U(o) = f\left(\{g_i(o_i)\}\right)
\]  

(3.2)

where \(g\) is a vector of functions defined as before and \(f\) is a mapping from a vector of real-values to a single real value.

For example, a seller’s utility can be defined as:

```python
seller_utility = NonLinearUtilityAggregationFunction(
    {'price': lambda x: x,
     'number of items': lambda x: 0.5 * x,
     'delivery': {'delivered': 1.0, 'not delivered': 0.0}},
    f=lambda x: x['price']/x['number of items'] - 0.5 * x[--;'delivery']
)
```

This utility will go up with the price and down with the number of items as expected but not in a linear fashion.

We can now evaluate different options similar to the case for the buyer:

```python
print(seller_utility({'price': 1.0, 'number of items': 3, 'delivery': 'not delivered'}))
```

0.6666666666666666

```python
print(seller_utility({'price': 1.0, 'number of items': 3, 'delivery': 'delivered'}))
```

0.16666666666666663

```python
print(seller_utility({'price': 1.8, 'number of items': 3, 'delivery': 'delivered'}))
```

0.7
3.3.5 Hyper Rectangle Utility Functions

In many cases, it is not possible to define a utility mapping for every issue independently. We provide the utility function `HyperVolumeUtilityFunction` to handle this situation by allowing for representation of a set of nonlinear functions defined on arbitrary hypervolumes of the space of outcomes.

The simplest example is a nonlinear-function that is defined over the whole space but that nonlinearly combines several issues to calculate the utility.

For example the previous `NonLinearUtilityFunction` for the seller can be represented as follows:

```python
seller_utility = HyperVolumeUtilityFunction(outcome_ranges= [None],
    utilities= [lambda x: 2.0*x['price']/x['number of items'] - 0.5 * int(x['delivery'] == 'delivered')])
print(seller_utility({'price': 1.0, 'number of items': 3, 'delivery': 'not delivered'}))
print(seller_utility({'price': 1.0, 'number of items': 3, 'delivery': 'delivered'}))
print(seller_utility({'price': 1.8, 'number of items': 3, 'delivery': 'delivered'}))
```

This function recovered exactly the same values as the `NonlinearUtilityFunction` defined earlier by defining a single hypervolume with the special value of `None` which applies the function to the whole space and then defining a single nonlinear function over the whole space to implement the required utility mapping.

`HyperVolumeUtilityFunction` was designed to a more complex situation in which you can have multiple nonlinear functions defined over different parts of the space of possible outcomes.

Here is an example in which we combine one global utility function and two different local ones:

```python
f = HyperRectangleUtilityFunction(outcome_ranges= [None, {0: (1.0, 2.0), 1: (1.0, 2.0)}, {0: (1.4, 2.0), 2: (2.0, 3.0)}],
    utilities= [5.0, 2.0, lambda x: 2 * x[2] + x[0]],
    weights= [1,0.5,2.5])
```

There are three nonlinear functions in this example:

- A global function which gives a utility of 5.0 everywhere
- A local function which gives a utility of 2.0 to any outcome for which the first issue (issue 0) has a value between 1.0 and 2.0 and the second issue (issue 1) has a value between 1.0 and 2.0 which is represented as: `{0: (1.0, 2.0), 1: (1.0, 2.0)}`
- A second local function which gives a utility that depends on both the third and first issues (lambda x: 2 * x[2] + x[0]) on the range `{0: (1.4, 2.0), 2: (2.0, 3.0)}`.

You can also have weights for combining these functions linearly. The default is just to sum all values from these functions to calculate the final utility.

Here are some examples:

* An outcome that falls in the range of all constraints:

```python
f([1.5, 1.5, 2.5])
```

22.25

* An outcome that falls in the range of the global and first local constraints only:
6.0

- An outcome that misses a value for some of the issues:

```python
print(f([1.5, 1.5]))
```

```
None
```

Notice that in this case, no utility is calculated because we do not know if the outcome falls within the range of the second local function or not. To allow such cases, the initializer of `HyperVolumeUtilityFunction` allows you to ignore such cases:

```python
g = HyperRectangleUtilityFunction(outcome_ranges=[None,
                                                {0: (1.0, 2.0), 1: (1.0, 2.0)},
                                                {0: (1.4, 2.0), 2: (2.0, 3.0)}],
                                          utilities=[5.0, 2.0, lambda x: 2 * x[2] + x[0]],
                                          ignore_failing_range_utilities=True,
                                          ignore_issues_not_in_input=True)
```

```python
print(g([1.5, 1.5]))
```

```
7.0
```

### 3.3.6 Nonlinear Hyper Rectangle Utility Functions

`HyperVolumeUtilityFunction` should be able to handle most complex multi-issue utility evaluations but we provide a more general class called `NonLinearHyperVolumeUtilityFunction` which replaces the simple weighted summation of local/global functions implemented in `HyperVolumeUtilityFunction` with a more general nonlinear mapping.

The relation between `NonLinearHyperVolumeUtilityFunction` and `HyperVolumeUtilityFunction` is exactly the same as that between `NonLinearUtilityAggregationFunction` and `LinearUtilityAggregationFunction`.

### 3.4 Other utility function types

There are several other builtin utility function types in the utilities module. Operations for utility function serialization to and from xml as well as normalization, finding pareto-frontier, generation of ufuns, etc are also available. Please check the documentation of the utilities module for more details.

```python
from pprint import pprint
pprint(negmas.utilities.__all__)
```

```python
['UtilityDistribution',
 'UtilityValue',
 'UtilityFunction',
 'UtilityFunctionProxy',
 'ConstUFun',
 'LinDiscountedUFun',
 'ExpDiscountedUFun',
 'MappingUtilityFunction',
 'LinearUtilityAggregationFunction',
 'NonLinearUtilityAggregationFunction',
 'HyperRectangleUtilityFunction',
]
```

(continues on next page)
3.5 Negotiators

Negotiations are conducted by negotiators. We reserve the term Agent to more complex entities that can interact with a simulation or the real world and spawn Negotiator objects as needed (see the situated module documentation). The base Negotiator are implemented in the negotiators module. The design of this module tried to achieve maximum flexibility by relying mostly on Mixins instead of inheretance for adding functionality as will be described later.

Classes exposed in this module end with either Agent or Mixin

```python
import negmas; negmas.negotiators.__all__
```

[['Negotiator', 'NegotiatorProxy', 'AspirationMixin']

To build your negotiator, you need to inherit from one class ending with Negotiator, implement its abstract functions and then add whatever mixins you need implementing their abstract functions (if any) in turn.

Negotiators related to a specific negotiation mechanism are implemented in that mechanism’s module. For example, negotiators designed for the Stacked Alternating Offers Mechanism are found in the sao module.

3.5.1 Agent (the base class of all negotiation agents)

The base class of all agents is Agent which has three abstract methods that MUST be implemented by any agent you inherit from it:

```python
import negmas; negmas.sao.__all__
```


There is a special type of negotiators called GeniusNegotiator implemented in the genius module that is capable of interacting with negotiation sessions running in the genius platform (JVM). Please refer to the documentation of this module for more information.
3.6 Mechanisms (Negotiations)

The base Mechanism class is implemented in the mechanisms module.

All protocols in the package inherit from the Protocol class and provide the following basic functionalities:

• checking capabilities of agents against requirements of the protocol

• allowing agents to be join and leave the negotiation under the control of the underlying protocol. For example the protocol may allow or disallow agents from entering the negotiation once it started, it may allow or disallow modifying the issues being negotiated, may allow only a predefined maximum and minimum number of agents to engage in the negotiation. All of this is controlled through parameters to the protocol initializer.

• provide the basic flow of protocols so that new protocols can be implemented by just overriding a single round() function.

• provide basic callbacks that can be extended by new protocols.

The simplest way to use a protocol is to just run one of the already provided protocols. This is an example of a full negotiation session:

```python
p = SAOMechanism(outcomes = 6, n_steps = 10)
p.add(LimitedOutcomesNegotiator(name='seller', acceptable_outcomes=[(2,), (3,), (5,)], outcomes=p.outcomes))
p.add(LimitedOutcomesNegotiator(name='buyer', acceptable_outcomes=[(1,), (4,), (3,)], outcomes=p.outcomes))
state = p.run()
p.state.agreement
```

You can create a new protocol by overriding a single function in the Protocol class. This is for example the full code of the AlternatingOffersProtocol for the multi-issue case.

```python
class MyAlternatingOffersProtocol(Mechanism):
    def __init__(self, issues=None, outcomes=None, n_steps=None, time_limit=None):
        super().__init__(issues=issues, outcomes=outcomes, n_steps=n_steps, time_limit=time_limit)
        self.current_offer = None
        self.current_offerer = None
        self.n_accepting_agents = 0

    def step_(self):
        end_negotiation = False
        n_agents = len(self.negotiators)
        accepted = False
        for i, agent in enumerate(self.negotiators):
            if self.current_offer is None:
                response = ResponseType.NO_RESPONSE
            else:
                response = agent.respond(state=self.state, offer=self.current_offer)
            if response == ResponseType.END_NEGOTIATION:
                end_negotiation = True
                self.current_offer = None
            else:
                if response != ResponseType.ACCEPT_OFFER:
                    self.current_offer = agent.propose(state=self.state)
```
Agents can now engage in interactions with this protocol as easily as any built-in protocol:

```python
p = MyAlternatingOffersProtocol(outcomes = 6, n_steps = 10)
p.add(LimitedOutcomesNegotiator(name='seller', acceptable_outcomes=[(2,), (3,), (5,), (6,)], outcomes=p.outcomes))
p.add(LimitedOutcomesNegotiator(name='buyer', acceptable_outcomes=[(1,), (4,), (3,), (6,)], outcomes=p.outcomes))
state = p.run()
p.state.agreement
```

The negotiation ran with the expected results
This section provides a set of use cases showing the flexibility of the framework implemented by NegMAS and its applicability to a variety of problems.

## 4.1 Running a Negotiation

NegMAS has several built-in negotiation Mechanisms (Protocol s), negotiation agents (Negotiator s), and UtilityFunction s. You can use these to run negotiations as follows:

```python
import random  # for generating random ufuns
random.seed(0)  # for reproducibility
from pprint import pprint  # for printing
from negmas import SAOMechanism, AspirationNegotiator, MappingUtilityFunction

session = SAOMechanism(outcomes=10, n_steps=100)
negotiators = [AspirationNegotiator(name=f'a{4}{_}') for _ in range(5)]
for negotiator in negotiators:
    session.add(negotiator, ufun=MappingUtilityFunction(lambda x: random.random() \n    \*+ x[0]))

pprint(session.run().__dict__)
```

```python
{ 'agreement': (4,),
  'broken': False,
  'current_offer': (4,),
  'current_proposer': 'a4-3c56bcb2-0278-45ee-b95b-c696d6a11a54',
  'error_details': '',
  'has_error': False,
  'n_acceptances': 0,
  'n_negotiators': 5,
  'relative_time': 0.96,
  'running': False,
  'started': True,
  'step': 95,
  'time': 0.012729342999989512,
  'timedout': False}
```
Negotiations end with a status that shows you what happens. In the above example, we can see that the negotiation was not broken and did not time-out. The agreement was on outcome \((4,)\) of the 10 possible outcomes of this negotiation. That offer was offered by negotiator \(a0\) (the rest of the agent ID is always a random value to ensure no name repetitions) which was accepted by all of the other 4 negotiators.

It is possible to run the same negotiation using a `Protocol` object instead of a `Mechanism` object

```python
random.seed(0) # for reproducibility
from negmas import SAOProtocol

session = SAOProtocol(outcomes=10, n_steps=100)

negotiators = [AspirationNegotiator(name='a' + _) for _ in range(5)]

for negotiator in negotiators:
    session.add(negotiator, ufun=MappingUtilityFunction(lambda x: random.random() * x[0]))

pprint(session.run().__dict__)
```

As you can see, we got the same output. `Protocol` is an alias of `Mechanism` in NegMAS.

Let’s try a more meaningful situation: Assume we have a buyer and a seller who are negotiating about a business transaction in which the buyer wants to maximize his profit while the seller wants to minimize her cost. They both would like to transact on as much as possible of the product and each has some preferred delivery time.

This can be modeled in the following negotiation:

```python
from negmas import Issue, SAOMechanism, AspirationNegotiator, normalize
from negmas.utilities import LinearUtilityAggregationFunction as LUFun

issues = [Issue(name='price', values=10), Issue(name='quantity', values=10), Issue(name='delivery_time', values=10)]

session = SAOMechanism(issues=issues, n_steps=5)

buyer_utility = normalize(ufun=LUFun(issue_utilities={'price': lambda x: 9.0 - x, 'quantity': lambda x: 0.2 * x, 'delivery_time': lambda x: x}), outcomes=session.outcomes)

seller_utility = normalize(ufun=LUFun(issue_utilities={'price': lambda x: x, 'quantity': lambda x: 0.2 * x, 'delivery_time': lambda x: 9.0 - x}), outcomes=session.outcomes)

session.add(AspirationNegotiator(name='buyer'), ufun=buyer_utility)
session.add(AspirationNegotiator(name='seller'), ufun=seller_utility)
pprint(session.run().__dict__)
```
In this run, we can see that the agreement was on a high price (9) which is preferred by the seller but with a delivery time of (8) which is preferred by the buyer. Negotiation took all of the allowed 5 steps.

We can check the negotiation history as well:

```python
for i, _ in enumerate(session.history):
    print(f'{i:03}: {_.current_proposer} offered {_.current_offer}')
```

```
000: buyer-a7cd6341-d966-42d2-a206-2104750f56c5 offered {'price': 0, 'quantity': 9, 'delivery_time': 9}
001: seller-bbbebd4e-b6ea-44ed-a819-2e9d1740f85 offered {'price': 9, 'quantity': 7, 'delivery_time': 0}
002: buyer-a7cd6341-d966-42d2-a206-2104750f56c5 offered {'price': 2, 'quantity': 7, 'delivery_time': 9}
003: seller-bbbebd4e-b6ea-44ed-a819-2e9d1740f85 offered {'price': 9, 'quantity': 9, 'delivery_time': 9}
004: seller-bbbebd4e-b6ea-44ed-a819-2e9d1740f85 offered {'price': 9, 'quantity': 9, 'delivery_time': 8}
```

We can even plot the complete negotiation history and visually see how far were the result from the pareto frontier (it was 0.0 utility units far from it).

```python
session.plot(plot_outcomes=False)
```

What happens if the seller was much more interested in delivery time.

Firstly, what do you expect?

Given that delivery time becomes a more important issue now, the buyer will get more utility points by allowing the price to go up given that the delivery time can be made earlier. This means that we should expect the delivery time to go down in the agreement and the price to go up if it can. Let’s see what happens:
session = SAOMechanism(issues=issues, n_steps=5)

buyer_utility = normalize(ufun=LUFun(issue_utilities={
    'price': lambda x: x,
    'quantity': lambda x: 0.2 * x,
    'delivery_time': lambda x: 9.0 - x},
    weights={'price': 1.0, 'quantity': 1.0, 'delivery_time': 10.0}),
    outcomes=session.outcomes)

session.add(AspirationNegotiator(name='buyer'), ufun=buyer_utility)

session.add(AspirationNegotiator(name='seller'), ufun=seller_utility)

print(session.run().__dict__)

We can check it visually as well:

```
session.plot(plot_outcomes=False)
```

It is clear that the new ufuns transformed the problem. Now we have a single outcome at the pareto front. The agreement is on it (0.0 utility points).

What happens if we repeat the earlier neotiation but with a much longer negotiation time:

```
session = SAOMechanism(issues=issues, n_steps=2000)

buyer_utility = normalize(ufun=LUFun(issue_utilities={
    'price': lambda x: 9.0 - x,
    'quantity': lambda x: 0.2 * x,
    'delivery_time': lambda x: x}),
    outcomes=session.outcomes)

seller_utility = normalize(ufun=LUFun(issue_utilities={
    'price': lambda x: x,
    'quantity': lambda x: 0.2 * x,
    'delivery_time': lambda x: x}))
```

(continues on next page)
Given the longer negotiation time, the buyer and the seller can both take a tougher stance conceding as slowly as possible (taking 1644 steps out of the available 2000 instead of just 5 as before). As a result they again achieve a point exactly on the pareto-front. Moreover, this point happens to maximize the welfare defined as the sum of the utility received by both partners.

Download Notebook.

### 4.2 Develop a new negotiator

TBD

Download Notebook.

### 4.3 Develop a new mechanism (protocol)

TBD

Download Notebook.
4.4 Develop a new simulation (world)

TBD
Download Notebook.

4.5 Develop a new agent (for your simulation)

TBD
Download Notebook.

4.6 Run a session of the SCML world

The SCML world (Supply Chain Management League) ships as an example world with NegMAS. It is used as one of the leagues of ANAC 2019.

In this tutorial, you will test a run of this world.

Firstly, let’s import everything from the SCML app

```python
from negmas.apps.scml import *
```

There are several ways to create an SCML world. One of the simplest is to use one of the class methods of the SCMLWorld class.

The resulting world will have 100 production steps so simulating it will take several minutes. **To speed up the simulation, use a small number of n_steps (e.g. 10)**

If you want to just test your installation (and do not care whether you get an accurate indicator of agent performance), you can set the number of steps to a small value (e.g. n_steps=10).

```
world = SCMLWorld.chain_world(agent_names_reveal_type=True, n_steps=100, n_intermediate_levels=1, compact=True, log_file_name='')
```

For debugging purposes, we set agent_names_reveal_type to true to be able to know the type of an agent from its name. During the actual ANAC 2019 competition this is passed as False preventing agents from knowing the type of other agents by just inspecting their name. Now you can run this world simulation by just calling run.

```
world.run()
```

Let’s see what happened in this run. Firstly, how many negotiations were conducted over time

```python
plt.plot(world.stats['n_negotiations'])
plt.xlabel('Simulation Step')
plt.ylabel('N. Negotiations')
plt.show()
```
It is clear that many negotiations happened at the beginning of the simulation with smaller number later. That is expected as the consumers publish their calls for proposals early on.

Several other statistics are available:

```python
print(list(world.stats.keys()))
```

We can for example check the activity level of this world (defined as the total amount of money transferred due to trade)

```python
plt.plot(world.stats['activity_level'])
plt.xlabel('Simulation Step')
plt.ylabel('Activity Level ($)')
plt.show()
```
We can see a picture of contracting in this world as follows:

```python
plt.plot(world.stats['n_contracts_concluded'], label='Concluded Contracts')
plt.plot(world.stats['n_contracts_cancelled'], label='Cancelled Contracts')
plt.plot(world.stats['n_contracts_signed'], label='Signed Contracts')
plt.plot(world.stats['n_contracts_executed'], label='Executed Contracts')
plt.legend()
plt.xlabel('Simulation Step')
plt.ylabel('N. Contracts')
plt.show()
```

We can also check the breaches that happened:

```python
plt.plot(world.stats['breach_level'])
plt.xlabel('Simulation Step')
plt.ylabel('Total Breach Level')
plt.show()
```

It is also possible to focus on the behavior of some specific agent. Let’s first find who was the winner (the factory manager that got maximum wealth). Notice that there can be multiple winners:

```python
winner_profits = [round(100.0 * world.stats[f'balance_{_}'][-1] / world.stats[f'balance_{_}'][0] - 100.0, 2) for _ in world.winners]
print(f"{world.winners} won at {winner_profits}%")
```

[greedy@2_3] won at [47.3]%

What was the balance of this winner over time:

```python
for winner in world.winners:
    plt.plot(world.stats[f'balance_{winner}'], label=f'{winner.__class__.__name__}')
```

(continues on next page)
Let’s check how did its total storage change over time:

```python
for winner in world.winners:
    plt.plot(world.stats[f'storage_{winner}'], label=winner)
plt.xlabel('Simulation Step')
plt.ylabel('Player\'s Total Storage (item)')
plt.legend()
plt.show()
```

We can actually check what happens to ALL competitors:

```python
for name, _ in sorted(((_.name, _) for _ in world.factory_managers), key=lambda x: x[0]):
    plt.plot(world.stats[f'balance_{name}'], label=name)
plt.xlabel('Simulation Step')
plt.ylabel('Player Balance ($)')
plt.legend()
plt.show()
```
4.7 Develop a factory manager (agent) for the SCM world

This tutorial describes how to develop an agent for the SCM world, test it, and submit it to the ANAC 2019 SCM league.

The first step is to install negmas:

```
pip install negmas
```

Once you have this library installed, you can start developing your factory manager:

```python
from negmas.apps.scml import FactoryManager

try:
    class MyFactoryManager(FactoryManager):
        """My factory manager""
        f = MyFactoryManager()
except Exception as e:
    print(e)
```

You are told that you cannot instantiate your newly created class as did not implement the abstract (required) methods. These abstract methods are useful in giving you an idea of all the callback you should expect.

If you want some default behavior implemented for you, you can inherit from one of the provided factory managers like `DoNothingFactoryManager` and `GreedyFactoryManager`. In this case, you only need to override the functions you modify

```python
from negmas.apps.scml import DoNothingFactoryManager

class MyFactoryManager(DoNothingFactoryManager):
    """My factory manager""
```

As the documentation states, this function is called whenever your factory manager receives a request from another agent to negotiate. You can either return `None` if you do not want to accept this negotiation or create a `Negotiator` that represents your agent in it.

Your do-nothing agent is almost ready. Let’s try it:

```
32 Chapter 4. Tutorials
```
The property `stats` in `World` gives you several statistics about the world for every time-step of the simulation.

Let’s check the contracts of this world:

Let’s try to run a tournament with this do-nothing agent against the built-in greedy agent (in the “collusion” track setting):

```python
from negmas.apps.scml.utils import anac2019_collusion
from negmas.apps.scml import GreedyFactoryManager

results = anac2019_collusion(competitors=(MyFactoryManager, GreedyFactoryManager),
                               agent_names_reveal_type=True,
                               n_configs=2,  # create 10 different configs
                               max_worlds_per_config=4,  # create no more then 4
                               worlds_per_config=1,  # number of runs of each
                               configured_world=1,  # we are running each
                               world_for_50_steps_only=True)
```

You can see the scores that each individual factory manager got (just a random sample):

```python
results.scores.tail()
```

You can also check the total scores for each factory manager type:

```python
results.total_scores
```

If you want, you can check if these differences are statistically significant using a t-test:

```python
results.ttest
```

So the greedy factory manager is slightly better than the do-nothing factory manager for this short simulation getting an average gain of 1.1% compared with nothing (0%) for the do-nothing factory manager (as expected). Moreover, this difference is not statistically significant as the p-value is 0.333 > 0.05. If you try running this this tournament for less than 20, the greedy factory manager will most likely lose money. In the actual league, we will run each world simulation between 50 and 100 steps (more toward the later).

You can just check the winner(s) list:

```python
results.winners
```

This information and much more is also stored in a log folder that gives details of every world and total scores, etc. The default location of this log folder is under `negmas/logs/tournaments` in your HOME directory (this can be changed by passing a `tournament_path` to the `anac2019_tournament` function.

To run a tournament in the “standard”/“sabotage” track settings, use “anac2019_std”/“anac2019_sabotage” instead of “anac2019_collusion”.

This information and much more is also stored in a log folder that gives details of every world and total scores, etc. The default location of this log folder is under `negmas/logs/tournaments` in your HOME directory (this can be changed by passing a `tournament_path` to the `anac2019_tournament` function.

The information stored in this folder is:
### File /Folder Name | Format | Content
---|---|---
**base_config** | FOLDER | Contains one JSON file for each configuration tried during the tournament before assigning agents to factories.
**assigned_config** | FOLDER | Contains one JSON file for each configuration tried during the tournament after assigning agents to factories. You can re-run this world using the `run_world` function in the `tournament` module.
**params.json** | JSON | The parameters used to create this tournament.
**scores.csv** | CSV | Scores of every agent in every world.
**total_scores.csv** | CSV | Scores of every agent type averaged over all runs.
**winners.csv** | CSV | Winners and their average scores.
**ttes_tests.csv** | CSV | Results of a factorial TTEST comparing the performance of all agent types.

Other than these files, a folder with the same number as the corresponding config file in the configs folder, keeps full statistics/log of every world with the following contents:

| File Name | Format | Content |
---|---|---|
**all_contracts.csv** | CSV | A record of all contracts.
**contracts_full_info.csv** | CSV | A record of all contracts with added information about the CFPs.
**canceled_contracts.csv** | CSV | Contracts that were canceled because one partner refused to sign it.
**signed_contracts.csv** | CSV | Contracts that were actually signed.
**negotiations.csv** | CSV | A record of all negotiations.
**breaches.csv** | CSV | A record of all breaches.
**stats.csv** | CSV | Helpful statistics about the state of the world at every time step (e.g., `N_negotiation`, `N_ContractsExecuted`, etc.) in CSV format.
**stats.json** | JSON | Helpful statistics about the state of the world at every time step (e.g., `N_negotiation`, `N_ContractsExecuted`, etc.) in JSON format.
**params.json** | JSON | The arguments used to run the world.
**logs.txt** | TXT | A log file giving details of the most important events during the simulation.

To develop a more useful agent, you will need to override one or more of the available callbacks in `FactoryManager` and use methods available in the SCMLAWI (SCML Agent World Interface) to act in the world in order to maximize your profit.

### 4.7.1 Most important callbacks:

The most important callbacks that your class is expected to override to be useful as a factory manager are the following:

- **init()** Called after the world is initialized, but before any simulation steps.
• step() Called in the simulation loop. Simulates one step of the agent’s logic. You can use this call to be proactive.

• on_new_cfp() Called whenever a new Call for Proposals (CFP) is published on the bulletin board. The agent can specify a condition (e.g., buy CFPs only) such that only those CFPs that satisfy this condition will trigger this callback. By default your agent will only receive CFPs about products that it can use for production or can produce. You can override that by changing the interesting_products property of your agent (probably in init()). This callback can be used for implementing reactive behavior.

• on_cfp_removed() Called whenever a CFP is removed from the bulletin board.

• on_negotiation_request_accepted()/on_negotiation_request_rejected() Called when a negotiation request initiated by the agent is accepted/rejected.

• on_negotiation_success()/on_negotiation_failure() Called when a negotiation the agent is involved in terminates.

• sign_contract() Called by the simulator when it is time to sign a contract. The agent can refuse to sign. By default, agents sign the contract.

• on_contract_signed()/on_contract_cancelled() Called when a contract the agent is party to is signed/cancelled (contracts will be canceled if any of the partners party to it refused to sign it).

• on_production_failure() Called whenever a production command scheduled by the agent cannot be executed (e.g. for lack of funds or need of input products).

4.7.2 More details

You can download a skeleton for developing your factory manager in either python or java ‘here <http://www.yasserm.com/scml/scml.zip>__. For more details, refer to the detailed description of the SCM world and the Agent, SCMLAgent, and FactoryManager documentation at NegMAS library documentation

4.7.3 What can the agent do and know?

The agent can act by calling various methods of its awi member (Agent World Interface). The most important of these are:

• request_negotiation() Requests a negotiation with another partner

• register_interest() / unregister_interest By default the agent will receive on_*_cfp callbacks only on products that its factory consumes or produces. To override this behavior, you can use these two methods of the awi.

• register_cfp() / remove_cfp() Registers/removes a call for proposals indicating interest in buying/selling some product and giving the negotiation issues (e.g. deliver time, unit cost, quantity, penalty, signing delay).

• evaluate_insurance() / buy_insurance() Gets the insurance premium for some potential contract or buys one

• execute() Executes an action in the world. The only supported actions are scheduling a production process to run at some future time-step, stopping (or canceling) a previously issued run command.

The agent can also access some useful information through its awi’s properties. Some of the most important such properties are:

• state The state of the factory giving its current storage, cash in wallet, and standing loans as well as all scheduled production commands.

• n_steps World simulation length

• current_step Current world simulation step
• **products/processes** Information about products/processes defined in this world (these are also accessible through local properties of the `FactoryManager`)

• **cfps** All calls for proposals currently published in the bulletin board

• **breaches** All breaches currently published in the bulletin board

### 4.7.4 Participation in the ANAC 2019 SCM league

Now, you completed the development of your factory manager, tested it by running it in worlds and tournaments, what exactly should you do to participate in the SCM league @ ANAC 2019:

You need to submit the following items:

• Names of all members of the team with their affiliations and email addresses

• Either a single python file with the whole implementation of your agent with any supporting code or a zip file with a single folder containing your code. In the later case, you will need to indicate the class name of your factory manager. Any factory manager names are accepted except (Insurance, Bank, MFactoryManager, CFactoryManager).

• A 2-pages academic report about your factory manager. Please check the submission website for details about this report.

That is it folks! You can now start developing your own factory manager. Have fun.

You can download a skeleton for developing your factory manager in either python or java ‘here `<http://www.yasserm.com/scml/scml.zip>`__.

### 4.7.5 More Information

For more information, please refer to the links in the CFP

[Download Notebook](#)
This part of the documentation describes the basic modules that provide the core functionality of NegMAS.

## 5.1 negmas.outcomes Module

Defines basic concept related to outcomes

Outcomes in this package are always assumed to be multi-issue outcomes where single-issue outcomes can be implemented as the special case with a single issue.

- Both Continuous and discrete issues are supported. All issue will have names. If none is given, a random name will be used. It is HIGHLY recommended to always name your issues.
- Outcomes are dictionaries with issue names as keys and issue values as values.

### Examples

Different ways to create issues:

```python
>>> issues = [Issue((0.5, 2.0), 'price'), Issue(['2018.10.' + str(_ for _ in range(1, 4)], 'date')
... , Issue(20, 'count'))
>>> for _ in issues: print(_)
price: (0.5, 2.0)
date: ['2018.10.1', '2018.10.2', '2018.10.3']
count: 20
```

Outcome example compatible with the given set of issues:

```python
>>> a = {'price': 1.2, 'date': '2018.10.04', 'count': 4}
```

### 5.1.1 Functions
outcome_is_valid(outcome, issues)  Test validity of an outcome given a set of issues.

outcome_is_complete(outcome, issues)  Tests that the outcome is valid and complete.

outcome_range_is_valid(outcome_range[, issues])  Tests whether the outcome range is valid for the set of issues.

outcome_range_is_complete(outcome_range[, ...])  Tests whether the outcome range is valid and complete for the set of issues.

outcome_in_range(outcome, outcome_range, *)  Tests that the outcome is contained within the given range of outcomes.

enumerate_outcomes(issues[, keep_issue_names])  Enumerates all outcomes of this set of issues if possible.

sample_outcomes(issues[, n_outcomes, ...])  Discretizes the issue space and returns either a predefined number of outcomes or uniform samples.

outcome_as_dict(outcome[, issue_names])  Converts the outcome to a dict no matter what was its type.

outcome_as_tuple(outcome)  Converts the outcome to a tuple no matter what was its type.

num_outcomes(issues)  rtype Optional[int]

outcome_is_valid

negmas.outcomesoutcome_is_valid(outcome, issues)

Test validity of an outcome given a set of issues.

Examples

```python
>>> issues = [Issue((0.5, 2.0), 'price'), Issue(['2018.10.'+ str(_) for _ in range(1, 4)], 'date'), Issue(20, 'count')]
>>> for _ in issues: print(_)
price: (0.5, 2.0)
date: ['2018.10.1', '2018.10.2', '2018.10.3']
count: 20
>>> print([outcome_is_valid({'price':3.0}, issues), outcome_is_valid({'date': '2018.10.4'}, issues), outcome_is_valid({'count': 21}, issues)])
[False, False, False]
>>> valid_incomplete = {'price': 1.9}
>>> print(outcome_is_valid(valid_incomplete, issues))
True
>>> print(outcome_is_complete(valid_incomplete, issues))
False
>>> valid_incomplete.update({'date': '2018.10.2', 'count': 5})
>>> print(outcome_is_complete(valid_incomplete, issues))
True
```

Parameters

- **outcome** (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – outcome tested which can contain values for a partial set of issue values

- **issues** (Collection[Issue]) – issues

Returns

If return_problem is True then a second return value contains a string with reason of failure

Return type Union[bool, Tuple[bool, str]]
outcome_is_complete

negmas.outcomes.outcome_is_complete(outcome, issues)
Tests that the outcome is valid and complete.

Examples

```python
>>> issues = [Issue((0.5, 2.0), 'price'), Issue(['2018.10.'+ str(_)
˓→range(1, 4)], 'date'), Issue(20, 'count')]
>>> for _ in issues: print(_)
price: (0.5, 2.0)
date: ['2018.10.1', '2018.10.2', '2018.10.3']
count: 20
>>> print(outcome_is_complete({'price':3.0}, issues), outcome_is_complete(
˓→'date': '2018.10.4'), issues), outcome_is_complete({'count': 21},
˓→issues)))
[False, False, False]
>>> valid_incomplete = {'price': 1.9}
>>> print(outcome_is_complete(valid_incomplete, issues))
False
>>> valid_incomplete.update({'date': '2018.10.2', 'count': 5})
>>> print(outcome_is_complete(valid_incomplete, issues))
True
>>> invalid = {'price': 2000, 'date': '2018.10.2', 'count': 5}
>>> print(outcome_is_complete(invalid, issues))
False
>>> invalid = {'unknown': 2000, 'date': '2018.10.2', 'count': 5}
>>> print(outcome_is_complete(invalid, issues))
False
```

Args:
outcome: outcome tested which much contain valid values all issues if it is to be considered complete.
issues: issues

Returns

If return_problem is True then a second return value contains a string with reason of failure

Return type: Union[bool, Tuple[bool, str]]

outcome_range_is_valid

negmas.outcomes.outcome_range_is_valid(outcome_range, issues=None)
Tests whether the outcome range is valid for the set of issues.

Parameters

- outcome_range (Mapping[Union[int, str], Union[int, float, str, List[int], List[float], List[str], Tuple[int, int], Tuple[float, float], List[Tuple[Union[int, float], Union[int, float]]]]) –
- issues (Optional[Collection[Issue]]) –

Example

```python
>>> try:
...     outcome_range_is_valid({'price': (0, 10)})
... except NotImplementedeError:
(continues on next page)
```
... print('Not implemented')
Not implemented

Returns:

Return type Union[bool, Tuple[bool, str]]

outcome_range_is_complete

negmas.outcomes.outcome_range_is_complete(outcome_range, issues=None)

Tests whether the outcome range is valid and complete for the set of issues

Parameters

- **outcome_range** (Mapping[Union[int, str], Union[int, float, str, List[int], List[float], List[str], Tuple[int, int], Tuple[float, float], List[Tuple[Union[int, float], Union[int, float]]]]) –
- **issues** (Optional[Collection[Issue]]) –

Example

```python
>>> try:
...    outcome_range_is_complete({'price': (0, 10)})
... except NotImplementedError:
...    print('Not implemented')
Not implemented
```

Returns:

Return type Union[bool, Tuple[bool, str]]

outcome_in_range

negmas.outcomes.outcome_in_range(outcome, outcome_range, *, strict=False, fail_incomplete=False)

Tests that the outcome is contained within the given range of outcomes.

An outcome range defines a value or a range of values for each issue.

Parameters

- **outcome** (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – Outcome being tested
- **outcome_range** (Mapping[Union[int, str], Union[int, float, str, List[int], List[float], List[str], Tuple[int, int], Tuple[float, float], List[Tuple[Union[int, float], Union[int, float]]]]) – Outcome range being tested against
- **strict** – Whether to enforce that all issues in the outcome must be mentioned in the outcome_range
- **fail_incomplete** – If True then outcomes that do not specify a value for all keys in the outcome_range
- **be considered not falling within it. If False then these outcomes will be considered falling** (will) –
- **the range given that the values for the issues mentioned in the outcome satisfy the range** (within) –
### Examples

```python
>>> outcome_range = {'price': (0.0, 2.0), 'distance': [0.3, 0.4], 'type': ['a', 'b'], 'area': 3}
>>> outcome_range_2 = {'price': [(0.0, 1.0), (1.5, 2.0)], 'area': [(3, 4), (7, 9)]}
>>> outcome_in_range({'price':3.0}, outcome_range)
False
>>> outcome_in_range({'date': '2018.10.4'}, outcome_range)
True
>>> outcome_in_range({'date': '2018.10.4'}, outcome_range, strict=True)
False
>>> outcome_in_range({'area': 3}, outcome_range, fail_incomplete=True)
False
>>> outcome_in_range({'area': 3}, outcome_range)
True
>>> outcome_in_range({'type': 'c'}, outcome_range)
False
>>> outcome_in_range({'type': 'a'}, outcome_range)
True
>>> outcome_in_range({'date': '2018.10.4'}, outcome_range_2)
True
>>> outcome_in_range({'area': 3.1}, outcome_range_2)
True
>>> outcome_in_range({'area': 3}, outcome_range_2)
False
>>> outcome_in_range({'area': 5}, outcome_range_2)
False
>>> outcome_in_range({'price': 0.4}, outcome_range_2)
True
>>> outcome_in_range({'price': 0.4}, outcome_range_2, fail_incomplete=True)
False
>>> outcome_in_range({'price': 1.2}, outcome_range_2)
False
>>> outcome_in_range({'price': 0.4, 'area': 3.9}, outcome_range_2)
True
>>> outcome_in_range({'price': 0.4, 'area': 10}, outcome_range_2)
False
>>> outcome_in_range({'price': 1.2, 'area': 10}, outcome_range_2)
False
>>> outcome_in_range({'price': 1.2, 'area': 4}, outcome_range_2)
False
>>> outcome_in_range({'type': 'a'}, outcome_range_2)
True
>>> outcome_in_range({'type': 'a'}, outcome_range_2, strict=True)
False
>>> outcome_range = {'price': 10}
>>> outcome_in_range({'price': 10}, outcome_range)
True
>>> outcome_in_range({'price': 11}, outcome_range)
False
```

**Returns**: Success or failure

**Return type**: bool

**Remarks**: Outcome ranges specify regions in an outcome space. They can have any of the following conditions:
• A key/issue not mentioned in the outcome range does not add any constraints meaning that all values are acceptable except if strict == True. If strict == True then no value will be accepted for issues not in the outcome range.

• A key/issue with the value None in the outcome range means all values on this issue are acceptable. This is the same as having this key/issue removed from the outcome space.

• A key/issue with the value [] (empty list) accepts no outcomes.

• A key/issue with a single value means that it is the only one acceptable.

• A key/issue with a single 2-items tuple (min, max) means that any value within that range is acceptable.

• A key/issue with a list of values means an output is acceptable if it falls within the condition specified by any of the values in the list (list == union). Each such value can be a single value, a 2-items tuple or another list. Notice that lists of lists can always be combined into a single list of values.

**enumerate_outcomes**

```python
def enumerate_outcomes(issues, keep_issue_names=True):
    """Enumerates all outcomes of this set of issues if possible""
    """
    # Parameters
    # issues (Iterable[Issue]) --
    # keep_issue_names --

    # Return type
    Union[List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Dict[str, Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[int, str], Union[int, float, str, list]]]], None]

    # Returns
    List
```

**sample_outcomes**

```python
def sample_outcomes(issues, n_outcomes=None, keep_issue_names=True, min_per_dim=5, expansion_policy=None):
    """Discretizes the issue space and returns either a predefined number of outcomes or uniform samples""
    """
    # Parameters
    # issues (Iterable[Issue]) -- The issues describing the issue space to be discretized
    # n_outcomes (Optional[int]) -- If None then exactly min_per_dim bins will be used for every continuous dimension and all outcomes
    # be returned (will) --
    # keep_issue_names --
    # min_per_dim --
    # expansion_policy -- None or 'repeat' or 'null' or 'no'. If repeat, then some of the outcomes will be repeated
    # None or 'no' then no expansion will happen if the total number of outcomes is less than (if)
    # If 'null' then expansion will be with None values (n_outcomes.)--
```

42 Chapter 5. Base Modules
Return type  Optional[List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]], None]]]]

Returns  List

Examples

enumerate the whole space >>> issues = [Issue(values=(0, 1), name='Price'), Issue(values=['a', 'b'], name='Name')] >>> sample_outcomes(issues=issues) [{'Price': 0.0, 'Name': 'a'}, {'Price': 0.0, 'Name': 'b'}, {'Price': 0.25, 'Name': 'a'}, {'Price': 0.25, 'Name': 'b'}, {'Price': 0.5, 'Name': 'a'}, {'Price': 0.5, 'Name': 'b'}, {'Price': 0.75, 'Name': 'a'}, {'Price': 0.75, 'Name': 'b'}, {'Price': 1.0, 'Name': 'a'}, {'Price': 1.0, 'Name': 'b'}]

enumerate with sampling for very large space (we have 10 outcomes in the discretized space) >>> issues = [Issue(values=(0, 1), name='Price'), Issue(values=['a', 'b'], name='Name')] >>> sampled=sample_outcomes(issues=issues, n_outcomes=5) >>> len(sampled) 5 >>> len(set(tuple(_.values()) for _ in sampled)) 5

outcome_as_dict

negmas.outcomes.outcome_as_dict(outcome, issue_names=None)

Converts the outcome to a dict no matter what was its type

outcome_as_tuple

negmas.outcomes.outcome_as_tuple(outcome)

Converts the outcome to a tuple no matter what was its type

num_outcomes

negmas.outcomes.num_outcomes(issues)

Return type  Optional[int]

5.1.2 Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutcomeType()</td>
<td>A helper class allowing for definition of types that behave as outcomes (either in the form of dict or tuple).</td>
</tr>
<tr>
<td>OutcomeRange</td>
<td>alias of typing.Mapping</td>
</tr>
<tr>
<td>ResponseType</td>
<td>Possible answers to offers during negotiation.</td>
</tr>
<tr>
<td>Issue(values[, name])</td>
<td>Encodes an Issue.</td>
</tr>
<tr>
<td>Issues(**kwargs)</td>
<td>Encodes a set of Issues.</td>
</tr>
</tbody>
</table>

OutcomeType

class negmas.outcomes.OutcomeType

Bases: object

A helper class allowing for definition of types that behave as outcomes (either in the form of dict or tuple).

This class is intended to be used when a simple tuple or dict is not enough for describing an outcome (e.g. to use editor features like auto-completion of members). You simply define your class as a dataclass and add your fields to it then inherit from OutcomeType. As we do nothing in the __init__ function, that is compatible with python dataclasses.
Examples

```python
>>> from negmas import OutcomeType, Issue
>>> @dataclass
... class MyOutcome(OutcomeType):
...     price: float = 0.0
...     quantity: int = 0

You can use MyOutcome as an outcome directly or convert it to a tuple/dict for other functions

```python
>>> outcome = MyOutcome(price=2.0, quantity=3)
>>> outcome.price
2.0
>>> outcome['price']
2.0
>>> outcome.astuple()
(2.0, 3)
>>> outcome.asdict()
{'price': 2.0, 'quantity': 3}
```

You can also use outputs from issues to initialize your class

```python
>>> issues = [Issue(name='price', values=(0.0, 3.0)), Issue(name='quantity', values=10)]
>>> sample = Issue.sample(issues=issues, n_outcomes=1)[0]

```python
>>> outcome = MyOutcome(**sample)
```python
>>> outcome.price == outcome['price']
True
```

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>asdict()</code></td>
<td>Converts the outcome to a dict containing all fields</td>
</tr>
<tr>
<td><code>astuple()</code></td>
<td>Converts the outcome to a tuple where the order of items is the same as they are defined as fields</td>
</tr>
<tr>
<td><code>get(name[, default])</code></td>
<td>Acts like dict.get</td>
</tr>
<tr>
<td><code>keys()</code></td>
<td>Return type <strong>List[str]</strong></td>
</tr>
<tr>
<td><code>values()</code></td>
<td>Return type <strong>List[str]</strong></td>
</tr>
</tbody>
</table>

Methods Documentation

`asdict()`  
Converts the outcome to a dict containing all fields

`astuple()`  
Converts the outcome to a tuple where the order of items is the same as they are defined as fields

`get(name[, default]=None)`  
Acts like dict.get

`keys()`  

Return type **List[str]**

`values()`
**ResponseType**

```python
class negmas.outcomes.ResponseType
    Bases: enum.Enum

    Possible answers to offers during negotiation.
```

**Attributes Summary**

```
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT_OFFER</td>
<td>0</td>
</tr>
<tr>
<td>END_NEGOTIATION</td>
<td>2</td>
</tr>
<tr>
<td>NO_RESPONSE</td>
<td>3</td>
</tr>
<tr>
<td>REJECT_OFFER</td>
<td>1</td>
</tr>
</tbody>
</table>
```

**Attributes Documentation**

- **ACCEPT_OFFER** = 0
- **END_NEGOTIATION** = 2
- **NO_RESPONSE** = 3
- **REJECT_OFFER** = 1

**Issue**

```python
class negmas.outcomes.Issue(values, name=None)
    Bases: negmas.common.NamedObject

    Encodes an Issue.
```

**Parameters**

- `values` (Union[List[str], int, Tuple[float, float]]) – Possible values for the issue
- `name` (Optional[str]) – Name of the issue. If not given, a random name will be generated

**Examples**

```python
>>> print(Issue(['to be', 'not to be'], name='THE problem'))
THE problem: ['to be', 'not to be']

>>> print(Issue(3, name='Cars'))
Cars: 3

>>> print(Issue((0.0, 1.0), name='required accuracy'))
required accuracy: (0.0, 1.0)

>>> a = Issue((0.0, 1.0), name='required accuracy')

>>> a.is_continuous()
True

>>> a.is_discrete()
False
```

**Remarks:**

- Issues can be initialized by either an iterable of strings, an integer or a tuple of two real values with the following meanings: *iterable of strings:* This is an issue that can any value within the
given set of values (strings) - int: This is an issue that takes any value from 0 to the given value -1
(int) - float: This is an issue that can take any real value between the given limits (min, max)

Attributes Summary

<table>
<thead>
<tr>
<th>all</th>
<th>A generator that generates all possible values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID of this entity.</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>outcome_range</td>
<td>An outcome range that represents the full space of the issues</td>
</tr>
<tr>
<td>type</td>
<td>The type of the issue.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity.</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>ali([n])</th>
<th>A generator that generates all possible values or samples n values for real Issues.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality</td>
<td>The number of possible outcomes for the issue.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>enumerate(issues, max_n_outcomes, astype)</td>
<td>rtype List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]]]]]</td>
</tr>
</tbody>
</table>

| from_genius(file_name[, force_single_issue, ...]) | Imports a the domain issues from a GENIUS XML file. |
| from_java(d, class_name) | rtype Issue |

| from_xml_str(xml_str[, force_single_issue, ...]) | Exports a list/dict of issues from a GENIUS XML file. |
| generate(issues[, counts, names]) | Generates a set of issues with given parameters. |
| is_continuous() | Test whether the issue is a continuous issue |
| is_discrete() | Test whether the issue is a discrete issue |
| n_outcomes(issues) | Returns the total number of outcomes in a set of issues. |
| rand() | Picks a random valid value. |
| rand_invalid() | Pick a random invalid value |
| rand_outcomes(n[, with_replacement, ...]) | Picks a random valid value. |
| rand_valid() | Picks a random valid value. |
| sample(issues, n_outcomes[, astype, ...]) | Samples some outcomes from the issue space defined by the list of issues |
| to_genius(issues, file_name[, enumerate_integer]) | Exports a the domain issues to a GENIUS XML file. |
| to_java() | |
| to_xml_str(issues[, enumerate_integer]) | Converts the list of issues into a well-formed xml string |
Attributes Documentation

all
A generator that generates all possible values.

Remarks:
• This function returns a generator for the case when the number of values is very large.
• If you need a list then use something like:

```python
>>> from negmas.outcomes import Issue
>>> list(Issue(5).all)
[0, 1, 2, 3, 4]
```

Return type Generator [+T_co, -T_contra, +V_co]

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

outcome_range
An outcome range that represents the full space of the issues

```python
Return type Mapping[Union[int, str], Union[int, float, str, List[int],
    List[float], List[str], Tuple[int, int], Tuple[float, float],
    List[Tuple[Union[int, float], Union[int, float]]]]]
```

type
The type of the issue.

Returns either ‘continuous’ or ‘discrete’

Return type str

uuid
The unique ID of this entity

Methods Documentation

alli (n=10)
A generator that generates all possible values or samples n values for real Issues.

Remarks:
• This function returns a generator for the case when the number of values is very large.
• If you need a list then use something like:

```python
>>> from negmas.outcomes import Issue
>>> list(Issue(5).alli)
[0, 1, 2, 3, 4]
```

Return type Generator [+T_co, -T_contra, +V_co]

cardinality()
The number of possible outcomes for the issue. A negative number means infinite

Return type int

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.
classmethod enumerate(issues, max_n_outcomes=None, astype=<class 'dict'>)

Return type List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]

classmethod from_genius(file_name, force_single_issue=False, force_numeric=False, keep_value_names=True, keep_issue_names=True, safe_parsing=True, n_discretization=None, max_n_outcomes=1000000.0)

Imports a the domain issues from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A List[Issue] or Dict[Issue]

Examples

```python
>>> issues, _ = Issue.from_genius(file_name = pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-domain.xml'))
>>> print([_.name for _ in issues])
['Laptop', 'Harddisk', 'External Monitor']
```

Remarks: See from_xml_str for all the parameters

classmethod from_java(d, class_name)

Return type Issue

classmethod from_xml_str(xml_str, force_single_issue=False, force_numeric=False, keep_value_names=True, keep_issue_names=True, safe_parsing=True, n_discretization=None, max_n_outcomes=1000000.0)

Exports a list/dict of issues from a GENIUS XML file.

Parameters

• xml_str (str) – The string containing GENIUS style XML domain issue definitions

• force_single_issue (bool) – Tries to generate a MappingUtility function with a single issue which is the product of all issues in the input

• keep_value_names (bool) – Keep names of values

• keep_issue_names (bool) – Keep names of issues

• safe_parsing (bool) – Turn on extra checks

• n_discretization (Optional[int]) – If not None, real valued issues are discretized with the given number of values

• max_n_outcomes (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns

• List[Issue] The issues (note that issue names will be stored in the name attribute of each issue if keep_issue_names)

• List[dict] A list of agent information dicts each contains ‘agent’, ‘class’, ‘utility_file_name’
Examples

```python
>>> domain_file_name = pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-domain.xml')
>>> issues, _ = Issue.from_xml_str(open(domain_file_name, 'r').read(), force_single_issue=True, keep_value_names=False, keep_issue_names=False)
>>> issue = issues[0]
>>> print(issue.cardinality())
27
>>> print(list(issue.all))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ... 21, 22, 23, 24, 25, 26]

>>> issues, _ = Issue.from_xml_str(open(domain_file_name, 'r').read(), force_single_issue=True, keep_value_names=False, keep_issue_names=True)
>>> print(issues[0].name)
Laptop-Harddisk-External Monitor
>>> print(len(issues))
1
>>> print(list(issues[0].all))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ... 21, 22, 23, 24, 25, 26]

>>> issues, _ = Issue.from_xml_str(open(domain_file_name, 'r').read(), force_single_issue=True, keep_value_names=True, keep_issue_names=False)
>>> issue = issues[0]
>>> print(issue.cardinality())
27
>>> print('
'.join(list(issue.all)[:5]))
Dell+60 Gb+19'' LCD
Dell+60 Gb+20'' LCD
Dell+60 Gb+23'' LCD
Dell+80 Gb+19'' LCD
Dell+80 Gb+20'' LCD
```

5.1. negmas.outcomes Module
\[\text{[3, 3, 3]}\]

```python
>>> print('\\n'.join([' '.join(list(issue.all)) for issue in issues]))
Dell Macintosh HP
60 Gb 80 Gb 120 Gb
19'' LCD 20'' LCD 23'' LCD
```

```python
>>> issues, _ = Issue.from_xml_str(open(domain_file_name, 'r').read(), ... force_single_issue=False, keep_issue_names=False, keep_value_names=False)
```

```python
>>> len(issues)
3
```

```python
>>> type(issues)
<class 'list'>
```

```python
>>> str(issues[0]).split(': ')[-1]
'3'
```

```python
>>> print([_.cardinality() for _ in issues])
[3, 3, 3]
```

```python
>>> domain_file_name = pkg_resources.resource_filename('negmas' ... , resource_name='tests/data/fuzzyagent/single_issue_domain.xml')
```

```python
>>> issues, _ = Issue.from_xml_str(open(domain_file_name, 'r').read(), ... force_single_issue=False, keep_issue_names=False, keep_value_names=False)
```

```python
>>> len(issues)
1
```

```python
>>> type(issues)
<class 'list'>
```

```python
>>> str(issues[0]).split(': ')[-1]
'(10.0, 40.0)'
```

```python
>>> print([_.cardinality() for _ in issues])
[-1]
```

**static generate** (issues, counts=None, names=None)

Generates a set of issues with given parameters. Each is optionally repeated

**Parameters**

- **issues** (Sequence[Union[int, List[str], Tuple[float, float]]]) – The parameters of the issues
- **counts** (Optional[Sequence[int]]) – The number of times to repeat each of the issues
- **names** (Optional[Sequence[str]]) – The names to assign to the issues. If None, then string representations of integers starting from zero will be used.

**Returns** The list of issues with given conditions

**Return type** List['Issue']

**is_continuous** ()

Test whether the issue is a continuous issue

**Returns** continuous or not

**Return type** bool

**is_discrete** ()

Test whether the issue is a discrete issue

**Returns** discrete or not

**Return type** bool
static n_outcomes(issues)
    Returns the total number of outcomes in a set of issues. -1 indicates infinity

    Return type int

rand()
    Picks a random valid value.

    Return type Union[int, float, str]

rand_invalid()
    Pick a random invalid value

    Return type Union[int, float, str]

rand_outcomes (n, with_replacement=False, fail_if_not_enough=False)
    Picks a random valid value.

    Return type Iterable[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]

rand_valid()
    Picks a random valid value.

    Return type Union[int, float, str]

classmethod sample(issues, n_outcomes, astype=<class 'dict'>, with_replacement=True, fail_if_not_enough=True)
    Samples some outcomes from the issue space defined by the list of issues

    Parameters
        • issues (Collection[Issue]) – List of issues to sample from
        • n_outcomes (int) – The number of outcomes required
        • astype (Type[+CT_co]) – The type of the outcomes. It can be tuple, dict or any OutcomeType
        • with_replacement (bool) – Whether sampling is with replacement (allowing repetition)
        • fail_if_not_enough – IF given then an exception is raised if not enough outcomes are available

    Return type List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]

    Returns a list of outcomes

Examples

```python
>>> from negmas import Issue, OutcomeType
>>> issues = [Issue(name='price', values=(0.0, 3.0)), Issue(name='quantity', values=10)]
```

Sampling outcomes as tuples

```python
>>> samples = Issue.sample(issues=issues, n_outcomes=10, astype=tuple)
>>> len(samples) == 10
True
>>> type(samples[0]) == tuple
True
```

Sampling outcomes as dicts

```python
```
```python
going from genius(file_name = pkg_resources.resource_filename('negmas','Laptop/Laptop-C-domain.xml'))
>>> Issue.to_genius(issues=issues, file_name = pkg_resources.resource_filename('negmas','LaptopConv/Laptop-C-domain.xml'))
>>> issues2, _ = Issue.from_genius(file_name = pkg_resources.resource_filename('negmas','LaptopConv/Laptop-C-domain.xml'))
>>> print('
'.join([' '.join(list(issue.all)) for issue in issues]))
Dell Macintosh HP
60 Gb 80 Gb 120 Gb
19'' LCD 20'' LCD 23'' LCD
>>> print('
'.join([' '.join(list(issue.all)) for issue in issues2]))
Dell Macintosh HP
60 Gb 80 Gb 120 Gb
19'' LCD 20'' LCD 23'' LCD
```

- Forcing Single outcome
>>> issues, _ = Issue.from_genius(file_name = pkg_resources.resource_filename('negmas'...
˓→filename('negmas'...
˓→Laptop/Laptop-C-domain.xml'),...  
˓→force_single_issue=True, keep_value_names=False, keep_issue_names=False)

>>> print(list(issues[0].all))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
˓→21, 22, 23, 24, 25, 26]

>>> issues.to_genius(issues=issues, enumerate_integer=True, ...
˓→filename = pkg_resources.resource_filename('negmas'...
˓→LaptopConv/Laptop-C-domain.xml'))

>>> issues3, _ = Issue.from_genius(file_name=pkg_resources.resource_filename('negmas'...
˓→filename('negmas'...
˓→LaptopConv/Laptop-C-domain.xml'))

>>> print([list(issue.all) for issue in issues3])
[['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', ...
˓→'14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', ...
˓→'26']]

Remarks: See from_xml_str for all the parameters
to_java()
classmethod to_xml_str(issues, enumerate_integer=True)
   Converts the list of issues into a well-formed xml string

Examples

>>> issues = [Issue(values=10, name='i1'), Issue(values=['a', 'b', 'c'], ...
˓→name='i2'), ...
˓→Issue(values=(2.5, 3.5), name='i3')]

>>> s = Issue.to_xml_str(issues)

>>> print(s.strip())
<negotiation_template>
<utility_space number_of_issues="3">
    <objective description="" etype="objective" index="0" name="root" type="" objective">
        <issue etype="discrete" index="1" name="i1" type="discrete" vtype="" discrete">
            <item index="1" value="0" cost="0" description="0" type="" discrete">
            </item>
            <item index="2" value="1" cost="0" description="1" type="" discrete">
            </item>
            <item index="3" value="2" cost="0" description="2" type="" discrete">
            </item>
            <item index="4" value="3" cost="0" description="3" type="" discrete">
            </item>
            <item index="5" value="4" cost="0" description="4" type="" discrete">
            </item>
            <item index="6" value="5" cost="0" description="5" type="" discrete">
            </item>
            <item index="7" value="6" cost="0" description="6" type="" discrete">
            </item>
            <item index="8" value="7" cost="0" description="7" type="" discrete">
            </item>
            <item index="9" value="8" cost="0" description="8" type="" discrete">
            </item>
        </issue>
    </objective>
</utility_space>

(continues on next page)
<item index="10" value="9" cost="0" description="9">
</item>
</issue>

<i etype="discrete" index="2" name="i2" type="discrete" vtype="discrete">
<item index="1" value="a" cost="0" description="a">
</item>
<item index="2" value="b" cost="0" description="b">
</item>
<item index="3" value="c" cost="0" description="c">
</item>
</issue>

<i etype="real" index="3" name="i3" type="real" vtype="real">
<range lowerbound="2.5" upperbound="3.5"></range>
</issue>
</objective>
</utility_space>
</negotiation_template>

```python
>>> issues2, _ = Issue.from_xml_str(s)
>>> print([_.__class__.__name__ for _ in issues2])
['Issue', 'Issue', 'Issue']
>>> print(len(issues2))
3
>>> print([str(_).split('
') for _ in issues2])
['i1: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']', 'i2: ['a', 'b', 'c']', 'i3: (2.5, 3.5)']
>>> print([_.values for _ in issues2])
[['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], ['a', 'b', 'c'], (2.5, 3.5)]
```

Return type str

Issues

```python
class negmas.outcomes.Issues(**kwargs)
    Bases: object

    Encodes a set of Issues.

    Parameters pairs (name-value)–

    Remarks:
    - Issues can be initialized by either an iterable of strings, an integer or a tuple of two real values with
    the following meanings: - iterable of strings: This is an issue that can any value within the
    given set of values (strings) - int: This is an issue that takes any value from 0 to the given value -1
    (int) - float: This is an issue that can take any real value between the given limits (min, max)

    Attributes Summary

<table>
<thead>
<tr>
<th>all</th>
<th>A generator that generates all possible values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>outcome_range</td>
<td>An outcome range that represents the full space of the issues</td>
</tr>
<tr>
<td>types</td>
<td>The type of the issue.</td>
</tr>
</tbody>
</table>
```

54 Chapter 5. Base Modules
Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cardinality()</code></td>
<td>The number of possible outcomes for the issue.</td>
</tr>
<tr>
<td><code>from_issue_collection(issues, [name])</code></td>
<td></td>
</tr>
<tr>
<td><code>from_single_issue(issue, [name])</code></td>
<td></td>
</tr>
<tr>
<td><code>is_finite()</code></td>
<td>Test whether all issues are discrete (finite outcome space)</td>
</tr>
<tr>
<td><code>is_infinite()</code></td>
<td>Test whether any issue is continuous (infinite outcome space)</td>
</tr>
<tr>
<td><code>n_outcomes()</code></td>
<td>Returns the total number of outcomes in a set of issues.</td>
</tr>
<tr>
<td><code>rand()</code></td>
<td>Picks a random valid value.</td>
</tr>
<tr>
<td><code>rand_invalid()</code></td>
<td>Pick a random invalid value</td>
</tr>
<tr>
<td><code>rand_valid()</code></td>
<td>Picks a random valid value</td>
</tr>
</tbody>
</table>

Attributes Documentation

**all**
A generator that generates all possible values.

**Remarks:**
- This function returns a generator for the case when the number of values is very large.
- If you need a list then use something like:

  ```python
  Return type  Generator[+T_co, -T_contra, +V_co]
  ```

**outcome_range**
An outcome range that represents the full space of the issues.

  ```python
  Return type  Mapping[Union[int, str], Union[int, float, str, List[int], List[float], List[str], Tuple[int, int], Tuple[float, float], List[Tuple[Union[int, float], Union[int, float]]]]]
  ```

**types**
The type of the issue.

  ```python
  Returns  either ‘continuous’ or ‘discrete’
  Return type  str
  ```

Methods Documentation

**cardinality()**
The number of possible outcomes for the issue. A negative number means infinite.

  ```python
  Return type  int
  ```

**classmethod from_issue_collection(issues, name=None)**

**classmethod from_single_issue(issue, name=None)**

**is_finite()**
Test whether all issues are discrete (finite outcome space).

  ```python
  Returns  discrete or not
  Return type  bool
  ```

**is_infinite()**
Test whether any issue is continuous (infinite outcome space).
**Returns**  continuous or not

**Return type**  bool

**n_outcomes()**
Returns the total number of outcomes in a set of issues. -1 indicates infinity

**Return type**  int

**rand()**
Picks a random valid value.

**Return type**  Dict[str, Union[int, float, str]]

**rand_invalid()**
Pick a random invalid value

**Return type**  Dict[str, Union[int, float, str]]

**rand_valid()**
Picks a random valid value.

**Return type**  Dict[str, Union[int, float, str]]

### 5.1.3 Class Inheritance Diagram

```
OutcomeType

Issues

NamedObject → Issue

Enum → ResponseType
```

### 5.2 negmas.utilities Module

Models basic types of utility functions.
Utility functions are at the core of negotiation. Agents engage in negotiations with the goal of maximizing some utility function. In most cases, these utility functions are assumed to be known a-priori and static for the duration of a single negotiation.

Notes

We try to allow for applications that do not necessarily have these two assumptions in the following ways:

- A utility function value (UtilityValue) can always represent a utility function distribution over all possible utility function values (UtilityDistribution) or a KnownUtilityValue which is a real number.
- The base class of all utility function functions is UtilityFunction and is assumed to map outcomes (Outcome objects) to the aforementioned generic utility values (UtilityValue objects).
- Utility functions can be constructed using any Callable which makes it possible to construct them so that they change depending on the context or the progression of the negotiation.

5.2.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pareto_frontier(ufuns[, outcomes, issues,...])</td>
<td>Finds all pareto-optimal outcomes in the list</td>
</tr>
<tr>
<td>make_discounted_ufun(ufun, am[,....])</td>
<td></td>
</tr>
<tr>
<td>normalize(ufun, outcomes[, rng, epsilon,...])</td>
<td>Normalizes a utility function to the range [0, 1]</td>
</tr>
</tbody>
</table>

**pareto_frontier**

```python
negmas.utilities.pareto_frontier(ufuns, outcomes=None, issues=None, n_discretization=10, sort_by_welfare=False)
```

Finds all pareto-optimal outcomes in the list

**Parameters**

- `ufuns` (Iterable[UtilityFunction]) – The utility functions
- `outcomes` (Optional[Iterable[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]]) – the outcomes to be checked. If None then all possible outcomes from the issues will be checked
- `issues` (Optional[Iterable[Issue]]) – The set of issues (only used when outcomes is None)
- `n_discretization` (Optional[int]) – The number of items to discretize each real-dimension into
- `sort_by_welfare` – If True, the results are sorted descendingly by total welfare

**Return type** Tuple[List[Tuple[float]], List[int]]

**Returns** Two lists of the same length. First list gives the utilities at pareto frontier points and second list gives their indices
make_discounted_ufun

```python
def make_discounted_ufun(ufun, ami, cost_per_round=None, power_per_round=None, discount_per_round=None, cost_per_relative_time=None, power_per_relative_time=None, discount_per_relative_time=None, cost_per_real_time=None, power_per_real_time=None, discount_per_real_time=None, dynamic_reservation=True):
```

normalize

```python
def normalize(ufun, outcomes, rng=(0.0, 1.0), epsilon=1e-06, infeasible_cutoff=None):
    Normalizes a utility function to the range \([0, 1]\)

    Parameters
    
    • `ufun` (*UtilityFunction*) – The utility function to normalize
    • `outcomes` (*Collection*) – A collection of outcomes to normalize for
    • `rng` (*Tuple*) – range to normalize to. Default is \([0, 1]\)
    • `epsilon` (*float*) – A small number specifying the resolution
    • `infeasible_cutoff` (*Optional*) – A value under which any utility is considered infeasible and is not used in normalization

    Returns
    A utility function that is guaranteed to be normalized for the set of given outcomes
```

5.2.2 Classes

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UtilityDistribution</td>
<td>alias of negmas.helpers.Distribution</td>
</tr>
<tr>
<td>UtilityFunction</td>
<td>The abstract base class for all utility functions.</td>
</tr>
<tr>
<td>ConstUFun</td>
<td><code>ConstUFun(value[, name, reserved_value, ami])</code></td>
</tr>
<tr>
<td>LinDiscountedUFun</td>
<td><code>LinDiscountedUFun(ufun, ami[, cost, factor, ...])</code></td>
</tr>
<tr>
<td>ExpDiscountedUFun</td>
<td><code>ExpDiscountedUFun(ufun, ami[, beta, factor, ...])</code></td>
</tr>
<tr>
<td>MappingUtilityFunction</td>
<td><code>MappingUtilityFunction(mapping[, default, ...])</code></td>
</tr>
<tr>
<td>LinearUtilityAggregationFunction</td>
<td><code>LinearUtilityAggregationFunction(issue_utilities) utility function for multi-issue negotiations.</code></td>
</tr>
<tr>
<td>NonLinearUtilityAggregationFunction</td>
<td><code>NonLinearUtilityAggregationFunction(...)</code></td>
</tr>
<tr>
<td>HyperRectangleUtilityFunction</td>
<td><code>HyperRectangleUtilityFunction(...[, ...])</code></td>
</tr>
<tr>
<td>NonlinearHyperRectangleUtilityFunction</td>
<td><code>NonlinearHyperRectangleUtilityFunction(...[, ...])</code></td>
</tr>
<tr>
<td>ComplexWeightedUtilityFunction</td>
<td><code>ComplexWeightedUtilityFunction(ufuns[, ...])</code></td>
</tr>
<tr>
<td>ComplexNonlinearUtilityFunction</td>
<td><code>ComplexNonlinearUtilityFunction(ufuns[, ...])</code></td>
</tr>
</tbody>
</table>

Continued on next page
Table 10 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IPUtilityFunction()</code></td>
<td>Independent Probabilistic Utility Function.</td>
</tr>
<tr>
<td><code>JavaUtilityFunction()</code></td>
<td>A utility function implemented in Java</td>
</tr>
<tr>
<td><code>RandomUtilityFunction()</code></td>
<td>A random utility function for a discrete outcome space</td>
</tr>
</tbody>
</table>

UtilityFunction

```python
class negmas.utilities.UtilityFunction(name=None, ami=None, reserved_value=None):
    Bases: abc.ABC, negmas.common.NamedObject
```

The abstract base class for all utility functions.

A utility function encapsulates a mapping from outcomes to UtilityValue(s). This is a generalization of standard utility functions that are expected to always return a real-value. This generalization is useful for modeling cases in which only partial knowledge of the utility function is available.

**Parameters**

- `name (str)` – Name of the utility function. If None, a random name will be given.
- `reserved_value (float)` – The value to return if the input offer to apply is None

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(offer)</code></td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td><code>approximate(ufuns, issues, n_outcomes[, ...])</code></td>
<td><strong>param cls</strong></td>
</tr>
<tr>
<td><code>compare(o1, o2)</code></td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td><code>conflict_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufunns</td>
</tr>
<tr>
<td><code>create(*args, **kwargs)</code></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>eu(offer)</code></td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td><code>from_genius(file_name, **kwargs)</code></td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>from_xml_str(xml_str[, domain_issues, ...])</code></td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td><code>generate_bilateral(outcomes[, ...])</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>generate_random(n, outcomes[, normalized])</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>generate_random_bilateral(outcomes)</code></td>
<td>Generates a couple of utility functions</td>
</tr>
</tbody>
</table>

Continued on next page
Table 12 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to_genius(u, issues, file_name, **kwargs)</code></td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>to_xml_str(u, issues[, discount_factor])</code></td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td><code>winwin_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td><code>xml(issues)</code></td>
<td>Converts the function into a well formed XML string preferably in GENIUS format.</td>
</tr>
</tbody>
</table>

Attributes Documentation

**base_type**

Returns the utility_function base type ignoring discounting and similar wrappings.

Return type: `str`

**id**

The unique ID of this entity

**is_dynamic**

Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is `False`, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**type**

Returns the utility_function type.

Each class inheriting from this `UtilityFunction` class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).__type__)  
linear_aggregation
>>> print(MappingUtilityFunction(lambda x: x).__type__)  
mapping
>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: -x).__type__)  
non_linear_aggregation
```

Returns utility_function type

Return type: `str`

**uuid**

The unique ID of this entity

Methods Documentation

```python
__call__(offer)
```

Calculate the utility_function value for a given outcome.
Parameters **offer** (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Remarks:

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.

- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns The utility_function value which may be a distribution. If `None` it means the utility_function value cannot be calculated.

Return type UtilityValue

**classmethod approximate**(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters

- cls –
- ufuns (List[UtilityFunction]) –
- issues (Iterable[Issue]) –
- n_outcomes (int) –
- min_per_dim –
- force_single_issue –

Returns:

Return type Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

**compare**(o1, o2)

Compares the two outcomes and returns a measure of the difference between their utilities

Return type Union[Distribution, float]

**classmethod conflict_level**(u1, u2, outcomes, max_tests=10000)

Finds the conflict level in these two ufuns

Parameters

- u1 (UtilityFunction) –
- u2 (UtilityFunction) –

Examples

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.randn(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

- The same ufun
A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, \len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, \len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type `float`

classmethod `create` (*args, **kwargs)

Creates an object and returns a proxy to it.

eu(offer)

Calculate the expected utility function value.

Parameters `offer` (Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]]]]) – The offer to be evaluated.

Returns The expected utility function for utility_priors and just utility_function for real-valued utilities.

Return type `float`

classmethod `from_genius` (file_name, **kwargs)

Imports a utility function from a GENIUS XML file.

Parameters `file_name` (str) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', ...
.. Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See from_xml_str for all the parameters

classmethod `from_xml_str` (xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)

Imports a utility function from a GENIUS XML string.

Parameters
• `xml_str (str)` – The string containing GENIUS style XML utility function definition

• `domain_issues (List(Issue))` – Optional issue space to confirm that the utility function is valid

• `force_single_issue (bool)` – Tries to generate a MappingUtility function with a single issue which is the

• `of all issues in the input (product)` –

• `keep_issue_names (bool)` – Keep names of issues

• `keep_value_names (bool)` – Keep names of values

• `safe_parsing (bool)` – Turn on extra checks

• `normalize_utility (bool)` – Normalize the output utilities to the range from 0 to 1

• `max_n_outcomes (int)` – Maximum number of outcomes allowed (effective only if `force_single_issue` is True)

**Returns** A utility function object (depending on the input file)

**Examples**

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C.prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C.prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD"',)) - 21.987727736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD"',)) - 22.68559475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C.prof1.xml'), 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.987727736172488) < 0.000001
```

(continues on next page)
```python
>>> assert abs(u(('Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD")) - 21.987727736172488) < 0.000001
>>> assert abs(u(('Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD")) - 22.6859475583014) < 0.000001
```

classmethod generate_bilateral(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

Parameters

- **n_outcomes** (int) – number of outcomes to use
- **conflict_level** (float) – How conflicting are the two ufun to generate. 1.0 means maximum conflict.
- **conflict_delta** – How variable is the conflict at different outcomes.
- **zero_summness** – How zero-sum like are the two ufun.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.5)
```

>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0)
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5)
... , conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True

Return type  Tuple[UtilityFunction, UtilityFunction]

classmethod generate_random(n, outcomes, normalized=True)
Generates a couple of utility functions

Parameters

• n (int) – number of utility functions to generate
• outcomes (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]]], Dict[Union[int, str], Union[int, float, str, list]]]]]) – number of outcomes to use
• normalized (bool) – if true, the resulting ufuns will be normalized between zero and one.

Return type  List[UtilityFunction]

classmethod generate_random_bilateral(outcomes)
Generates a couple of utility functions

Parameters

• n_outcomes (int) – number of outcomes to use
• conflict_level – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
• conflict_delta – How variable is the conflict at different outcomes.
• zero_summness – How zero-sum like are the two ufuns.

Return type  Tuple[UtilityFunction, UtilityFunction]

classmethod to_genius(u, issues, file_name, **kwargs)
Exports a utility function from a GENIUS XML file.

Parameters

• file_name (str) – File name to export to
• u (UtilityFunction) – utility function
• issues (List[Issue]) – The issues being considered as defined in the domain

Returns  None
Examples

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-domain.xml'), keep_issue_names=False)
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name=pkg_resources.resource_filename('negmas', 'data/LaptopConv/Laptop-C-prof1.xml'))
```

Remarks: See `to_xml_str` for all the parameters

classmethod `to_xml_str` (*u*, *issues*, *discount_factor=None*)
Exports a utility function to a well formatted string

Return type *str*

classmethod `winwin_level` (*u1*, *u2*, *outcomes*, *max_tests=10000*)
Finds the conflict level in these two ufun

Parameters

- `u1 (UtilityFunction)` –
- `u2 (UtilityFunction)` –

Examples

- A nonlinear same ufun case

  ```python
  >>> outcomes = [(_,) for _ in range(10)]
  >>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
  ```

- A linear strictly zero sum case

  ```python
  >>> outcomes = [(_,) for _ in range(10)]
  >>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
  >>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
  ```

Return type *float*

xml (*issues*)
Converts the function into a well formed XML string preferably in GENIUS format.

If the output has with `<objective>` then discount factor and reserved value should also be included If the output has `<utility_space>` it will not be appended in `to_xml_str`

Return type *str*
Class \texttt{ConstUFun}

```python
class\negmas\utilities\ConstUFun(value, name=None, reserved_value=None, ami=None):
    Bases: \negmas\utilities\UtilityFunction
```

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_call_ (offer)</td>
<td>Calculate the utility function value for a given outcome.</td>
</tr>
<tr>
<td>approximate(ufuns, issues, n_outcomes[, ...])</td>
<td>param cls</td>
</tr>
<tr>
<td>compare(o1, o2)</td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td>conflict_level(u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>eu(offer)</td>
<td>Calculate the expected utility function value.</td>
</tr>
<tr>
<td>from_genius(file_name, **kwargs)</td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>from_xml_str(xml_str[, domain_issues, ...])</td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td>generate_bilateral(outcomes[, ...])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random(n, outcomes[, normalized])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random_bilateral(outcomes)</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>to_genius(u, issues, file_name, **kwargs)</td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>to_xml_str(u, issues[, discount_factor])</td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td>winwin_level(u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>xml(issues)</td>
<td>Converts the function into a well formed XML string preferably in GENIUS format.</td>
</tr>
</tbody>
</table>

Attributes Documentation

\textbf{base_type}

Returns the utility function base type ignoring discounting and similar wrappings.

\textbf{Return type} str

\textbf{id}

The unique ID of this entity

\textbf{is_dynamic}

Whether the utility function can potentially depend on negotiation state (mechanism information).
NegMAS Documentation, Release 0.3.2

- If this property is `False`, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).

**type**
Returns the utility_function type.
Each class inheriting from this `UtilityFunction` class will have its own type. The default type is the empty string.

**Examples**

```python
global from negmas.utilities import *
global print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).
global type)
linear_aggregation
global print(MappingUtilityFunction(lambda x: x).type)
mapping
global print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x:
global --x).type)
non_linear_aggregation
```

**Returns** utility_function type

**Return type** str

**uuid**
The unique ID of this entity

**Methods Documentation**

**__call__(offer)**
Calculate the utility_function value for a given outcome.

**Parameters**

- `offer` (`Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]]]]`) – The offer to be evaluated.

**Remarks:**

- You cannot return None from overriden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

**Returns** The utility_function value which may be a distribution. If `None` it means the utility_function value cannot be calculated.

**Return type** UtilityValue

**classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5,**

```python
force_single_issue=False)
```

**Parameters**

- `cls` –
- `ufuns (List[UtilityFunction])` –
issues (Iterable[Issue]) –

n_outcomes (int) –

min_per_dim –

force_single_issue –

Returns:

Return type Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]]], List[Issue]]

compare (o1, o2)

Compares the two outcomes and returns a measure of the difference between their utilities

Return type Union[Distribution, float]

classmethod conflict_level (u1, u2, outcomes, max_tests=10000)

Finds the conflict level in these two ufuns

Parameters

• u1 (UtilityFunction) –

• u2 (UtilityFunction) –

Examples

• A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random._random(len(outcomes))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1._mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

• The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
0.0
```

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes)), endpoint=True)))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes)), endpoint=True)))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create (*args, **kwargs)

Creates an object and returns a proxy to it.

eu (offer)

Calculate the expected utility_function value.
Parameters **offer** (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

**Returns** The expected utility function for utility_priors and just utility_function for real-valued utilities.

**Return type** float

classmethod **from_genius**(file_name, **kwargs)**
Imports a utility function from a GENIUS XML file.

**Parameters**

- **file_name** (str) – File name to import from

**Returns** A utility function object (depending on the input file)

**Examples**

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

**Remarks:** See **from_xml_str** for all the parameters
classmethod **from_xml_str**(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
Imports a utility function from a GENIUS XML string.

**Parameters**

- **xml_str** (str) – The string containing GENIUS style XML utility function definition
- **domain_issues** (List[Issue]) – Optional issue space to confirm that the utility function is valid
- **force_single_issue** (bool) – Tries to generate a MappingUtility function with a single issue which is the
- **of all issues in the input** (product) –
- **keep_issue_names** (bool) – Keep names of issues
- **keep_value_names** (bool) – Keep names of values
- **safe_parsing** (bool) – Turn on extra checks
- **normalize_utility** (bool) – Normalize the output utilities to the range from 0 to 1
- **max_n_outcomes** (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

**Returns** A utility function object (depending on the input file)
Examples

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD'),) - 21.987727736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD'),) - 22.68559475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)  # (continues on next page)
... , keep_issue_names=False, keep_value_names=False, normalize_UTILITY=True)

>>> assert abs(u((0,)) - 0.599329436957658) < 0.1

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas'...

... , resource_name='tests/data/Laptop/Laptop-C-prof1.xml'))...

... , 'r').read(), force_single_issue=False, normalize_UTILITY=True)

>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 0.599329436957658) < 0.1

>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD"}) - 0.6342209804130308) < 0.01

>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 1.0) < 0.0001

classmethod generate_bilateral (outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

Parameters

• n_outcomes (int) – number of outcomes to use

• conflict_level (float) – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• conflict_delta – How variable is the conflict at different outcomes.

• zero_summness – How zero-sum like are the two ufuns.

Examples

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.0)

>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, conflict_delta=0.0, win_win=0.0)

>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, conflict_delta=0.0, win_win=1.0)

>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True

Return type Tuple[UtilityFunction, UtilityFunction]

classmethod generate_random (n, outcomes, normalized=True)

Generates a couple of utility functions

Parameters
• \( n (\text{int}) \) – number of utility functions to generate

• \( \text{outcomes} \ (\text{Union}[\text{int}, \ \text{List}[\text{Union[OutcomeType, Tuple[Union[\text{int, float, str, list]}], Dict[Union[\text{int, str}, \ \text{Union[\text{int, float, str, list}]]]]]]) \) – number of outcomes to use

• \( \text{normalized (bool)} \) – if true, the resulting ufuns will be normalized between zero and one.

**Return type** \( \text{List[UtilityFunction]} \)

**classmethod generate_random_bilateral(outcomes)**
Generates a couple of utility functions

**Parameters**

• \( n\_\text{outcomes (int)} \) – number of outcomes to use

• \( \text{conflict\_level} \) – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• \( \text{conflict\_delta} \) – How variable is the conflict at different outcomes.

• \( \text{zero\_summness} \) – How zero-sum like are the two ufuns.

**Return type** \( \text{Tuple[UtilityFunction, UtilityFunction]} \)

**classmethod to_genius(u, issues, file_name, **kwargs)**
Exports a utility function from a GENIUS XML file.

**Parameters**

• \( \text{file\_name (str)} \) – File name to export to

• \( u (\text{UtilityFunction}) \) – utility function

• \( \text{issues (List[Issue])} \) – The issues being considered as defined in the domain

**Returns** None

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', ... resource_filename='tests/data/Laptop/Laptop-C-domain.xml'))
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', ... resource_filename='tests/data/Laptop/Laptop-C-prof1.xml'))
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, ... file_name = pkg_resources.resource_filename('negmas', ... resource_filename='tests/data/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See to_xml_str for all the parameters

**classmethod to_xml_str(u, issues, discount_factor=None)**
Exports a utility function to a well formatted string

**Return type** \( \text{str} \)
classmethod `winwin_level`(u1, u2, outcomes, max_tests=10000)

Finds the conflict level in these two ufuns

**Parameters**

- `u1` (*UtilityFunction*)
- `u2` (*UtilityFunction*)

**Examples**

- A nonlinear same ufun case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

**Return type** `float`

`xml` (*issues*)

Converts the function into a well formed XML string preferably in GENIUS format. If the output has with </objective> then discount factor and reserved value should also be included. If the output has </utility_space> it will not be appended in `to_xml_str`.

**Return type** `str`

**LinDiscountedUFun**

```python
class negmas.utilities.LinDiscountedUFun(ufun, ami, cost=None, factor='step', power=1.0, name=None, reserved_value=None, dynamic_reservation=True)
```

**Bases:** `negmas.utilities.UtilityFunction`

A utility function with linear discounting based on some factor of the negotiation

**Parameters**

- `ufun` (*UtilityFunction*) – The utility function that is being discounted
- `cost` (*Optional*[float]) – discount factor
- `factor` (*Union*[str, Callable[[AgentMechanismInterface], float]]) – The name of the AgentMechanismInterface variable based on which discounting operate
- `power` (*float*) – A power to raise the total cost to before discounting it from the utility_function value
Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td>approximate</td>
<td>(\text{ufuns, issues, n_outcomes[...]})</td>
</tr>
<tr>
<td>param cls</td>
<td></td>
</tr>
<tr>
<td>compare</td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td>conflict_level</td>
<td>(\text{u1, u2, outcomes[, max_tests]})</td>
</tr>
<tr>
<td>create</td>
<td>(<em>\text{args, **kwargs}</em>)</td>
</tr>
<tr>
<td>eu</td>
<td>(\text{offer}))</td>
</tr>
<tr>
<td>from_genius</td>
<td>(\text{file_name, **kwargs}))</td>
</tr>
<tr>
<td>from_xml_str</td>
<td>(\text{xml_str[, domain_issues, ...]})</td>
</tr>
<tr>
<td>generate_bilateral</td>
<td>(\text{outcomes[...]})</td>
</tr>
<tr>
<td>generate_random</td>
<td>(n, outcomes[, normalized]))</td>
</tr>
<tr>
<td>generate_random_bilateral</td>
<td>(\text{outcomes}))</td>
</tr>
<tr>
<td>to_genius</td>
<td>(\text{u, issues, file_name, **kwargs}))</td>
</tr>
<tr>
<td>to_xml_str</td>
<td>(\text{u, issues[, discount_factor]})</td>
</tr>
<tr>
<td>xml</td>
<td>(\text{issues}))</td>
</tr>
</tbody>
</table>

Attributes Documentation

**base_type**
Returns the utility_function base type ignoring discounting and similar wrappings.

**id**
The unique ID of this entity

**is_dynamic**
Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is `False`, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).
type

Returns the utility_function type.

Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *

>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation

>>> print(MappingUtilityFunction(lambda x: x).type)
mapping

>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: x).type)
non_linear_aggregation
```

Returns utility_function type

Return type str

uuid

The unique ID of this entity

Methods Documentation

__call__(offer)

Calculate the utility_function value for a given outcome.

Parameters

```text
Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.
```

Remarks:

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

Return type UtilityValue

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters

```text
Parameters

• cls –
• ufuns (List[UtilityFunction]) –
• issues (Iterable[Issue]) –
• n_outcomes (int) –
• min_per_dim –
• force_single_issue –
```

Returns:
Return type Tuple[List[MappingUtilityFunction],
List[Union[OutcomeType, Tuple[Union[int, float, str, list],
Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

compare(o1, o2)
Compares the two outcomes and returns a measure of the difference between their utilities

Return type Union[Distribution, float]

classmethod conflict_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns

Parameters
• u1 (UtilityFunction) –
• u2 (UtilityFunction) –

Examples

• A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.
˓→random(len(outcomes))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.
˓→mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

• The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0,
˓→len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0,
˓→len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

eu(offer)
Calculate the expected utility_function value.

Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list],
Dict[Union[int, str], Union[int, float, str, list]]], Dict[Union[int, str],
Union[int, float, str, list]]]) – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float
classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See from_xml_str for all the parameters

classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
Imports a utility function from a GENIUS XML string.

Parameters

- xml_str (str) – The string containing GENIUS style XML utility function definition
- domain_issues (List[Issue]) – Optional issue space to confirm that the utility function is valid
- force_single_issue (bool) – Tries to generate a MappingUtility function with a single issue which is the
- of all issues in the input (product)
- keep_issue_names (bool) – Keep names of issues
- keep_value_names (bool) – Keep names of values
- safe_parsing (bool) – Turn on extra checks
- normalize_utility (bool) – Normalize the output utilities to the range from 0 to 1
- max_n_outcomes (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml')).read())
```
(continues on next page)
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(("Dell+60 Gb+19'' LCD",)) - 21.987727736172488) < 0.000001
>>> assert abs(u(("HP+80 Gb+20'' LCD",)) - 22.68559475583014) < 0.000001
```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True)
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 0.599329436957658) < 0.1
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': '20'' LCD'}) - 0.6342209804130308) < 0.01
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 1.0) < 0.0001
```

classmethod generate_bilateral

Generates a couple of utility functions

**Parameters**

- **n_outcomes** *(int)* – number of outcomes to use
- **conflict_level** *(float)* – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- **conflict_delta** – How variable is the conflict at different outcomes.
- **zero_summness** – How zero-sum like are the two ufuns.

**Examples**

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

**Return type** *Tuple[UtilityFunction, UtilityFunction]*

classmethod generate_random

Generates a couple of utility functions

**Parameters**

- **n** *(int)* – number of utility functions to generate
• **outcomes** *(Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str, Union[int, float, str, list]]]]]]]]* – number of outcomes to use

• **normalized** *(bool)* – if true, the resulting ufuns will be normalized between zero and one.

_**Return type** List[UtilityFunction]_

**classmethod generate_random_bilateral** *(outcomes)*
Generates a couple of utility functions

**Parameters**

• **n_outcomes** *(int)* – number of outcomes to use

• **conflict_level** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• **conflict_delta** – How variable is the conflict at different outcomes.

• **zero_summness** – How zero-sum like are the two ufuns.

_**Return type** Tuple[UtilityFunction, UtilityFunction]_

**classmethod to_genius** *(u, issues, file_name, **kwargs)*
Exports a utility function from a GENIUS XML file.

**Parameters**

• **file_name** *(str)* – File name to export to

• **u** *(UtilityFunction)* – utility function

• **issues** *(List[Issue])* – The issues being considered as defined in the domain

_**Returns** None_

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, __, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas...', resource_name='tests/Laptop/Laptop-C-domain.xml'))
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas...', resource_name='tests/Laptop/Laptop-C-prof1.xml'))
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name = pkg_resources.resource_filename('negmas...', resource_name='tests/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See **to_xml_str** for all the parameters

**classmethod to_xml_str** *(u, issues, discount_factor=None)*
Exports a utility function to a well formatted string

_**Return type** str_

**classmethod winwin_level** *(u1, u2, outcomes, max_tests=10000)*
Finds the conflict level in these two ufuns

5.2. negmas.utilities Module 81
Parameters

- \( u_1 (\text{UtilityFunction}) \)
- \( u_2 (\text{UtilityFunction}) \)

Examples

- A nonlinear same ufun case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

Return type float

xml(issues)

Converts the function into a well formed XML string preferably in GENIUS format.

If the output has with </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in to_xml_str

Return type str

ExpDiscountedUFun

class negmas.utilities.ExpDiscountedUFun(ufun, ami, beta=None, factor='step', name=None, reserved_value=None, dynamic_reservation=True)

Bases: negmas.utilities.UtilityFunction

A discounted utility function based on some factor of the negotiation

Parameters

- \( ufun (\text{UtilityFunction}) \) – The utility function that is being discounted
- \( \text{beta} (\text{Optional}[\text{float}]) \) – discount factor
- \( \text{factor} (\text{Union}[\text{str}, \text{Callable}[[\text{AgentMechanismInterface}, \text{float}]]]) \) – The name of the AgentMechanismInterface variable based on which discounting operate
- \* must receive a mechanism info object and returns a float representing the factor(\text{callable})–

Attributes Summary

<table>
<thead>
<tr>
<th>base_type</th>
<th>Returns the utility_function base type ignoring discounting and similar wrappings.</th>
</tr>
</thead>
</table>

Continued on next page
Table 17 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

**__call__**(offer) Calculate the utility_function value for a given outcome.

**approximate**(ufuns, issues, n_outcomes[, . . . ]) Returns a more efficient utility_function value.

**compare**(o1, o2) Compares the two outcomes and returns a measure of the difference between their utilities

**conflict_level**(u1, u2, outcomes[, max_tests]) Finds the conflict level in these two ufuns

**create**(arg1, **kw1) Creates an object and returns a proxy to it.

**eu**(offer) Calculate the expected utility_function value.

**from_genius**(file_name, **kw1) Imports a utility function from a GENIUS XML file.

**from_xml_str**(xml_str[, domain_issues, . . . ]) Imports a utility function from a GENIUS XML string.

**generate_bilateral**(outcomes[, . . . ]) Generates a couple of utility functions

**generate_random**(n, outcomes[, normalized]) Generates a couple of utility functions

**generate_random_bilateral**(outcomes) Generates a couple of utility functions

**to_genius**(u, issues, file_name, **kw1) Exports a utility function from a GENIUS XML file.

**to_xml_str**(u, issues[, discount_factor]) Exports a utility function to a well formatted string

**winwin_level**(u1, u2, outcomes[, max_tests]) Finds the conflict level in these two ufuns

**xml**(issues) Converts the function into a well formed XML string preferably in GENIUS format.

Attributes Documentation

**base_type**

Returns the utility_function base type ignoring discounting and similar wrappings.

**id**

The unique ID of this entity

**is_dynamic**

Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is **False**, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is **True**, the ufun may depend on negotiation state but it may also not depend on it.

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**type**

Returns the utility_function type.
Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *

>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation

>>> print(MappingUtilityFunction(lambda x: x).type)
mapping

>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: __).
˓→x).type)
non_linear_aggregation
```

Returns utility_function type

Return type str

uuid

The unique ID of this entity

Methods Documentation

__call__(offer)

Calculate the utility_function value for a given outcome.

Parameters

- `offer` (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Remarks:

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns

The utility_function value which may be a distribution. If `None` it means the utility_function value cannot be calculated.

Return type UtilityValue

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters

- `cls` –
- `ufuns` (List[UtilityFunction]) –
- `issues` (Iterable[Issue]) –
- `n_outcomes` (int) –
- `min_per_dim` –
- `force_single_issue` –

Returns:
**Return type** Tuple[List[MappingUtilityFunction],
List[Union[OutcomeType, Tuple[Union[int, float, str, list]],
Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

`compare(o1, o2)`
Compares the two outcomes and returns a measure of the difference between their utilities

**Return type** Union[Distribution, float]

**classmethod conflict_level(u1, u2, outcomes, max_tests=10000)**
Finds the conflict level in these two ufuns

**Parameters**
- `u1` (*UtilityFunction*) –
- `u2` (*UtilityFunction*) –

**Examples**

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.randint(10, size=len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

- The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

**Return type** float

**classmethod create(*args, **kwargs)**
Creates an object and returns a proxy to it.

**eu(offer)**
Calculate the expected utility_function value.

**Parameters**
- `offer` (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

**Returns** The expected utility_function for utility_priors and just utility_function for real-valued utilities.

**Return type** float
classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See from_xml_str for all the parameters

classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
Imports a utility function from a GENIUS XML string.

Parameters

- xml_str (str) – The string containing GENIUS style XML utility function definition
- domain_issues (List[Issue]) – Optional issue space to confirm that the utility function is valid
- force_single_issue (bool) – Tries to generate a MappingUtility function with a single issue which is the
- of all issues in the input (product) –
- keep_issue_names (bool) – Keep names of issues
- keep_value_names (bool) – Keep names of values
- safe_parsing (bool) – Turn on extra checks
- normalize_utility (bool) – Normalize the output utilities to the range from 0 to 1
- max_n_outcomes (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml')))
```
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True)

>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=False)

>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.987727736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.68559475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False, keep_issue_names=False, keep_value_names=False)

>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01

5.2. negmas.utilities Module
classmethod generate_bilateral(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

Parameters

- **n_outcomes** *(int)* – number of outcomes to use
- **conflict_level** *(float)* – How conflicting are the two ufuncs to generate. 1.0 means maximum conflict.
- **conflict_delta** – How variable is the conflict at different outcomes.
- **zero_summness** – How zero-sum like are the two ufuncs.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5
... , conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

Return type  Tuple*[UtilityFunction, UtilityFunction]*

classmethod generate_random(n, outcomes, normalized=True)

Generates a couple of utility functions

Parameters

- **n** *(int)* – number of utility functions to generate
• `outcomes` ([Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]]]) – number of outcomes to use

• `normalized` (bool) – if true, the resulting ufuns will be normalized between zero and one.

**Return type** List[UtilityFunction]

**classmethod generate_random_bilateral**(outcomes)
Generates a couple of utility functions

**Parameters**

• `n_outcomes` (int) – number of outcomes to use

• `conflict_level` – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• `conflict_delta` – How variable is the conflict at different outcomes.

• `zero_summness` – How zero-sum like are the two ufuns.

**Return type** Tuple[UtilityFunction, UtilityFunction]

**classmethod to_genius**(u, issues, file_name, **kwargs)
Exports a utility function from a GENIUS XML file.

**Parameters**

• `file_name` (str) – File name to export to

• `u` (UtilityFunction) – utility function

• `issues` (List[Issue]) – The issues being considered as defined in the domain

**Returns** None

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
... _, __, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-domain.xml'))
... u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
... UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name = pkg_resources.resource_filename('negmas', 'data/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See `to_xml_str` for all the parameters

**classmethod to_xml_str**(u, issues, discount_factor=None)
Exports a utility function to a well formatted string

**Return type** str

**classmethod winwin_level**(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns

**5.2. negmas.utilities Module**
Parameters

- $u_1 (UtilityFunction)$ –
- $u_2 (UtilityFunction)$ –

Examples

- A nonlinear same ufun case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, \len(outcomes), endpoint=True))))
``` 

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, \len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, \len(outcomes), endpoint=True))))
``` 

Return type float

`xml (issues)`

Converts the function into a well formed XML string preferably in GENIUS format.

If the output has with </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in `to_xml_str`

Return type str

MappingUtilityFunction

class negmas.utilities.MappingUtilityFunction (mapping, default=None, name=None, reserved_value=None, ami=None)

Bases: negmas.utilities.UtilityFunction

Outcome mapping utility function.

This is the simplest possible utility function and it just maps a set of “Outcome’s to a set of “Utility-Value”(s). It is only usable with single-issue negotiations. It can be constructed with wither a mapping (e.g. a direct) or a callable function.

Parameters

- `mapping` (Union[Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]], int, str, float]], Union[Distribution, float]], Mapping[Union[Sequence[+T_co], Mapping[-KT, +VT_co], int, float], Union[Distribution, float]]]]) – Either a callable or a mapping from Outcome (dict) to UtilityValue.
- `default` – value returned for outcomes causing exception (e.g. invalid outcomes).
- `name` (Optional[str]) – name of the utility function. If None a random name will be generated.

Examples:

Single issue outcome case:
```python
>>> issue = Issue(values=['to be', 'not to be'], name='THE problem')
>>> print(str(issue))
The problem: ['to be', 'not to be']

>>> f = MappingUtilityFunction({'to be':10.0, 'not to be':0.0})
>>> print(list(map(f, ['to be', 'not to be'])))
[10.0, 0.0]

>>> f = MappingUtilityFunction(mapping={'to be':-10.0, 'not to be':10.0})
>>> print(list(map(f, ['to be', 'not to be'])))
[-10.0, 10.0]

>>> f = MappingUtilityFunction(lambda x: float(len(x)))
>>> print(list(map(f, ['to be', 'not to be'])))
[5.0, 9.0]

Multi issue case:

```text
>>> issues = [Issue((10.0, 20.0), 'price'), Issue(['delivered', 'not delivered'], 'delivery'), Issue(5, 'quality')]
...

>>> print(list(map(str, issues)))
['price: (10.0, 20.0)', 'delivery: ['delivered', 'not delivered']', 'quality: 5']

>>> f = MappingUtilityFunction(lambda x: x['price'] if x['delivery'] == 'delivered' else -1.0)

>>> g = MappingUtilityFunction(lambda x: x['price'] if x['delivery'] == 'delivered' else -1.0, default=-1000)

>>> f({'price': 16.0}) is None
True

>>> g({'price': 16.0})
-1000

>>> f({'price': 16.0, 'delivery': 'delivered'})
16.0

>>> f({'price': 16.0, 'delivery': 'not delivered'})
-1.0
```

Remarks:

- If the mapping used failed on the outcome (for example because it is not a valid outcome), then the default value given to the constructor (which defaults to None) will be returned.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary
__call__(offer) Calculate the utility function value for a given outcome.

approximate(ufuns, issues, n_outcomes[...]) param cls

compare(o1, o2) Compares the two outcomes and returns a measure of the difference between their utilities

conflict_level(u1, u2, outcomes[, max_tests]) Finds the conflict level in these two ufuns

create(*args, **kwargs) Creates an object and returns a proxy to it.

eu(offer) Calculate the expected utility function value.

from_genius(file_name, **kwargs) Imports a utility function from a GENIUS XML file.

from_xml_str(xml_str[, domain_issues, ...]) Imports a utility function from a GENIUS XML string.

generate_bilateral(outcomes[, ...]) Generates a couple of utility functions

generate_random(n, outcomes[, normalized]) Generates a couple of utility functions

generate_random_bilateral(outcomes) Generates a couple of utility functions

to_genius(u, issues, file_name, **kwargs) Exports a utility function from a GENIUS XML file.

to_xml_str(u, issues[, file_name, **kwargs]) Exports a utility function to a well formatted string

winwin_level(u1, u2, outcomes[, max_tests]) Finds the conflict level in these two ufuns

Examples

Attributes Documentation

base_type
Returns the utility function base type ignoring discounting and similar wrappings.

Return type str

id
The unique ID of this entity

is_dynamic
Whether the utility function can potentially depend on negotiation state (mechanism information).

• If this property is False, the ufun can safely be assumed to be static (not dependent on negotiation state).
• If this property is True, the ufun may depend on negotiation state but it may also not depend on it.

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

type
Returns the utility function type.

Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

Examples
```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation
>>> print(MappingUtilityFunction(lambda x: x).type)
mapping
>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: -x).type)
non_linear_aggregation
```

**Returns** utility_function type

**Return type** str

**uuid**
The unique ID of this entity

**Methods Documentation**

**__call__(offer)**
Calculate the utility_function value for a given outcome.

**Parameters**
- **offer** (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be evaluated.

**Remarks:**
- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

**Returns** The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

**Return type** UtilityValue

**classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5,**
```
force_single_issue=False)```

**Parameters**
- **cls** –
- **ufuns** (List[UtilityFunction]) –
- **issues** (Iterable[Issue]) –
- **n_outcomes** (int) –
- **min_per_dim** –
- **force_single_issue** –

**Returns:**

**Return type** Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

**compare(o1, o2)**
Compares the two outcomes and returns a measure of the difference between their utilities

**Return type** Union[Distribution, float]
**classmethod conflict_level**(u1, u2, outcomes, max_tests=10000)

Finds the conflict level in these two ufuns

**Parameters**

- **u1** *(UtilityFunction)*  
- **u2** *(UtilityFunction)*

**Examples**

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.random(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))  
1.0
```

- The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))  
0.0
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))  
1.0
```

**Return type** float

**classmethod create**(\*args, **kwargs)

Creates an object and returns a proxy to it.

**eu**(offer)

Calculate the expected utility_function value.

**Parameters** offer *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])* – The offer to be evaluated.

**Returns** The expected utility_function for utility_priors and just utility_function for real-valued utilities.

**Return type** float

**classmethod from_genius**(file_name, **kwargs)

Imports a utility function from a GENIUS XML file.

**Parameters**

- **file_name** *(str)* – File name to import from

**Returns** A utility function object (depending on the input file)
Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', ...
... 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See `from_xml_str` for all the parameters

```python
classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False,
force_numeric=False, keep_issue_names=True, keep_value_names=True, normalize_utility=True,
max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
```

Imports a utility function from a GENIUS XML string.

Parameters

- `xml_str (str)` – The string containing GENIUS style XML utility function definition
- `domain_issues (List[Issue])` – Optional issue space to confirm that the utility function is valid
- `force_single_issue (bool)` – Tries to generate a MappingUtility function with a single issue which is the
- of all issues in the input (product) –
- `keep_issue_names (bool)` – Keep names of issues
- `keep_value_names (bool)` – Keep names of values
- `safe_parsing (bool)` – Turn on extra checks
- `normalize_utility (bool)` – Normalize the output utilities to the range from 0 to 1
- `max_n_outcomes (int)` – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', ...
... 'Laptop/Laptop-C-prof1.xml'))
... , 'r').read(), force_single_issue=False,
... , normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', '19'' LCD')) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', '20'' LCD')) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', '19'' LCD')) - 1.0) < 0.0001
```
```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+60 Gb+20'' LCD',)) - 22.6859475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.98772736172488) < 0.000001
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 21.98772736172488) < 0.000001
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD"}) - 22.6859475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+60 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01
```

(continues on next page)
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 1.0) < 0.0001

**classmethod generate_bilateral**(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

**Parameters**

- **n_outcomes** (*int*) – number of outcomes to use
- **conflict_level** (*float*) – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- **conflict_delta** – How variable is the conflict at different outcomes.
- **zero_summness** – How zero-sum like are the two ufuns.

**Examples**

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0,... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0,... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5,... , conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

**Return type** `Tuple[UtilityFunction, UtilityFunction]`

**classmethod generate_random**(n, outcomes, normalized=True)

Generates a couple of utility functions

**Parameters**

- **n** (*int*) – number of utility functions to generate
- **outcomes** (*Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str, Union[int, float, str, list]]]]]]]*) – number of outcomes to use
- **normalized** (*bool*) – if true, the resulting ufuns will be normalized between zero and one.

**Return type** `List[UtilityFunction]`

**classmethod generate_random_bilateral**(outcomes)

Generates a couple of utility functions

**Parameters**
• **n_outcomes** (int) – number of outcomes to use
• **conflict_level** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
• **conflict_delta** – How variable is the conflict at different outcomes.
• **zero_summness** – How zero-sum like are the two ufuns.

Return type **Tuple**[

`UtilityFunction`,
`UtilityFunction`]

**classmethod to_genius**(*u, issues, file_name, **kwargs*)
Exports a utility function from a GENIUS XML file.

Parameters

• **file_name** (str) – File name to export to
• **u** (*UtilityFunction*) – utility function
• **issues** (*List[Issue]*) – The issues being considered as defined in the domain

Returns None

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', ... → resource_name='tests/data/Laptop/Laptop-C-domain.xml'))
... ...
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', ... → resource_name='tests/data/Laptop/Laptop-C-prof1.xml'))
... ...
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, ... → file_name=pkg_resources.resource_filename('negmas', ... → resource_name='tests/data/LaptopConv/Laptop-C-prof1.xml'))
```

Remarks: See **to_xml_str** for all the parameters

**classmethod to_xml_str**(*u, issues, discount_factor=None*)
Exports a utility function to a well formatted string

Return type **str**

**classmethod winwin_level**(*u1, u2, outcomes, max_tests=10000*)
Finds the conflict level in these two ufuns

Parameters

• **u1** (*UtilityFunction*) –
• **u2** (*UtilityFunction*) –

**Examples**

• A nonlinear same ufun case
```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

**Return type** float

**xml** *(issues)*

**Examples**

```python
>>> issue = Issue(values=['to be', 'not to be'], name='THE problem')
>>> print(str(issue))
THE problem: ['to be', 'not to be']
```

```python
>>> f = MappingUtilityFunction({'to be':10.0, 'not to be':0.0})
>>> print(list(map(f, ['to be', 'not to be'])))
[10.0, 0.0]
```

```python
>>> print(f.xml([issue]))
<issue index="1" etype="discrete" type="discrete" vtype="discrete" name="THE problem">
  <item index="1" value="to be" cost="0" evaluation="10.0" description="to be">
  </item>
  <item index="2" value="not to be" cost="0" evaluation="0.0" description="not to be">
  </item>
</issue>
<weight index="1" value="1.0">
</weight>
<BLANKLINE>
```

**Return type** str

**LinearUtilityAggregationFunction**

```python
class negmas.utilities.LinearUtilityAggregationFunction(issue_utilities, weights=None, name=None, reserved_value=None, ami=None)
Bases: negmas.utilities.UtilityFunction
```

A linear utility function for multi-issue negotiations.

Models a linear utility function using predefined weights:

**Parameters**

- **issue_utilities** *(Union[Mutablemapping[any], Union[Callable[[any], any], Mapping[~KT, +VT_co], Sequence[~T_co]], Sequence[Union[Callable[[any], any], Mapping[~KT, +VT_co], Sequence[~T_co]]]])* – utility functions for individual issues
• **weights** *(Union[Mapping[Any, float], Sequence[float], None]) – weights for combining issue_utilities

• **name** *(Optional[str]) – name of the utility function. If None a random name will be generated.*

**Notes**

The utility value is calculated as:

\[ u = \sum_{i=0}^{n_{\text{outcomes}}-1} w_i * u_i \]

**Examples**

```python
g = LinearUtilityAggregationFunction({"price": lambda x: 2.0*x, ... }, weights={"price": 1.0, "delivery": 2.0, "quality": 4.0})

float(g({'quality': 2, 'price': 14.0, 'delivery': 'delivered'})) - (1.0*(2.0*14)+2.0*10+4.0*(2.0-3.0))
```

You can use lists instead of dictionaries for defining outcomes, weights and nonlinearity but that is less readable

```python
g = LinearUtilityAggregationFunction([lambda x: 2.0*x, ... ], weights=[1.0, 2.0, 4.0])

float(g((14.0, 'delivered', 2))) - (1.0*(2.0*14)+2.0*10+4.0*(2.0-3.0))
```

**Remarks:** The mapping need not use all the issues in the output as the last example show.

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 21 – continued from previous page

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(offer)</td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td>approximate(ufuns, issues, n_outcomes[, ...])</td>
<td>Calculates the approximate utility_function value.</td>
</tr>
<tr>
<td>compare(o1, o2)</td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities.</td>
</tr>
<tr>
<td>conflict_level(u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>eu(offer)</td>
<td>Calculates the expected utility_function value.</td>
</tr>
<tr>
<td>from_genius(file_name, **kwargs)</td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>from_xml_str(xml_str[, domain_issues, ...])</td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td>generate_bilateral(outcomes[, ...])</td>
<td>Generates a couple of utility functions.</td>
</tr>
<tr>
<td>generate_random(n, outcomes[, normalized])</td>
<td>Generates a couple of utility functions.</td>
</tr>
<tr>
<td>generate_random_bilateral(outcomes)</td>
<td>Generates a couple of utility functions.</td>
</tr>
<tr>
<td>to_genius(u, issues, file_name, **kwargs)</td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>to_xml_str(u, issues[, discount_factor])</td>
<td>Exports a utility function to a well formatted string.</td>
</tr>
<tr>
<td>winwin_level(u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns.</td>
</tr>
<tr>
<td>xml(issues)</td>
<td>Generates an XML string representing the utility function.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td></td>
<td>Return type: str</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td></td>
<td>• If this property is False, the ufun can safely be assumed to be static (not dependent on negotiation state).</td>
</tr>
<tr>
<td></td>
<td>• If this property is True, the ufun may depend on negotiation state but it may also not depend on it.</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td></td>
<td>Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.</td>
</tr>
</tbody>
</table>
Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation
>>> print(MappingUtilityFunction(lambda x: x).type)
mapping
>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: x).type)
non_linear_aggregation
```

Returns  utility_function type

Return type  str

**uuid**

The unique ID of this entity

Methods Documentation

__call__(offer)

Calculate the utility_function value for a given outcome.

Parameters

- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None])* – The offer to be evaluated.

Remarks:

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns  The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

Return type  UtilityValue

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters

- **cls** –
- **ufuns** *(List[UtilityFunction])* –
- **issues** *(Iterable[Issue])* –
- **n_outcomes** *(int)* –
- **min_per_dim** –
- **force_single_issue** –

Returns:

Return type  Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

compare(o1, o2)

Compares the two outcomes and returns a measure of the difference between their utilities
Return type Union[Distribution, float]

classmethod conflict_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns

Parameters

- u1 (UtilityFunction)
- u2 (UtilityFunction)

Examples

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.randint(0, 1, len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

- The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

eu(offer)
Calculate the expected utility_function value.

Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float

classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)
Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas'...Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See `from_xml_str` for all the parameters

```python
classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
```
Imports a utility function from a GENIUS XML string.

**Parameters**

- `xml_str (str)` – The string containing GENIUS style XML utility function definition
- `domain_issues (List[Issue])` – Optional issue space to confirm that the utility function is valid
- `force_single_issue (bool)` – Tries to generate a MappingUtility function with a single issue which is the
- of all issues in the input (product)
- `keep_issue_names (bool)` – Keep names of issues
- `keep_value_names (bool)` – Keep names of values
- `safe_parsing (bool)` – Turn on extra checks
- `normalize_utility (bool)` – Normalize the output utilities to the range from 0 to 1
- `max_n_outcomes (int)` – Maximum number of outcomes allowed (effective only if force_single_issue is True)

**Returns** A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas'...Laptop/Laptop-C-prof1.xml')...issue=False,..., normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001
```
```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True,
normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.6859475583014) < 0.000001
```

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True,
keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.98772736172488) < 0.000001
```

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False,
normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01
```

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False,
normalize_utility=True)
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 0.599329436957658) < 0.1
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': '20'' LCD'}) - 0.6342209804130308) < 0.01
```

(continues on next page)
```python
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 1.0) < 0.0001
```

classmethod `generate_bilateral` *(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)*

Generates a couple of utility functions

**Parameters**

- `n_outcomes` *(int)* – number of outcomes to use
- `conflict_level` *(float)* – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- `conflict_delta` – How variable is the conflict at different outcomes.
- `zero_summness` – How zero-sum like are the two ufuns.

**Examples**

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, ... , conflict_delta=0.0, win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, ... , conflict_delta=0.0, win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, ... , conflict_delta=0.0, win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

**Return type** `Tuple[UtilityFunction, UtilityFunction]`

classmethod `generate_random` *(n, outcomes, normalized=True)*

Generates a couple of utility functions

**Parameters**

- `n` *(int)* – number of utility functions to generate
- `outcomes` *(Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str, Union[int, float, str, list]]]]]]]*) – number of outcomes to use
- `normalized` *(bool)* – if true, the resulting ufuns will be normalized between zero and one.

**Return type** `List[UtilityFunction]`

classmethod `generate_random_bilateral` *(outcomes)*

Generates a couple of utility functions

**Parameters**
• `n_outcomes` *(int)* – number of outcomes to use

• `conflict_level` – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• `conflict_delta` – How variable is the conflict at different outcomes.

• `zero_summness` – How zero-sum like are the two ufuns.

**Return type** `Tuple[UtilityFunction, UtilityFunction]`

### `classmethod to_genius(u, issues, file_name, **kwargs)`

Exports a utility function from a GENIUS XML file.

**Parameters**

• `file_name` *(str)* – File name to export to

• `u` *(UtilityFunction)* – utility function

• `issues` *(List[Issue]*) – The issues being considered as defined in the domain

**Returns** None

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain

>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', ...
resource_filename='tests/data/Laptop/Laptop-C-domain.xml'))

>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', ...
resource_filename='tests/data/Laptop/Laptop-C-prof1.xml'))

>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, ...
file_name = pkg_resources.resource_filename('negmas', ...
resource_name='tests/data/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See `to_xml_str` for all the parameters

### `classmethod to_xml_str(u, issues, discount_factor=None)`

Exports a utility function to a well formatted string

**Return type** `str`

### `classmethod winwin_level(u1, u2, outcomes, max_tests=10000)`

Finds the conflict level in these two ufuns

**Parameters**

• `u1` *(UtilityFunction)* –

• `u2` *(UtilityFunction)* –

**Examples**

• A nonlinear same ufun case

5.2. negmas.utilities Module 107
• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
```

**Return type** `float`

**xml** *(issues)*

Generates an XML string representing the utility function

**Parameters** `issues` *(List[Issue])* –

**Examples**

```python
>>> issues = [Issue(values=10, name='i1'), Issue(values=['delivered', 'not delivered'], name='i2')
... , Issue(values=4, name='i3')]
>>> f = LinearUtilityAggregationFunction([
... lambda x: 2.0*x,
... {'delivered': 10, 'not delivered': -10},
... MappingUtilityFunction(lambda x: x-3),
... ], weights=[1.0, 2.0, 4.0])
>>> print(f.xml(issues))
<issue index="1" etype="discrete" type="discrete" vtype="discrete" name="i1">
    <item index="1" value="0" evaluation="0.0" />
    <item index="2" value="1" evaluation="2.0" />
    <item index="3" value="2" evaluation="4.0" />
    <item index="4" value="3" evaluation="6.0" />
    <item index="5" value="4" evaluation="8.0" />
    <item index="6" value="5" evaluation="10.0" />
    <item index="7" value="6" evaluation="12.0" />
    <item index="8" value="7" evaluation="14.0" />
    <item index="9" value="8" evaluation="16.0" />
    <item index="10" value="9" evaluation="18.0" />
</issue>
<issue index="2" etype="discrete" type="discrete" vtype="discrete" name="i2">
    <item index="1" value="delivered" evaluation="10" />
    <item index="2" value="not delivered" evaluation="-10" />
</issue>
<issue index="3" etype="discrete" type="discrete" vtype="discrete" name="i3">
    <item index="1" value="0" evaluation="-3" />
    <item index="2" value="1" evaluation="-2" />
    <item index="3" value="2" evaluation="-1" />
    <item index="4" value="3" evaluation="0" />
</issue>
<weight index="1" value="1.0" />
</weight>
<weight index="2" value="2.0" />
</weight>
<weight index="3" value="4.0" />
```
Return type  &str

NonLinearUtilityAggregationFunction

class negmas.utilities.NonLinearUtilityAggregationFunction(issue_utilities, f, name=None, reserved_value=None, ami=None)

Bases: negmas.utilities.UtilityFunction

A nonlinear utility function.

Allows for the modeling of a single nonlinear utility function that combines the utilities of different issues.

Parameters

- **issue_utilities** (MutableMapping[Any, Union[Callable[[Any], Any], Mapping[-KT, +VT_co], Sequence[+T_co]]) – A set of mappings from issue values to utility functions. These are generic mappings so Callable(s) and Mapping(s) are both accepted

- **f** (Callable[[Dict[Any, Union[Distribution, float]]], Union[Distribution, float]]) – A nonlinear function mapping from a dict of utility_function-per-issue to a float
• **name** *(Optional[str])* – name of the utility function. If None a random name will be generated.

**Notes**

The utility is calculated as:

\[
u = f\\left(u_0\\right)\\left(i_0\\right)\\left(right)\\left(\\right)\\left(u_1\\right)\\left(i_1\\right)\\left(right)\\left(\\right)\\ldots\\left(u_n\\right)\\left(i_n\\right)\\left(right)\\right)
\]

where \(u_j()\) is the utility function for issue \(j\) and \(i_j\) is value of issue \(j\) in the evaluated outcome.

**Examples**

```python
>>> issues = [Issue((10.0, 20.0), 'price'), Issue(['delivered', 'not delivered'], 'delivery')]
... print(list(map(str, issues))
['price: (10.0, 20.0)', 'delivery: ['delivered', 'not delivered']]

>>> g = NonLinearUtilityAggregationFunction({'price': lambda x: 2.0*x, 'delivery': {'delivered': 10, 'not delivered': -10}})
... float(g({'quality': 2, 'price': 14.0, 'delivery': 'delivered'})) - ((2.0*14)+2.0*(2.0-3.0))
0.0

>>> g = NonLinearUtilityAggregationFunction({'price': lambda x: 2.0*x, 'delivery': {'delivered': 10, 'not delivered': -10}})
... float(g({'price': 14.0, 'delivery': 'delivered'})) - (2.0*(2.0*14))
0.0
```

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base_type</strong></td>
<td>Returns the utility function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td><strong>id</strong></td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td><strong>is_dynamic</strong></td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Returns the utility function type.</td>
</tr>
<tr>
<td><strong>uuid</strong></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>
Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong> (offer)</td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td>approximate (ufuns, issues, n_outcomes[, ...])</td>
<td>param cls</td>
</tr>
<tr>
<td>compare (o1, o2)</td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td>conflict_level (u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>create (*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>euf (offer)</td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td>from_genius (file_name, **kwargs)</td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>from_xml_str (xml_str[, domain_issues, ...])</td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td>generate_bilateral (outcomes[, ...])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random (n, outcomes[, normalized])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random_bilateral (outcomes)</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>to_genius (u, issues, file_name, **kwargs)</td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>to_xml_str (u, issues[, discount_factor])</td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td>winwin_level (u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>xml (issues)</td>
<td>Converts the function into a well formed XML string preferably in GENIUS format.</td>
</tr>
</tbody>
</table>

Attributes Documentation

base_type
Returns the utility_function base type ignoring discounting and similar wrappings.

Return type | str

id
The unique ID of this entity

is_dynamic
Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is False, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is True, the ufun may depend on negotiation state but it may also not depend on it.

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

type
Returns the utility_function type.

Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).
      type)
```
(continues on next page)
linear_aggregation

```python
>>> print(MappingUtilityFunction(lambda x: x).type)
mapping

>>> print(NonLinearUtilityAggregationFunction({1: lambda x:x}, f=lambda x:_→x).type)
non_linear_aggregation
```

Returns utility_function type

Return type str

uuid

The unique ID of this entity

Methods Documentation

__call__(offer)

Calculate the utility_function value for a given outcome.

Parameters

- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int,str], Union[int, float, str, list]], None])* – The offer to be evaluated.

Remarks:

- You cannot return `None` from overridden `apply()` functions but raise an exception (`ValueError`) if it was not possible to calculate the `UtilityValue`.
- Return A `UtilityValue` not a float for real-valued utilities for the benefit of inspection code.

Returns: The utility_function value which may be a distribution. If `None` it means the utility_function value cannot be calculated.

Return type `UtilityValue`

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters

- **cls** –
- **ufuns** *(List[UtilityFunction])* –
- **issues** *(Iterable[Issue])* –
- **n_outcomes** *(int)* –
- **min_per_dim** –
- **force_single_issue** –

Returns:

Return type `Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int,str], Union[int, float, str, list]]]], List[Issue]]`

compare(o1, o2)

Compares the two outcomes and returns a measure of the difference between their utilities

Return type `Union[Distribution, float]`

classmethod conflict_level(u1, u2, outcomes, max_tests=10000)

Finds the conflict level in these two ufuns
Parameters

- \( u_1 (UtilityFunction) \)
- \( u_2 (UtilityFunction) \)

Examples

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.rand(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

- The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type `float`

**classmethod create(**args, **kwargs)**

Creates an object and returns a proxy to it.

**eu***(offer)*

Calculate the expected utility_function value.

Parameters **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.*

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type `float`

**classmethod from_genius***(file_name, **kwargs)**

Imports a utility function from a GENIUS XML file.

Parameters **file_name** *(str) – File name to import from.*

Returns A utility function object (depending on the input file)
Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See `from_xml_str` for all the parameters

```python
classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
```

Imports a utility function from a GENIUS XML string.

Parameters

- `xml_str (str)` – The string containing GENIUS style XML utility function definition
- `domain_issues (List[Issue])` – Optional issue space to confirm that the utility function is valid
- `force_single_issue (bool)` – Tries to generate a MappingUtility function with a single issue which is the
- `of all issues in the input (product)` –
- `keep_issue_names (bool)` – Keep names of issues
- `keep_value_names (bool)` – Keep names of values
- `safe_parsing (bool)` – Turn on extra checks
- `normalize_utility (bool)` – Normalize the output utilities to the range from 0 to 1
- `max_n_outcomes (int)` – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
```

114 Chapter 5. Base Modules
```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.6859475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.98772736172488) < 0.000001
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 21.98772736172488) < 0.000001
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD"}) - 22.6859475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True)
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 0.599329436957658) < 0.1
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD"}) - 0.6342209804130308) < 0.01
(continues on next page)```
class method `generate_bilateral(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)`

Generates a couple of utility functions

**Parameters**

- `n_outcomes (int)` – number of outcomes to use
- `conflict_level (float)` – How conflicting are the two ufun to generate. 1.0 means maximum conflict.
- `conflict_delta` – How variable is the conflict at different outcomes.
- `zero_summness` – How zero-sum like are the two ufun.

**Examples**

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

**Return type** `Tuple[UtilityFunction, UtilityFunction]`

class method `generate_random(n, outcomes, normalized=True)`

Generates a couple of utility functions

**Parameters**

- `n (int)` – number of utility functions to generate
- `outcomes` (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str, Union[int, float, str, list]]]]]]]) – number of outcomes to use
- `normalized (bool)` – if true, the resulting ufun will be normalized between zero and one.

**Return type** `List[UtilityFunction]`

class method `generate_random_bilateral(outcomes)`

Generates a couple of utility functions

**Parameters**

```python
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 1.0) < 0.0001
```
• **n_outcomes** (*int*) – number of outcomes to use

• **conflict_level** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• **conflict_delta** – How variable is the conflict at different outcomes.

• **zero_summness** – How zero-sum like are the two ufuns.

Return type: Tuple[UtilityFunction, UtilityFunction]

### classmethod to_genius(*u, issues, file_name, **kwargs*)

Exports a utility function from a GENIUS XML file.

**Parameters**

• **file_name** (*str*) – File name to export to

• **u** (*UtilityFunction*) – utility function

• **issues** (*List[Issue]*) – The issues being considered as defined in the domain

**Returns** None

### Examples

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-domain.xml'), keep_issue_names=False)
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name = pkg_resources.resource_filename('negmas', 'data/LaptopConv/Laptop-C-prof1.xml'))

Remarks: See to_xml_str for all the parameters

### classmethod to_xml_str(*u, issues, discount_factor=None*)

Exports a utility function to a well formatted string

**Return type** *str*

### classmethod winwin_level(*u1, u2, outcomes, max_tests=10000*)

Finds the conflict level in these two ufuns

**Parameters**

• **u1** (*UtilityFunction*) –

• **u2** (*UtilityFunction*) –

### Examples

• A nonlinear same ufun case
A linear strictly zero sum case

```
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, _len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, _len(outcomes), endpoint=True))))
```

Return type float

xml((issues))
Converts the function into a well formed XML string preferably in GENIUS format.
If the output has with </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in to_xml_str

Return type str

HyperRectangleUtilityFunction

class negmas.utilities.HyperRectangleUtilityFunction (outcome_ranges, utilities, weights=None, *, ignore_issues_not_in_input=False, ignore_failing_range_utilities=False, name=None, reserved_value=None, ami=None)

Bases: negmas.utilities.UtilityFunction

A utility function defined as a set of hyper-volumes.

The utility function that is calculated by combining linearly a set of probably nonlinear functions applied in predefined hyper-volumes of the outcome space.

Args:
outcome_ranges: The outcome_ranges for which the mappings are defined mappings: The possibly nonlinear mappings corresponding to the outcome_ranges weights: The optional weights to use for combining the outputs of the mappings ignore_issues_not_in_input: If a hyper-volume local function is defined for some issue that is not in the outcome being evaluated ignore it. ignore_failing_range_utilities: If a hyper-volume local function fails, just assume it did not exist for this outcome. name: name of the utility function. If None a random name will be generated.

Examples: We will use the following issue space of cardinality 10\times5\times4:

```
>>> issues = [Issue(10), Issue(5), Issue(4)]
```

Now create the utility function with

```
>>> f = HyperRectangleUtilityFunction(outcome_ranges=[
...   {0: (1.0, 2.0), 1: (1.0, -> 2.0)},
...   {0: (1.4, 2.0), 2: (2.0, -> 3.0)}]
...   , utilities=[2.0, lambda x: 2*x[2] + x[0]])
```

(continues on next page)
We can now calculate the utility_function of some outcomes:

• An outcome that belongs to the both outcome_ranges:

```python
>>> [f({0: 1.5, 1: 1.5, 2: 2.5}), g({0: 1.5, 1: 1.5, 2: 2.5}), h({0: 1.5, 1: 1.5, 2: 2.5})]
[8.5, 8.5, 8.5]
```

• An outcome that belongs to the first hypervolume only:

```python
>>> [f({0: 1.5, 1: 1.5, 2: 1.0}), g({0: 1.5, 1: 1.5, 2: 1.0}), h({0: 1.5, 1: 1.5, 2: 1.0})]
[2.0, 2.0, 2.0]
```

• An outcome that belongs to and has the first hypervolume only:

```python
>>> [f({0: 1.5}), g({0: 1.5}), h({0: 1.5})]
[None, 0.0, None]
```

• An outcome that belongs to the second hypervolume only:

```python
>>> [f({0: 1.5, 2: 2.5}), g({0: 1.5, 2: 2.5}), h({0: 1.5, 2: 2.5})]
[None, 6.5, None]
```

• An outcome that has and belongs to the second hypervolume only:

```python
>>> [f({2: 2.5}), g({2: 2.5}), h({2: 2.5})]
[None, 0.0, None]
```

• An outcome that belongs to no outcome_ranges:

```python
>>> [f({0: 11.5, 1: 11.5, 2: 12.5}), g({0: 11.5, 1: 11.5, 2: 12.5}), h({0: 11.5, 1: 11.5, 2: 12.5})]
[0.0, 0.0, 0.0]
```

Remarks:

• The number of outcome_ranges, mappings, and weights must be the same
• if no weights are given they are all assumed to equal unity
• mappings can either by an OutcomeUtilityMapping or a constant.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td>adjust_params</td>
<td></td>
</tr>
<tr>
<td>approximate</td>
<td>param cls</td>
</tr>
<tr>
<td>compare</td>
<td></td>
</tr>
<tr>
<td>conflict_level</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>create</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>eu</td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td>from_genius</td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>from_xml_str</td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td>generate_bilateral</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random_bilateral</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>to_genius</td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>to_xml_str</td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td>winwin_level</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>xml</td>
<td>Represents the function as XML</td>
</tr>
</tbody>
</table>

Attributes Documentation

base_type

Returns the utility_function base type ignoring discounting and similar wrappings.

Return type str

id

The unique ID of this entity

is_dynamic

Whether the utility function can potentially depend on negotiation state (mechanism information).

• If this property is False, the ufun can safely be assumed to be static (not dependent on negotia-
• If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**type**

Returns the utility_function type.

Each class inheriting from this `UtilityFunction` class will have its own type. The default type is the empty string.

**Examples**

```python
>>> from negmas.utilities import *

>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).__type__)  # doctest: +NORMALIZE_WHITESPACE
linear_aggregation

>>> print(MappingUtilityFunction(lambda x: x).__type__)  # doctest: +NORMALIZE_WHITESPACE
mapping

>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: x).__type__)  # doctest: +NORMALIZE_WHITESPACE
non_linear_aggregation
```

**Returns** utility_function type

**Return type** str

**uuid**

The unique ID of this entity

### Methods Documentation

**`__call__` (offer)**

Calculate the utility_function value for a given outcome.

**Parameters**

- **offer** (`Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]`) – The offer to be evaluated.

**Remarks:**

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

**Returns** The utility_function value which may be a distribution. If `None` it means the utility_function value cannot be calculated.

**Return type** UtilityValue

**adjust_params ()**

**classmethod approximate (ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)**

**Parameters**

- **cls** –
• ufuncs (List[UtilityFunction]) –
• issues (Iterable[Issue]) –
• n_outcomes (int) –
• min_per_dim –
• force_single_issue –

Returns:

Return type Tuple[List[MappingUtilityFunction],
List[Union[OutcomeType, Tuple[Union[int, float, str, list]],
Dict[Union[int,str], Union[int,float,str,list]]]], List[Issue]]

compare (o1, o2)
Compares the two outcomes and returns a measure of the difference between their utilities

Return type Union[Distribution, float]

classmethod conflict_level (u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns

Parameters
• u1 (UtilityFunction) –
• u2 (UtilityFunction) –

Examples

• A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.rand(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

• The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create (*args, **kwargs)
Creates an object and returns a proxy to it.
eu (offer)
Calculate the expected utility_function value.

Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float
classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserve_value
0.0
>>> d
1.0
```

Remarks: See from_xml_str for all the parameters
classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
Imports a utility function from a GENIUS XML string.

Parameters

- xml_str (str) – The string containing GENIUS style XML utility function definition
- domain_issues (List[Issue]) – Optional issue space to confirm that the utility function is valid
- force_single_issue (bool) – Tries to generate a MappingUtility function with a single issue which is the
- of all issues in the input (product) –
- keep_issue_names (bool) – Keep names of issues
- keep_value_names (bool) – Keep names of values
- safe_parsing (bool) – Turn on extra checks
- normalize_utility (bool) – Normalize the output utilities to the range from 0 to 1
- max_n_outcomes (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)
Returns  A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=True, keep_issue_names=False, keep_value_names=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD')),) - 21.987727736172488 < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD')),) - 22.68559475583014 < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.987727736172488) < 0.000001
```

(continues on next page)
class method `generate_bilateral` *(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)*

Generates a couple of utility functions

**Parameters**

- `n_outcomes (int)` – number of outcomes to use
- `conflict_level (float)` – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- `conflict_delta` – How variable is the conflict at different outcomes.
- `zero_summness` – How zero-sum like are the two ufuns.

**Examples**

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0,
... win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0,
... win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5,
... win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

**Return type** `Tuple[UtilityFunction, UtilityFunction]`
classmethod `generate_random` \(n, \text{outcomes}, \text{normalized}=\text{True}\)
Generates a couple of utility functions

Parameters

- \(n\) (int) – number of utility functions to generate
- \text{outcomes} (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]) – number of outcomes to use
- \text{normalized} (bool) – if true, the resulting ufuns will be normalized between zero and one.

Return type List[UtilityFunction]

classmethod `generate_random_bilateral` \(\text{outcomes}\)
Generates a couple of utility functions

Parameters

- \text{n_outcomes} (int) – number of outcomes to use
- \text{conflict_level} – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- \text{conflict_delta} – How variable is the conflict at different outcomes.
- \text{zero_summness} – How zero-sum like are the two ufuns.

Return type Tuple[UtilityFunction, UtilityFunction]

classmethod `to_genius` \(u, \text{issues}, \text{file_name}, **\text{kwargs}\)
Exports a utility function from a GENIUS XML file.

Parameters

- \text{file_name} (str) – File name to export to
- \(u\) (UtilityFunction) – utility function
- \text{issues} (List[Issue]) – The issues being considered as defined in the domain

Returns None

Examples

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-domain.xml'), keep_issue_names=False)
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name = pkg_resources.resource_filename('negmas', resource_name='tests/data/LaptopConv/Laptop-C-prof1.xml'))
```

Remarks: See to_xml_str for all the parameters
classmethod to_xml_str(u, issues, discount_factor=None)
Exports a utility function to a well formatted string

Return type str

classmethod winwin_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufun

Parameters
• u1 (UtilityFunction) –
• u2 (UtilityFunction) –

Examples
• A nonlinear same ufun case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True)))
```

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

Return type float

xml(issues)
Represents the function as XML

Parameters issues (List[Issue]) –

Examples
```python
>>> f = HyperRectangleUtilityFunction(outcome_ranges=[
... {0: (1.0, 2.0), 1: (1.0, 2.0)},
... {0: (1.4, 2.0), 2: (2.0, 3.0)}
... , utilities=[2.0, 9.0 + 4.0])
>>> print(f.xml([Issue((0.0, 4.0), name='0'), Issue((0.0, 9.0), name='1')
... , Issue((0.0, 9.0), name='2')]).strip())
```

(continues on next page)
Return type  str

NonlinearHyperRectangleUtilityFunction

class  negmas.utilities.NonlinearHyperRectangleUtilityFunction(hypervolumes, mappings, f, name=None, reserved_value=None, ami=None)

Bases: negmas.utilities.UtilityFunction

A utility function defined as a set of outcome_ranges.

Parameters

- **hypervolumes** *(Iterable[Mapping[Union[int, str], Union[int, float, str, List[int], List[float], Tuple[int, int], Tuple[float, float], List[Tuple[Union[int, float], Union[int, float]]]]]]) – see HyperRectangleUtilityFunction

- **mappings** *(List[Union[Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], int, str, float]], Union[Distribution, float]], Mapping[Sequence[+T_co], Mapping[~KT, +VT_co], int, float], Union[Distribution, float]]]]) – see HyperRectangleUtilityFunction

- **f** *(Callable[[List[Union[Distribution, float]]], Union[Distribution, float]]) – A nonlinear function to combine the results of mappings

- **name** *(Optional[str]) – name of the utility function. If None a random name will be generated

Attributes Summary

| base_type | Returns the utility_function base type ignoring discounting and similar wrappings. |
| id | The unique ID of this entity |
| is_dynamic | Whether the utility function can potentially depend on negotiation state (mechanism information). |
| name | A convenient name of the entity (intended primarily for printing/logging/debugging). |
| type | Returns the utility_function type. |
| uuid | The unique ID of this entity |

Methods Summary
__call__(offer) Calculate the utility_function value for a given outcome.

approximate(ufuns, issues, n_outcomes[, ...])

param cls

compare(o1, o2) Compares the two outcomes and returns a measure of the difference between their utilities

conflict_level(u1, u2, outcomes[, max_tests]) Finds the conflict level in these two ufuns

create(*args, **kwargs) Creates an object and returns a proxy to it.

eu(offer) Calculate the expected utility_function value.

from_genius(file_name, **kwargs) Imports a utility function from a GENIUS XML file.

from_xml_str(xml_str[, domain_issues, ...]) Imports a utility function from a GENIUS XML string.

generate_bilateral(outcomes[, ...]) Generates a couple of utility functions

generate_random(n, outcomes[, normalized]) Generates a couple of utility functions

generate_random_bilateral(outcomes) Generates a couple of utility functions

to_genius(u, issues, file_name, **kwargs) Exports a utility function from a GENIUS XML file.

to_xml_str(u, issues[, discount_factor]) Exports a utility function to a well formatted string

winwin_level(u1, u2, outcomes[, max_tests]) Finds the conflict level in these two ufuns

xml(issues) Converts the function into a well formed XML string preferably in GENIUS format.

Attributes Documentation

base_type
Returns the utility_function base type ignoring discounting and similar wrappings.

Return type str

id
The unique ID of this entity

is_dynamic
Whether the utility function can potentially depend on negotiation state (mechanism information).

• If this property is False, the ufun can safely be assumed to be static (not dependent on negotiation state).

• If this property is True, the ufun may depend on negotiation state but it may also not depend on it.

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

type
Returns the utility_function type.

Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation
```
Returns  utility_function type
Return type  str

uuid
The unique ID of this entity

Methods Documentation

__call__(offer)
Calculate the utility_function value for a given outcome.

Parameters
offer  (Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be evaluated.

Remarks:
• You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
• Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns  The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.
Return type  UtilityValue

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters
• cls –
• ufuns (List[UtilityFunction]) –
• issues (Iterable[Issue]) –
• n_outcomes (int) –
• min_per_dim –
• force_single_issue –

Returns:
Return type  Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]]], List[Issue]]

compare(o1, o2)
Compares the two outcomes and returns a measure of the difference between their utilities

Return type  Union[Distribution, float]

classmethod conflict_level(ul, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns
Parameters

- \(u_1\) (UtilityFunction)
- \(u_2\) (UtilityFunction)

Examples

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.Random(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

- The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create(*args, **kwargs)

Creates an object and returns a proxy to it.

eu(offer)

Calculate the expected utility_function value.

Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float

classmethod from_genius(file_name, **kwargs)

Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)
Examples

```python
>>> from negmas import UtilityFunction
gg
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', ...
... 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0

Remarks: See from_xml_str for all the parameters

classmethod from_xml_str (xml_str, domain_issues=None, force_single_issue=False,
force_numeric=False, keep_issue_names=True, keep_value_names=True, normalize_utility=True,
max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)

Imports a utility function from a GENIUS XML string.

Parameters

• xml_str (str) – The string containing GENIUS style XML utility function definition

• domain_issues (List[Issue]) – Optional issue space to confirm that the utility function is valid

• force_single_issue (bool) – Tries to generate a MappingUtility function with a single issue which is the

• of all issues in the input (product) –

• keep_issue_names (bool) – Keep names of issues

• keep_value_names (bool) – Keep names of values

• safe_parsing (bool) – Turn on extra checks

• normalize_utility (bool) – Normalize the output utilities to the range from 0 to 1

• max_n_outcomes (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', ...
... 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False,
... force_numeric=False, keep_issue_names=True, keep_value_names=True,
... normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001
```
u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.68559475583014) < 0.000001

u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.98772736172488) < 0.000001

u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.68559475583014) < 0.000001
classmethod generate_bilateral(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

Parameters

- **n_outcomes (int)** – number of outcomes to use
- **conflict_level (float)** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- **conflict_delta** – How variable is the conflict at different outcomes.
- **zero_summness** – How zero-sum like are the two ufuns.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, ...
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, ...
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, ...
... , conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

Return type **Tuple[UtilityFunction, UtilityFunction]**

classmethod generate_random(n, outcomes, normalized=True)

Generates a couple of utility functions

Parameters

- **n (int)** – number of utility functions to generate
- **outcomes** (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]]]]]]) – number of outcomes to use
- **normalized (bool)** – if true, the resulting ufuns will be normalized between zero and one.

Return type **List[UtilityFunction]**

classmethod generate_random_bilateral(outcomes)

Generates a couple of utility functions

Parameters
**negmas.utilities Module**

- **n_outcomes (int)** – number of outcomes to use
- **conflict_level** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- **conflict_delta** – How variable is the conflict at different outcomes.
- **zero_summness** – How zero-sum like are the two ufuns.

**classmethod to_genius (u, issues, file_name, **kwargs)**

Exports a utility function from a GENIUS XML file.

**Parameters**

- **file_name (str)** – File name to export to
- **u (UtilityFunction)** – utility function
- **issues (List[Issue])** – The issues being considered as defined in the domain

**Returns** None

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-domain.xml'))
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name = pkg_resources.resource_filename('negmas', resource_name='tests/data/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See to_xml_str for all the parameters

**classmethod to_xml_str (u, issues, discount_factor=None)**

Exports a utility function to a well formatted string

**Return type** str

**classmethod winwin_level (u1, u2, outcomes, max_tests=10000)**

Finds the conflict level in these two ufuns

**Parameters**

- **u1 (UtilityFunction)** –
- **u2 (UtilityFunction)** –

**Examples**

- A nonlinear same ufun case
A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

Return type float

XML (issues)

Converts the function into a well formed XML string preferably in GENIUS format.

If the output has with </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in to_xml_str

Return type str

ComplexWeightedUtilityFunction

class negmas.utilities.ComplexWeightedUtilityFunction(ufuns, weights=None, name=None, reserved_value=None, ami=None)

Bases: negmas.utilities.UtilityFunction

A utility function composed of linear aggregation of other utility functions

Parameters

- ufuns (Iterable[UtilityFunction]) – An iterable of utility functions
- weights (Optional[Iterable[float]]) – Weights used for combination
- name – Utility function name

Attributes Summary

- base_type
  Returns the utility_function base type ignoring discounting and similar wrappings.
- id
  The unique ID of this entity
- is_dynamic
  Whether the utility function can potentially depend on negotiation state (mechanism information).
- name
  A convenient name of the entity (intended primarily for printing/logging/debugging).
- type
  Returns the utility_function type.
- uuid
  The unique ID of this entity

Methods Summary

- __call__(offer)
  Calculate the utility_function value for a given outcome.

Continued on next page
### Table 30 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compare(o1, o2)</code></td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td><code>conflict_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufun</td>
</tr>
<tr>
<td><code>create(*args, **kwargs)</code></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>eu(offer)</code></td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td><code>from_genius(file_name, **kwargs)</code></td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>from_xml_str(xml_str[, domain_issues, ...])</code></td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td><code>generate_bilateral(outcomes[,...])</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>generate_random(n, outcomes[, normalized])</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>generate_random_bilateral(outcomes)</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>to_genius(u, issues, file_name, **kwargs)</code></td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>to_xml_str(u, issues[, discount_factor])</code></td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td><code>winwin_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufun</td>
</tr>
<tr>
<td><code>xml(issues)</code></td>
<td>Converts the function into a well formed XML string preferably in GENIUS format.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

#### `base_type`

Returns the utility_function base type ignoring discounting and similar wrappings.

**Return type** `str`

#### `id`

The unique ID of this entity

#### `is_dynamic`

Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is `False`, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

#### `name`

A convenient name of the entity (intended primarily for printing/logging/debugging).

#### `type`

Returns the utility_function type.

Each class inheriting from this `UtilityFunction` class will have its own type. The default type is the empty string.

### Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1:lambda x:x, 2:lambda x:x}).type)  # type
linear_aggregation
>>> print(MappingUtilityFunction(lambda x: x).type)
mapping
```

(continues on next page)
```
>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: __
˓→x).type)
non_linear_aggregation
```

**Returns** utility_function type

**Return type** str

**uuid**
The unique ID of this entity

### Methods Documentation

__call__(offer)

Calculate the utility_function value for a given outcome.

**Parameters**

- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])* – The offer to be evaluated.

**Remarks:**

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

**Returns** The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

**Return type** UtilityValue

**classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)**

**Parameters**

- **cls** –
- **ufuns** *(List[UtilityFunction])*
- **issues** *(Iterable[Issue])*
- **n_outcomes** *(int)*
- **min_per_dim** –
- **force_single_issue** –

**Returns**

- **Return type** Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

**compare(o1, o2)**

Compares the two outcomes and returns a measure of the difference between their utilities

**Return type** Union[Distribution, float]

**classmethod conflict_level(u1, u2, outcomes, max_tests=10000)**

Finds the conflict level in these two ufuns

**Parameters**

- **u1** *(UtilityFunction)* –
**Examples**

- **u2:**

```python
u2 = UtilityFunction()

# A nonlinear strictly zero sum case
outcomes = [(_,) for _ in range(10)]
u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.randn(len(outcomes))))
u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0

# The same ufun
print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
0.0

# A linear strictly zero sum case
outcomes = [(_,) for _ in range(10)]
u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

**Return type:** float

**classmethod create(**args, **kwargs)**

Creates an object and returns a proxy to it.

**eu:**

Calculate the expected utility function value.

**Parameters** offer: (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

**Returns** The expected utility function for utility_priors and just utility_function for real-valued utilities.

**Return type:** float

**classmethod from_genius:**

Imports a utility function from a GENIUS XML file.

**Parameters** file_name (str) – File name to import from

**Returns** A utility function object (depending on the input file)

**Examples**
```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See from_xml_str for all the parameters

```python
classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
```

Imports a utility function from a GENIUS XML string.

Parameters

- **xml_str** *(str)* – The string containing GENIUS style XML utility function definition
- **domain_issues** *(List[Issue])* – Optional issue space to confirm that the utility function is valid
- **force_single_issue** *(bool)* – Tries to generate a MappingUtility function with a single issue which is the
- **of all issues in the input** *(product)* –
- **keep_issue_names** *(bool)* – Keep names of issues
- **keep_value_names** *(bool)* – Keep names of values
- **safe_parsing** *(bool)* – Turn on extra checks
- **normalize_utility** *(bool)* – Normalize the output utilities to the range from 0 to 1
- **max_n_outcomes** *(int)* – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001
```
```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+60 Gb+20'' LCD',)) - 22.6855475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.98772736172488) < 0.000001
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 21.98772736172488) < 0.000001
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD"}) - 22.6855475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+60 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01
```

(continues on next page)
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 1.0) < 0.0001

classmethod generate_bilateral(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

Parameters

- *n_outcomes* (*int*) - number of outcomes to use
- *conflict_level* (*float*) - How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- *conflict_delta* - How variable is the conflict at different outcomes.
- *zero_summness* - How zero-sum like are the two ufuns.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, ...
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, ...
... , conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, ...
... , conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

Return type  Tuple[
UtilityFunction, UtilityFunction]

classmethod generate_random(n, outcomes, normalized=True)

Generates a couple of utility functions

Parameters

- *n* (*int*) - number of utility functions to generate
- *outcomes* (*Union*[ int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]]]) - number of outcomes to use
- *normalized* (*bool*) - if true, the resulting ufuns will be normalized between zero and one.

Return type  List[UtilityFunction]

classmethod generate_random_bilateral(outcomes)

Generates a couple of utility functions

Parameters
NegMAS Documentation, Release 0.3.2

- `n_outcomes (int)` – number of outcomes to use
- `conflict_level` – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- `conflict_delta` – How variable is the conflict at different outcomes.
- `zero_summness` – How zero-sum like are the two ufuns.

**Return type** Tuple[UtilityFunction, UtilityFunction]

**classmethod to_genius**(*u, issues, file_name, **kwargs*)
Exports a utility function from a GENIUS XML file.

**Parameters**
- `file_name (str)` – File name to export to
- `u (UtilityFunction)` – utility function
- `issues (List[Issue])` – The issues being considered as defined in the domain

**Returns** None

**Examples**

```python
glam from negmas import UtilityFunction
glam from negmas import load_genius_domain
glam _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-domain.xml'), keep_issue_names=False)
glam u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
glam UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name=pkg_resources.resource_filename('negmas', 'data/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See *to_xml_str* for all the parameters

**classmethod to_xml_str**(*u, issues, discount_factor=None*)
Exports a utility function to a well formatted string

**Return type** str

**classmethod winwin_level**(*u1, u2, outcomes, max_tests=10000*)
Finds the conflict level in these two ufuns

**Parameters**
- `u1 (UtilityFunction)` –
- `u2 (UtilityFunction)` –

**Examples**

- A nonlinear same ufun case
```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, _len(outcomes), endpoint=True))))
```

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, _len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, _len(outcomes), endpoint=True))))
```

Return type `float`

`xml` *(issues)*

Converts the function into a well formed XML string preferably in GENIUS format.

If the output has with </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in **to_xml_str**

Return type `str`

*ComplexNonlinearUtilityFunction*

```python
```

Bases: *negmas.utilities.UtilityFunction*

A utility function composed of nonlinear aggregation of other utility functions

**Parameters**

• **ufuns** *(Iterable[UtilityFunction])* – An iterable of utility functions

• **combination_function** – The function used to combine results of ufuns

• **name** – Utility function name

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>base_type</code></td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td><code>is_dynamic</code></td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td><code>type</code></td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td><code>uuid</code></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**
__call__(offer)  
Calculate the utility_function value for a given outcome.

approximate(ufuns, issues, n_outcomes[,...])  
param cls

compare(o1, o2)  
Compares the two outcomes and returns a measure of the difference between their utilities

calculate_level(u1, u2, outcomes[, max_tests])  
Finds the conflict level in these two ufuns

create(*args, **kwargs)  
Creates an object and returns a proxy to it.

eu(offer)  
Calculate the expected utility_function value.

from_genius(file_name, **kwargs)  
Imports a utility function from a GENIUS XML file.

from_xml_str(xml_str[, domain_issues, ...])  
Imports a utility function from a GENIUS XML string.

generate_bilateral(outcomes[, ...])  
Generates a couple of utility functions

generate_random(n, outcomes[, normalized])  
Generates a couple of utility functions

generate_random_bilateral(outcomes)  
Generates a couple of utility functions

to_genius(u, issues, file_name, **kwargs)  
Exports a utility function from a GENIUS XML file.

to_xml_str(u, issues[, discount_factor])  
Exports a utility function to a well formatted string

winwin_level(u1, u2, outcomes[, max_tests])  
Finds the conflict level in these two ufuns

xml(issues)  
Converts the function into a well formed XML string preferably in GENIUS format.

Attributes Documentation

base_type  
Returns the utility_function base type ignoring discounting and similar wrappings.

Return type str

id  
The unique ID of this entity

is_dynamic  
Whether the utility function can potentially depend on negotiation state (mechanism information).

• If this property is False, the ufun can safely be assumed to be static (not dependent on negotiation state).

• If this property is True, the ufun may depend on negotiation state but it may also not depend on it.

name  
A convenient name of the entity (intended primarily for printing/logging/debugging).

type  
Returns the utility_function type.

Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x:x, 2: lambda x:x}).__type)
linear_aggregation
```
Returns  utility_function type  

Return type  str

type

uuid
The unique ID of this entity

Methods Documentation

__call__(offer)
Calculate the utility_function value for a given outcome.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offer</td>
<td>Union[OutcomeType, Tuple[Union[int, float, str, list]]], Dict[Union[int, str], Union[int, float, str, list]]</td>
<td>The offer to be evaluated.</td>
</tr>
</tbody>
</table>

Remarks:
- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

Return type UtilityValue

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cls</td>
<td>List[UtilityFunction]</td>
<td></td>
</tr>
<tr>
<td>ufun</td>
<td>List[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]</td>
<td></td>
</tr>
<tr>
<td>issues</td>
<td>Iterable[Issue]</td>
<td></td>
</tr>
<tr>
<td>n_outcomes</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>min_per_dim</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>force_single_issue</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

Returns:

Return type Tuple[List[MappingUtilityFunction], List[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]], List[Issue]]

compare (o1, o2)
Compares the two outcomes and returns a measure of the difference between their utilities

Return type Union[Distribution, float]

classmethod conflict_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns
Parameters

- \( u_1(UtilityFunction) \)
- \( u_2(UtilityFunction) \)

Examples

- A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.rand(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

- The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create(*args, **kwargs)

Creates an object and returns a proxy to it.

eu(offer)

Calculate the expected utility_function value.

Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float

classmethod from_genius(file_name, **kwargs)

Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)
Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See `from_xml_str` for all the parameters

```python
classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
```

Imports a utility function from a GENIUS XML string.

Parameters

- `xml_str (str)` – The string containing GENIUS style XML utility function definition
- `domain_issues (List[Issue])` – Optional issue space to confirm that the utility function is valid
- `force_single_issue (bool)` – Tries to generate a MappingUtility function with a single issue which is the
- `of all issues in the input (product)` –
- `keep_issue_names (bool)` – Keep names of issues
- `keep_value_names (bool)` – Keep names of values
- `safe_parsing (bool)` – Turn on extra checks
- `normalize_utility (bool)` – Normalize the output utilities to the range from 0 to 1
- `max_n_outcomes (int)` – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns

A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001
```
u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)

assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001

assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.6859475583014) < 0.000001

u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)

assert abs(u((0,)) - 21.98772736172488) < 0.000001

assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 21.98772736172488) < 0.000001

assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': '20'' LCD'}) - 22.6859475583014) < 0.000001

u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=True)

assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1

assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01

u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'tests/data/Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True)

assert abs(u((0,)) - 0.599329436957658) < 0.1

assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor': '19'' LCD'}) - 0.599329436957658) < 0.1

assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor': '20'' LCD'}) - 0.6342209804130308) < 0.01

(continues on next page)
classmethod generate_bilateral(outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)
Generates a couple of utility functions

Parameters

- n_outcomes (int) – number of outcomes to use
- conflict_level (float) – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
- conflict_delta – How variable is the conflict at different outcomes.
- zero_summness – How zero-sum like are the two ufuns.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0
```

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

Return type Tuple[UtilityFunction, UtilityFunction]

classmethod generate_random(n, outcomes, normalized=True)
Generates a couple of utility functions

Parameters

- n (int) – number of utility functions to generate
- outcomes (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]]]]]]) – number of outcomes to use
- normalized (bool) – if true, the resulting ufuns will be normalized between zero and one.

Return type List[UtilityFunction]

classmethod generate_random_bilateral(outcomes)
Generates a couple of utility functions

Parameters
• **n_outcomes** (*int*) – number of outcomes to use

• **conflict_level** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• **conflict_delta** – How variable is the conflict at different outcomes.

• **zero_summness** – How zero-sum like are the two ufuns.

**Return type** Tuple[UtilityFunction, UtilityFunction]

**classmethod to_genius** (*u, issues, file_name, **kwargs*)

Exports a utility function from a GENIUS XML file.

**Parameters**

• **file_name** (*str*) – File name to export to

• **u** (*UtilityFunction*) – utility function

• **issues** (*List[Issue]*) – The issues being considered as defined in the domain

**Returns** None

**Examples**

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-domain.xml'), keep_issue_names=False)
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-prof1.xml'), keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name = pkg_resources.resource_filename('negmas', 'data/LaptopConv/Laptop-C-prof1.xml'))
```

**Remarks:** See to_xml_str for all the parameters

**classmethod to_xml_str** (*u, issues, discount_factor=None*)

Exports a utility function to a well formatted string

**Return type** str

**classmethod winwin_level** (*u1, u2, outcomes, max_tests=10000*)

Finds the conflict level in these two ufuns

**Parameters**

• **u1** (*UtilityFunction*) –

• **u2** (*UtilityFunction*) –

**Examples**

• A nonlinear same ufun case

5.2. negmas.utilities Module
```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

* A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
```

Return type float

`xml` *(issues)*

Converts the function into a well formed XML string preferably in GENIUS format. If the output has with </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in `to_xml_str`

Return type str

**IPUtilityFunction**

class negmas.utilities.IPUtilityFunction(outcomes, distributions=None, issue_names=None, name=None, reserved_value=None, ami=None)

Bases: negmas.utilities.UtilityFunction

Independent Probabilistic Utility Function.

Parameters

- outcomes *(Iterable[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]])* – Iterable of outcomes
- distributions – distributions associated with the outcomes
- name – ufun name

Examples

```python
>>> f = IPUtilityFunction(outcomes=[('o1'), ('o2')], distributions=[UtilityDistribution(dtype='uniform', loc=0.0, scale=0.5), UtilityDistribution(dtype='uniform', loc=0.1, scale=0.5)])
>>> f(('o1',))
U(0.0, 0.5)
```

```python
>>> f = IPUtilityFunction(outcomes=[{'cost': 10, 'dist': 20}, {'cost': 10, 'dist': 30}], distributions=[UtilityDistribution(dtype='uniform', loc=0.0, scale=0.5), UtilityDistribution(dtype='uniform', loc=0.1, scale=0.5)])
>>> f({'cost': 10, 'dist': 30})
U(0.1, 0.6)
```
### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong> (offer)</td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td>approximate (ufuns, issues, n_outcomes[...])</td>
<td>param cls</td>
</tr>
<tr>
<td>compare (o1, o2)</td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td>conflict_level (u1, u2, outcomes[, max_tests[]])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>create (*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>distribution (outcome)</td>
<td>Returns the distribution associated with a specific outcome :type outcome: Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]] :param outcome:</td>
</tr>
<tr>
<td>eu (offer)</td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td>from_genius (file_name, **kwargs)</td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>from_mapping (mapping[, range, uncertainty, ...])</td>
<td>Generates a distribution from which u may have been sampled :type mapping: Dict[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]], float] :param mapping: mapping from outcomes to float values :type range: Tuple[float, float] :param range: range of the utility_function values</td>
</tr>
<tr>
<td>from_ufun (u[, range, uncertainty, variability])</td>
<td>Generates a distribution from which u may have been sampled :type u: MappingUtilityFunction :param u: :type range: Tuple[float, float] :param range: range of the utility_function values</td>
</tr>
<tr>
<td>from_xml_str (xml_str[, domain_issues, ...])</td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td>generate_bilateral (outcomes[...])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random (n, outcomes[, normalized])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random_bilateral (outcomes)</td>
<td>Generates a couple of utility functions</td>
</tr>
</tbody>
</table>
Table 34 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key(outcome)</code></td>
<td>Returns the key of the given outcome in self.distributions.</td>
</tr>
<tr>
<td><code>sample()</code></td>
<td>Samples the utility_function distribution to create a mapping utility function</td>
</tr>
<tr>
<td><code>to_genius(u, issues, file_name, **kwargs)</code></td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>to_xml_str(u, issues[, discount_factor])</code></td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td><code>winwin_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufuncs</td>
</tr>
<tr>
<td><code>xml(issues)</code></td>
<td>Converts the function into a well formed XML string preferably in GENIUS format.</td>
</tr>
</tbody>
</table>

Attributes Documentation

- **base_type**
  Returns the utility_function base type ignoring discounting and similar wrappings.

  **Return type**  
  str

- **id**
  The unique ID of this entity

- **is_dynamic**
  Whether the utility function can potentially depend on negotiation state (mechanism information).
  - If this property is `False`, the ufun can safely be assumed to be static (not dependent on negotiation state).
  - If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

- **name**
  A convenient name of the entity (intended primarily for printing/logging/debugging).

- **type**
  Returns the utility_function type.
  Each class inheriting from this `UtilityFunction` class will have its own type. The default type is the empty string.

Examples

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)  
linear_aggregation
>>> print(MappingUtilityFunction(lambda x: x).type)  
mapping
>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: x).type)  
non_linear_aggregation
```

  **Returns**  utility_function type

  **Return type**  str

- **uuid**
  The unique ID of this entity
Methods Documentation

__call__(offer)
Calculate the utility_function value for a given outcome.

Parameters
offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Remarks:
• You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
• Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

Returns
The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.
Return type UtilityValue

classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)

Parameters
• cls
• ufuns (List[UtilityFunction]) –
• issues (Iterable[Issue]) –
• n_outcomes (int) –
• min_per_dim –
• force_single_issue –

Returns:
Return type Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

compare(o1, o2)
Compares the two outcomes and returns a measure of the difference between their utilities

Return type Union[Distribution, float]

classmethod conflict_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns

Parameters
• u1 (UtilityFunction) –
• u2 (UtilityFunction) –

Examples

• A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.random(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
```
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0

• The same ufun

>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, _len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, _len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type `float`

classmethod `create(*args,**kwargs)`
Creates an object and returns a proxy to it.

distribution `(outcome)`
Returns the distribution associated with a specific outcome:

Parameters `outcome` *(OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]*) – The offer to be evaluated.

Returns:

Return type `Union[Distribution, float]`

eu `(offer)`
Calculate the expected utility value.

Parameters `offer` *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]*) – The offer to be evaluated.

Returns The expected utility for utility_priors and just utility for real-valued utilities.

Return type `float`

classmethod `from_genius(file_name,**kwargs)`
Imports a utility function from a GENIUS XML file.

Parameters `file_name` *(str]*) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resource.resource_filename('negmas', ...

>>> u.__class__.__name__
```

Remarks: See from_xml_str for all the parameters

**classmethod from_mapping** *(mapping, range=(0.0, 1.0), uncertainty=0.5, variability=0.0)*

Generates a distribution from which *u* may have been sampled:

```python
def from_mapping(mapping, range=(0.0, 1.0), uncertainty=0.5, variability=0.0):
    return IPUtilityFunction(mapping, range, uncertainty, variability)
```

Examples

- No uncertainty

```python
>>> mapping = {'o1': 0.3, 'o2': 0.7}
>>> p = IPUtilityFunction.from_mapping(mapping, uncertainty=0.0)
>>> print(p)
{'o1': U(0.3, 0.3), 'o2': U(0.7, 0.7)}
```

- Full uncertainty

```python
>>> mapping = {'o1': 0.3, 'o2': 0.7}
>>> p = IPUtilityFunction.from_mapping(mapping, uncertainty=1.0)
>>> print(p)
{'o1': U(0.0, 1.0), 'o2': U(0.0, 1.0)}
```

- Some uncertainty

```python
>>> mapping = {'o1': 0.3, 'o2': 0.7}
>>> p = IPUtilityFunction.from_mapping(mapping, uncertainty=0.1)
>>> [_.scale for _ in p.distributions.values()]
[0.1, 0.1]
>>> for k, v in p.distributions.items():
...    assert v.loc <= mapping[k]
```

Return type **IPUtilityFunction**

Returns a new IPUtilityFunction

**classmethod from_ufun** *(u, range=(0.0, 1.0), uncertainty=0.5, variability=0.0)*

Generates a distribution from which *u* may have been sampled:

```python
def from_ufun(u, range=(0.0, 1.0), uncertainty=0.5, variability=0.0):
    return IPUtilityFunction(u, range, uncertainty, variability)
```

Examples

- No uncertainty

```python
>>> u.reserved_value
0.0
>>> d
1.0
```
```python
>>> u = MappingUtilityFunction(mapping=dict(zip(('o1',), ('o2',)), [0.3, 0.7]))
>>> p = IPUtilityFunction.from_ufun(u, uncertainty=0.0)
>>> print(p)
{('o1',): U(0.3, 0.3), ('o2',): U(0.7, 0.7)}
```

• Full uncertainty

```python
>>> u = MappingUtilityFunction(mapping=dict(zip(('o1',), ('o2',)), [0.3, 0.7]))
>>> p = IPUtilityFunction.from_ufun(u, uncertainty=1.0)
>>> print(p)
{('o1',): U(0.0, 1.0), ('o2',): U(0.0, 1.0)}
```

• Some uncertainty

```python
>>> u = MappingUtilityFunction(mapping=dict(zip(('o1',), ('o2',)), [0.3, 0.7]))
>>> p = IPUtilityFunction.from_ufun(u, uncertainty=0.1)
>>> print([_.scale for _ in p.distributions.values()])
[0.1, 0.1]
>>> for k, v in p.distributions.items():
...    assert v.loc <= u(k)
```

Return type  IPUtilityFunction

Returns a new IPUtilityFunction

```
class method from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
```

Imports a utility function from a GENIUS XML string.

Parameters

• **xml_str**(str) – The string containing GENIUS style XML utility function definition

• **domain_issues**(List[Issue]) – Optional issue space to confirm that the utility function is valid

• **force_single_issue**(bool) – Tries to generate a MappingUtility function with a single issue which is the

• **of all issues in the input**(product)–

• **keep_issue_names**(bool) – Keep names of issues

• **keep_value_names**(bool) – Keep names of values

• **safe_parsing**(bool) – Turn on extra checks

• **normalize_utility**(bool) – Normalize the output utilities to the range from 0 to 1

• **max_n_outcomes**(int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)
Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

```
... keep_issue_names=False, keep_value_names=False, normalize_˓→utility=True)
>>> assert abs(u((0,)) - 0.599329436957658) < 0.1

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_˓→filename('negmas'...
˓→Laptop/Laptop-C-prof1.xml')
...
˓→'r').read(), force_single_issue=False, normalize_˓→utility=True)
>>> assert abs(u({'Laptop': 'Dell', 'Harddisk': '60 Gb', 'External Monitor':...
˓→'19'' LCD'}) - 0.599329436957658) < 0.1
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '80 Gb', 'External Monitor':...
˓→'20'' LCD'}) - 0.6342209804130308) < 0.01
>>> assert abs(u({'Laptop': 'HP', 'Harddisk': '60 Gb', 'External Monitor':...
˓→'19'' LCD'}) - 1.0) < 0.0001

classmethod generate_bilateral( outcomes, conflict_level=0.5, conflict_delta=0.005, ˓→win_win=0.5)
Generates a couple of utility functions

Parameters

• n_outcomes (int) – number of outcomes to use
• conflict_level (float) – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
• conflict_delta – How variable is the conflict at different outcomes.
• zero_summness – How zero-sum like are the two ufuns.

Examples

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_˓→level=0.0...
˓→win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_˓→level=1.0...
˓→win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_˓→level=0.5...
˓→win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True

Return type  Tuple[UtilityFunction, UtilityFunction]

classmethod generate_random( n, outcomes, normalized=True)
Generates a couple of utility functions

Parameters
• **n** (int) – number of utility functions to generate

• **outcomes** (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]) – number of outcomes to use

• **normalized** (bool) – if true, the resulting ufuns will be normalized between zero and one.

**Return type** List[UtilityFunction]

classmethod **generate_random_bilateral** (outcomes)
Generates a couple of utility functions

**Parameters**

• **n_outcomes** (int) – number of outcomes to use

• **conflict_level** – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• **conflict_delta** – How variable is the conflict at different outcomes.

• **zero_summness** – How zero-sum like are the two ufuns.

**Return type** Tuple[UtilityFunction, UtilityFunction]

**key** (outcome)
Returns the key of the given outcome in self.distributions.

**Parameters** outcome (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])

**Returns** tuple

**Examples:**

```python
>>> f = IPUtilityFunction(outcomes=[('o1',), ('o2',)])
... , distributions=[UtilityDistribution(dtype='uniform', loc=0.0, scale=0.5)]
... , UtilityDistribution(dtype='uniform', loc=0.1, scale=0.5))
... f.key({0: 'o1'})
... {'o1',}
... f.key({'o1'})
... {'o1',}
... f.distributions
... {'o1'},: U(0.0, 0.5), ('o2',): U(0.1, 0.6)
... f.distribution({'o1'})
U(0.0, 0.5)

>>> f = IPUtilityFunction(outcomes=[{'cost': 10, 'dist': 20}, {'dist': 30, 'cost': 10}]
... , distributions=[UtilityDistribution(dtype='uniform', loc=0.0, scale=0.5)]
... , UtilityDistribution(dtype='uniform', loc=0.1, scale=0.5))
... f.key({'dist': 30, 'cost': 10})
... (10, 30)
... f.key({'cost': 10, 'dist': 30})
... (10, 30)
... f.distributions
... (10, 20): U(0.0, 0.5), (10, 30): U(0.1, 0.6)
... f.distribution((10, 20.0))
U(0.0, 0.5)
... f.distribution({'cost': 10, 'dist': 20})
U(0.0, 0.5)
```
sample()
Samples the utility_function distribution to create a mapping utility function

Examples

```python
>>> import random
>>> f = IPUtilityFunction(outcomes=[('o1',), ('o2',)]
... , distributions=[UtilityDistribution(dtype='uniform', loc=0.0, scale=0.2)
... , UtilityDistribution(dtype='uniform', loc=0.4, scale=0.5)])
>>> u = f.sample()
>>> assert u(('o1',)) <= 0.2
>>> assert 0.4 <= u(('o2',)) <= 0.9
```

Return type: MappingUtilityFunction

Returns: MappingUtilityFunction
classmethod to_genius(u, issues, file_name, **kwargs)
Exports a utility function from a GENIUS XML file.

Parameters

- **file_name** (**str**) – File name to export to
- **u** (**UtilityFunction**) – utility function
- **issues** (**List[Issue]**) – The issues being considered as defined in the domain

Returns: None

Examples

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.
... resource_filename('negmas'...
... , resource_name='tests/
... data/Laptop/Laptop-C-domain.xml')
... , keep_issue_names=False)
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.
... resource_filename('negmas'...
... , resource_name='tests/
... data/Laptop/Laptop-C-prof1.xml')
... , keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount...
... , file_name = pkg_resources.resource_filename('negmas'...
... , resource_name='tests/
... data/LaptopConv/Laptop-C-prof1.xml'))
```

Remarks: See to_xml_str for all the parameters
classmethod to_xml_str(u, issues, discount_factor=None)
Exports a utility function to a well formatted string

Return type: **str**
classmethod winwin_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuns

Parameters
Examples

- A nonlinear same ufun case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, -len(outcomes), endpoint=True))))
```

- A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, -len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, -len(outcomes), endpoint=True))))
```

Return type float

xml (issues)

Converts the function into a well formed XML string preferably in GENIUS format.

If the output has </objective> then discount factor and reserved value should also be included If the output has </utility_space> it will not be appended in to_xml_str

Return type str

JavaUtilityFunction

```python
class negmas.utilities.JavaUtilityFunction(java_object, java_class_name, *args, **kwargs)
```

A utility function implemented in Java

Attributes Summary

- `base_type` Returns the utility_function base type ignoring discounting and similar wrappings.
- `id` The unique ID of this entity
- `is_dynamic` Whether the utility function can potentially depend on negotiation state (mechanism information).
- `name` A convenient name of the entity (intended primarily for printing/logging/debugging).
- `type` Returns the utility_function type.
- `uuid` The unique ID of this entity

Methods Summary

- `__call__` Calculate the utility_function value for a given outcome.
Table 36 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>approximate()</code></td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td><code>compare(o1, o2)</code></td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td><code>conflict_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td><code>create(*args, **kwargs)</code></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>eu(offer)</code></td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td><code>from_dict(java_object, *args, **kwargs)</code></td>
<td>Creates a Python object representing the corresponding Java object</td>
</tr>
<tr>
<td><code>from_genius(file_name, **kwargs)</code></td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>from_xml_str(xml_str[, domain_issues, ...])</code></td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td><code>generate_bilateral(outcomes[, ...])</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>generate_random(n, outcomes[, normalized])</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>generate_random_bilateral(outcomes)</code></td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td><code>init_java_bridge(java_object, java_class_name)</code></td>
<td>Initializes a connection to the java bridge creating a member called <code>java_object</code> that can be used to access the counterpart object in Java</td>
</tr>
<tr>
<td><code>to_genius(u, issues, file_name, **kwargs)</code></td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td><code>to_xml_str(u, issues[, discount_factor])</code></td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td><code>winwin_level(u1, u2, outcomes[, max_tests])</code></td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td><code>xml(issues)</code></td>
<td>Converts the function into a well formed XML string preferably in GENIUS format.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

**base_type**

Returns the utility_function base type ignoring discounting and similar wrappings.

**Return type**: `str`

**id**

The unique ID of this entity

**is_dynamic**

Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is `False`, the ufun can safely be assumed to be static (not dependent on negotiation state).
- If this property is `True`, the ufun may depend on negotiation state but it may also not depend on it.

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**type**

Returns the utility_function type.

Each class inheriting from this `UtilityFunction` class will have its own type. The default type is the empty string.
Examples

```python
>>> from negmas.utilities import *

>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation

>>> print(MappingUtilityFunction(lambda x: x).type)
mapping

>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: __x).type)
non_linear_aggregation
```

**Returns** utility_function type

**Return type** str

**uuid**
The unique ID of this entity

**Methods Documentation**

___call___(offer)

Calculate the utility_function value for a given outcome.

**Parameters**

- **offer** (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

**Remarks:**

- You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.
- Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

**Returns** The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

**Return type** UtilityValue

**classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)**

**Parameters**

- **cls** –
- **ufuns** (List[UtilityFunction]) –
- **issues** (Iterable[Issue]) –
- **n_outcomes** (int) –
- **min_per_dim** –
- **force_single_issue** –

**Returns:**

**Return type** Tuple[List[MappingUtilityFunction], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

**compare(o1, o2)**

Compares the two outcomes and returns a measure of the difference between their utilities
Return type Union[Distribution, float]

classmethod conflict_level(ul1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufun.

Parameters
• u1 (UtilityFunction) –
• u2 (UtilityFunction) –

Examples
• A nonlinear strictly zero sum case

```
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.random(len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1._mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

• The same ufun

```
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u1, outcomes=outcomes))
0.0
```

• A linear strictly zero sum case

```
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

Return type float

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

eu(offer)
Calculate the expected utility_function value.

Parameters offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float

classmethod from_dict(java_object, *args, **kwargs)
Creates a Python object representing the corresponding Java object.

classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from
Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'))
>>> u.__class__.__name__
'LinearUtilityAggregationFunction'
>>> u.reserved_value
0.0
>>> d
1.0
```

Remarks: See from_xml_str for all the parameters

classmethod from_xml_str (xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)

Imports a utility function from a GENIUS XML string.

Parameters

- **xml_str (str)** – The string containing GENIUS style XML utility function definition
- **domain_issues (List[Issue])** – Optional issue space to confirm that the utility function is valid
- **force_single_issue (bool)** – Tries to generate a MappingUtility function with a single issue which is the
  - **of all issues in the input (product)**–
- **keep_issue_names (bool)** – Keep names of issues
- **keep_value_names (bool)** – Keep names of values
- **safe_parsing (bool)** – Turn on extra checks
- **normalize_utility (bool)** – Normalize the output utilities to the range from 0 to 1
- **max_n_outcomes (int)** – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)

Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
```

(continues on next page)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml')), resource_name='tests/data/Laptop/Laptop-C-prof1.xml', 'r').read(), force_single_issue=True
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 21.98772736172488) < 0.000001
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 22.68559475583014) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml')), resource_name='tests/data/Laptop/Laptop-C-prof1.xml', 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=False)
>>> assert abs(u((0,)) - 21.98772736172488) < 0.000001

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml')), resource_name='tests/data/Laptop/Laptop-C-prof1.xml', 'r').read(), force_single_issue=False, normalize_utility=False)
>>> assert abs(u(('Dell+60 Gb+19'' LCD',)) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP+80 Gb+20'' LCD',)) - 0.6342209804130308) < 0.01

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml')), resource_name='tests/data/Laptop/Laptop-C-prof1.xml', 'r').read(), force_single_issue=True, normalize_utility=True)
>>> assert abs(u((0,)) - 0.599329436957658) < 0.1

>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml')), resource_name='tests/data/Laptop/Laptop-C-prof1.xml', 'r').read(), force_single_issue=True, keep_issue_names=False, keep_value_names=False, normalize_utility=True)
>>> assert abs(u((0,)) - 0.599329436957658) < 0.1

(continues on next page)
class method generate_bilateral

Generates a couple of utility functions

Parameters

- `n_outcomes (int)` – number of outcomes to use
- `conflict_level (float)` – How conflicting are the two ufuncs to generate. 1.0 means maximum conflict.
- `conflict_delta` – How variable is the conflict at different outcomes.
- `zero_summness` – How zero-sum like are the two ufuncs.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, conflict_delta=0.0, win_win=1.0)
0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
```

Return type `Tuple[UtilityFunction, UtilityFunction]`

class method generate_random

Generates a couple of utility functions

Parameters

- `n (int)` – number of utility functions to generate
- `outcomes (Union[int, List[Union[OutcomeType, Tuple[Union[int, list]]]], Dict[Union[int, List[Union[int, str, list]]], Union[int, float, str, list]]])` – number of outcomes to use
- `normalized (bool)` – if true, the resulting ufuns will be normalized between zero and one.
Return type  List[UtilityFunction]

classmethod generate_random_bilateral(outcomes)
Generates a couple of utility functions

Parameters

• n_outcomes (int) – number of outcomes to use
• conflict_level – How conflicting are the two ufun to generate. 1.0 means maximum conflict.
• conflict_delta – How variable is the conflict at different outcomes.
• zero_summness – How zero-sum like are the two ufun.

Return type  Tuple[UtilityFunction, UtilityFunction]

init_java_bridge(java_object, java_class_name, auto_load_java=False,
python_shadow_object=None)
initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

Parameters

• java_object – A java object that already exists of the correct type. If given no new objects will be created
• java_class_name (str) – The type of the Java object to be created
• auto_load_java (bool) – When true, a JVM will be automatically created (if one is not available)
• python_shadow_object (Optional[Any]) – A python object to shadow the java object. The object will just call the corresponding
• on this shadow object whenever it needs. (method)

Remarks:

• sets a member called java_object that can be used to access the corresponding Java object created
• if python_shadow_object is given, it must be an object of a type that has an internal class called Java which has a single member called ‘implements’ which is a list of one string element representing the Java interface being implemented (it must be either jnegmas.PyCallable or an extension of it).

classmethod to_genius(u, issues, file_name, **kwargs)
Exports a utility function from a GENIUS XML file.

Parameters

• file_name (str) – File name to export to
• u (UtilityFunction) – utility function
• issues (List[Issue]) – The issues being considered as defined in the domain

Returns  None

Examples

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, _, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-domain.xml'),
                                             resource_name='tests/
                                             data/Laptop/Laptop-C-domain.xml'),
                                             keep_issue_names=False)
```
NegMAS Documentation, Release 0.3.2

>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', 'data/Laptop/Laptop-C-prof1.xml'))
... keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, file_name=pkg_resources.resource_filename('negmas', 'data/LaptopConv/Laptop-C-prof1.xml'))

Remarks: See to_xml_str for all the parameters

classmethod to_xml_str(u, issues, discount_factor=None)
Exports a utility function to a well formatted string

  Return type  str

classmethod winwin_level(u1, u2, outcomes, max_tests=10000)
Finds the conflict level in these two ufuncs

  Parameters
  • u1 (UtilityFunction)
  • u2 (UtilityFunction)

Examples

  • A nonlinear same ufun case

>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))

  • A linear strictly zero sum case

>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))

  Return type  float

xml(issues)
Converts the function into a well formed XML string preferably in GENIUS format.

If the output has with </objective> then discount factor and reserved value should also be included If
the output has </utility_space> it will not be appended in to_xml_str

  Return type  str

RandomUtilityFunction

class negmas.utilities.RandomUtilityFunction(outcomes)
  Bases: negmas.utilities.MappingUtilityFunction

  A random utility function for a discrete outcome space
Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_type</td>
<td>Returns the utility_function base type ignoring discounting and similar wrappings.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>is_dynamic</td>
<td>Whether the utility function can potentially depend on negotiation state (mechanism information).</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>type</td>
<td>Returns the utility_function type.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(offer)</td>
<td>Calculate the utility_function value for a given outcome.</td>
</tr>
<tr>
<td>approximate(ufuns, issues, n_outcomes[, ...])</td>
<td>param cls</td>
</tr>
<tr>
<td>compare(o1, o2)</td>
<td>Compares the two outcomes and returns a measure of the difference between their utilities</td>
</tr>
<tr>
<td>conflict_level(u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>eu(offer)</td>
<td>Calculate the expected utility_function value.</td>
</tr>
<tr>
<td>from_genius(file_name, **kwargs)</td>
<td>Imports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>from_xml_str(xml_str[, domain_issues, ...])</td>
<td>Imports a utility function from a GENIUS XML string.</td>
</tr>
<tr>
<td>generate_bilateral(outcomes[, ...])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random(n, outcomes[, normalized])</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>generate_random_bilateral(outcomes)</td>
<td>Generates a couple of utility functions</td>
</tr>
<tr>
<td>to_genius(u, issues, file_name, **kwargs)</td>
<td>Exports a utility function from a GENIUS XML file.</td>
</tr>
<tr>
<td>to_xml_str(u, issues[, discount_factor])</td>
<td>Exports a utility function to a well formatted string</td>
</tr>
<tr>
<td>winwin_level(u1, u2, outcomes[, max_tests])</td>
<td>Finds the conflict level in these two ufuns</td>
</tr>
<tr>
<td>xml(issues)</td>
<td></td>
</tr>
</tbody>
</table>

Examples

Attributes Documentation

base_type
Returns the utility_function base type ignoring discounting and similar wrappings.

Return type str

id
The unique ID of this entity

is_dynamic
Whether the utility function can potentially depend on negotiation state (mechanism information).

- If this property is False, the ufun can safely be assumed to be static (not dependent on negotiation state).
• If this property is True, the ufun may depend on negotiation state but it may also not depend on it.

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**type**

Returns the utility_function type.

Each class inheriting from this UtilityFunction class will have its own type. The default type is the empty string.

**Examples**

```python
>>> from negmas.utilities import *
>>> print(LinearUtilityAggregationFunction({1: lambda x: x, 2: lambda x: x}).type)
linear_aggregation
>>> print(MappingUtilityFunction(lambda x: x).type)
mapping
>>> print(NonLinearUtilityAggregationFunction({1: lambda x: x}, f=lambda x: -x).type)
non_linear_aggregation
```

Returns utility_function type

Return type str

**uuid**

The unique ID of this entity

**Methods Documentation**

**__call__(offer)**

Calculate the utility_function value for a given outcome.

**Parameters**

offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be evaluated.

**Remarks:**

• You cannot return None from overridden apply() functions but raise an exception (ValueError) if it was not possible to calculate the UtilityValue.

• Return A UtilityValue not a float for real-valued utilities for the benefit of inspection code.

**Returns** The utility_function value which may be a distribution. If None it means the utility_function value cannot be calculated.

Return type UtilityValue

**classmethod approximate(ufuns, issues, n_outcomes, min_per_dim=5, force_single_issue=False)**

**Parameters**

• cls –

• ufuns (List[UtilityFunction]) –

• issues (Iterable[Issue]) –
• **n_outcomes** (int) –
• **min_per_dim** –
• **force_single_issue** –

Returns:

**Return type** Tuple[List[MappingUtilityFunction],
List[Union[OutcomeType, Tuple[Union[int, float, str, list]]]], Dict[Union[int, str], Union[dict, list, Dict[Union[int, str], Union[int, float, str, list]]]], List[Issue]]

**compare**(o1, o2)

Compares the two outcomes and returns a measure of the difference between their utilities

**Return type** Union[Distribution, float]

**classmethod conflict_level**(u1, u2, outcomes, max_tests=10000)

Finds the conflict level in these two ufun

**Parameters**

• **u1** *(UtilityFunction)* –
• **u2** *(UtilityFunction)* –

**Examples**

• A nonlinear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.random.randint(0, 10, len(outcomes)))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, 1.0 - np.array(list(u1.mapping.values())))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

• The same ufun

```python
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
0.0
```

• A linear strictly zero sum case

```python
>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))
>>> print(UtilityFunction.conflict_level(u1=u1, u2=u2, outcomes=outcomes))
1.0
```

**Return type** float

**classmethod create**( *args, **kwargs)

Creates an object and returns a proxy to it.

**eu**(offer)

Calculate the expected utility_function value.
Parameters offer `(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])` – The offer to be evaluated.

Returns The expected utility_function for utility_priors and just utility_function for real-valued utilities.

Return type float

classmethod from_genius(file_name, **kwargs)
Imports a utility function from a GENIUS XML file.

Parameters file_name (str) – File name to import from

Returns A utility function object (depending on the input file)

Examples

```python
>>> from negmas import UtilityFunction
>>> u, d = UtilityFunction.from_genius(file_name = pkg_resources.resource_filename('negmas'...

Remarks: See from_xml_str for all the parameters

classmethod from_xml_str(xml_str, domain_issues=None, force_single_issue=False, force_numeric=False, keep_issue_names=True, keep_value_names=True, safe_parsing=True, normalize_utility=True, max_n_outcomes=1000000.0, ignore_discount=False, ignore_reserved=False)
Imports a utility function from a GENIUS XML string.

Parameters

• xml_str (str) – The string containing GENIUS style XML utility function definition

• domain_issues (List[Issue]) – Optional issue space to confirm that the utility function is valid

• force_single_issue (bool) – Tries to generate a MappingUtility function with a single issue which is the

• of all issues in the input (product) –

• keep_issue_names (bool) – Keep names of issues

• keep_value_names (bool) – Keep names of values

• safe_parsing (bool) – Turn on extra checks

• normalize_utility (bool) – Normalize the output utilities to the range from 0 to 1

• max_n_outcomes (int) – Maximum number of outcomes allowed (effective only if force_single_issue is True)

Returns A utility function object (depending on the input file)
Examples

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=True, keep_issue_names=False, keep_value_names=True)
>>> assert abs(u(('Dell', '60 Gb', "19'' LCD")) - 0.599329436957658) < 0.1
>>> assert abs(u(('HP', '80 Gb', "20'' LCD")) - 0.6342209804130308) < 0.01
>>> assert abs(u(('HP', '60 Gb', "19'' LCD")) - 1.0) < 0.0001

```

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False)
>>> assert abs(u(("Dell+60 Gb+19'' LCD")) - 21.98772736172488) < 0.000001
>>> assert abs(u("HP+80 Gb+20'' LCD")) - 22.68559475583014 < 0.000001

```

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=True, normalize_utility=False, keep_issue_names=False, keep_value_names=False)
>>> assert abs(u((0,)) - 21.987727736172488) < 0.000001

```

```python
>>> u, _ = UtilityFunction.from_xml_str(open(pkg_resources.resource_filename('negmas', 'Laptop/Laptop-C-prof1.xml'), 'r').read(), force_single_issue=False, normalize_utility=False)
>>> assert abs(u({"Laptop": 'Dell', 'Harddisk': '60 Gb', 'External Monitor': "19'' LCD"}) - 21.987727736172488) < 0.000001
>>> assert abs(u({"Laptop": 'HP', 'Harddisk': '80 Gb', 'External Monitor': "20'' LCD"}) - 22.68559475583014) < 0.000001

```

(continues on next page)
classmethod generate_bilateral (outcomes, conflict_level=0.5, conflict_delta=0.005, win_win=0.5)

Generates a couple of utility functions

Parameters

• **n_outcomes** (int) – number of outcomes to use
• **conflict_level** (float) – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.
• **conflict_delta** – How variable is the conflict at different outcomes.
• **zero_summness** – How zero-sum like are the two ufuns.

Examples

```python
>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
0.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=1.0, conflict_delta=0.0, win_win=0.0)
>>> print(UtilityFunction.conflict_level(u1, u2, outcomes=10))
1.0

>>> u1, u2 = UtilityFunction.generate_bilateral(outcomes=10, conflict_level=0.5, conflict_delta=0.0, win_win=1.0)
>>> 0.0 <= UtilityFunction.conflict_level(u1, u2, outcomes=10) <= 1.0
True
```

Return type Tuple[UtilityFunction, UtilityFunction]

classmethod generate_random (n, outcomes, normalized=True)

Generates a couple of utility functions

Parameters
• n (int) – number of utility functions to generate

• outcomes (Union[int, List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]) – number of outcomes to use

• normalized (bool) – if true, the resulting ufuns will be normalized between zero and one.

Return type List[UtilityFunction]

classmethod generate_random_bilateral(outcomes)
Generates a couple of utility functions

Parameters

• n_outcomes (int) – number of outcomes to use

• conflict_level – How conflicting are the two ufuns to generate. 1.0 means maximum conflict.

• conflict_delta – How variable is the conflict at different outcomes.

• zero_summness – How zero-sum like are the two ufuns.

Return type Tuple[UtilityFunction, UtilityFunction]

classmethod to_genius(u, issues, file_name, **kwargs)
Exports a utility function from a GENIUS XML file.

Parameters

• file_name (str) – File name to export to

• u(UtilityFunction) – utility function

• issues(List[Issue]) – The issues being considered as defined in the domain

Returns None

Examples

```python
>>> from negmas import UtilityFunction
>>> from negmas import load_genius_domain
>>> _, __, issues = load_genius_domain(domain_file_name=pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-domain.xml'), keep_issue_names=False)
>>> u, discount = UtilityFunction.from_genius(file_name=pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop/Laptop-C-prof1.xml'), ... keep_issue_names=False)
>>> UtilityFunction.to_genius(u=u, issues=issues, discount_factor=discount, ... file_name = pkg_resources.resource_filename('negmas', ... resource_name='tests/data/LaptopConv/Laptop-C-prof1.xml'))
```

Remarks: See to_xml_str for all the parameters

classmethod to_xml_str(u, issues, discount_factor=None)
Exports a utility function to a well formatted string

Return type str
classmethod winwin_level(u1, u2, outcomes, max_tests=10000)
    Finds the conflict level in these two ufuns

Parameters
    • u1 (UtilityFunction)
    • u2 (UtilityFunction)

Examples
    • A nonlinear same ufun case

>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))

    • A linear strictly zero sum case

>>> outcomes = [(_,) for _ in range(10)]
>>> u1 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(0.0, 1.0, len(outcomes), endpoint=True))))
>>> u2 = MappingUtilityFunction(dict(zip(outcomes, np.linspace(1.0, 0.0, len(outcomes), endpoint=True))))

Return type float

xml(issues)

Examples
    >>> issue = Issue(values=['to be', 'not to be'], name='THE problem')
    >>> print(str(issue))
    THE problem: ['to be', 'not to be']
    >>> f = MappingUtilityFunction({'to be':10.0, 'not to be':0.0})
    >>> print(list(map(f, ['to be', 'not to be'])))
    [10.0, 0.0]
    >>> print(f.xml([issue]))
    <issue index="1" etype="discrete" type="discrete" vtype="discrete" name="THE problem">
        <item index="1" value="to be" cost="0" evaluation="10.0" description="to be"></item>
        <item index="2" value="not to be" cost="0" evaluation="0.0" description="not to be"></item>
    </issue>
    <weight index="1" value="1.0"> </weight>

Return type str
5.3 negmas.mechanisms Module

Provides interfaces for defining negotiation mechanisms.

5.3.1 Classes

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Protocol</th>
<th>MechanismRoundResult</th>
</tr>
</thead>
<tbody>
<tr>
<td>([issues, outcomes, n_steps, ...])</td>
<td>Base class for all negotiation Mechanisms.</td>
<td>An alias for Mechanism</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
<td>([broken, timeout, ...])</td>
</tr>
</tbody>
</table>
Mechanism

class negmas.mechanisms.Mechanism(issues=None, outcomes=None, n_steps=None, time_limit=None, step_time_limit=None, max_n_agents=None, dynamic_entry=False, cache_outcomes=True, max_n_outcomes=1000000, keep_issue_names=True, annotation=None, state_factory=<class 'negmas.common.MechanismState'>, enable_callbacks=False, name=None)

Bases: negmas.common.NamedObject, negmas.events.EventSource, abc.ABC

Base class for all negotiation Mechanisms.

Override the round function of this class to implement a round of your mechanism

Attributes Summary

agreement
all
completed
dynamic_entry
history
id
issues
max_n_agents
n_outcomes
n_steps
name
	negotiators
outcomes
participants
relative_time
remaining_steps
remaining_time
requirements
running
state

time
time_limit
uuid

Methods Summary

add(negotiator, *[ufun, role])
add_requirements(requirements)
announce(event)
can_accept_more_agents()
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_enter(agent)</td>
<td>Whether the agent can enter the negotiation now.</td>
</tr>
<tr>
<td>can_leave(agent)</td>
<td>Can the agent leave now?</td>
</tr>
<tr>
<td>can_participate(agent)</td>
<td>Checks if the agent can participate in this type of negotiation in general.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>discrete_outcomes([n_max, astype])</td>
<td>A discrete set of outcomes that spans the outcome space.</td>
</tr>
<tr>
<td>extra_state()</td>
<td>Returns any extra state information to be kept in the state and history properties</td>
</tr>
<tr>
<td>get_info(id)</td>
<td>Returns the mechanism information which contains its static config plus methods to access current state</td>
</tr>
<tr>
<td>is_satisfying(capabilities)</td>
<td>Checks if the given capabilities are satisfying mechanism requirements.</td>
</tr>
<tr>
<td>is_valid(outcome)</td>
<td>Checks whether the outcome is valid given the issues</td>
</tr>
<tr>
<td>on_mechanism_error()</td>
<td>Called when there is a mechanism error</td>
</tr>
<tr>
<td>on_negotiation_end()</td>
<td>Called at the end of each negotiation</td>
</tr>
<tr>
<td>on_negotiation_start()</td>
<td>Called before starting the negotiation.</td>
</tr>
<tr>
<td>pareto_frontier([n_max, sort_by_welfare])</td>
<td>rtype Tuple[List[Tuple[float]], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]</td>
</tr>
</tbody>
</table>
n_steps

name

A convenient name of the entity (intended primarily for printing/logging/debugging).

negotiators

outcomes

participants

Returns a list of all participant names

    Return type List[NegotiatorInfo]

relative_time

Returns a number between 0 and 1 indicating elapsed relative time or steps.

    Return type Optional[float]

remaining_steps

Returns the remaining number of steps until the end of the mechanism run. None if unlimited

    Return type Optional[int]

remaining_time

Returns remaining time in seconds. None if no time limit is given.

    Return type Optional[float]

requirements

A dictionary specifying the requirements that must be in the capabilities of any agent to join the mechanism.

running

state

Returns the current state. Override extra_state if you want to keep extra state

time

Elapsed time since mechanism started in seconds. None if the mechanism did not start running

    Return type Optional[float]

time_limit

uuid

The unique ID of this entity

Methods Documentation

add (negotiator, *, ufun=None, role=None, **kwargs)

Add an agent to the negotiation.

Parameters

• negotiator (Negotiator) – The agent to be added.

• ufun (Optional[UtilityFunction]) – The utility function to use. If None, then the agent must already have a stored utility function otherwise it will fail to enter the negotiation.

• role (Optional[str]) – The role the agent plays in the negotiation mechanism. It is expected that mechanisms inheriting from this class will check this parameter to ensure that the role is a valid role and is still possible for negotiators to join on that role. Roles may include things like moderator, representative etc based on the mechanism

    Return type Optional[bool]
Returns

- True if the agent was added.
- False if the agent was already in the negotiation.
- None if the agent cannot be added.

add_requirements(requirements)

Adds requirements.

Return type None

announce(event)

Raises an event and informs all event sinks that are registered for notifications on this event type

can_accept_more_agents()

Whether the mechanism can currently accept more negotiators.

Return type bool

can_enter(agent)

Whether the agent can enter the negotiation now.

Return type bool

can_leave(agent)

Can the agent leave now?

Return type bool

can_participate(agent)

Checks if the agent can participate in this type of negotiation in general.

Parameters

agent (Negotiator) –

Returns

True if it can

Return type bool

Remarks: The only reason this may return False is if the mechanism requires some requirements that are not within the capabilities of the agent.

When evaluating compatibility, the agent is considered incapable of participation if any of the following conditions hold:

* A mechanism requirement is not in the capabilities of the agent
* A mechanism requirement is in the capabilities of the agent by the values required for it is not in the values announced by the agent.

An agent that lists a None value for a capability is announcing that it can work with all its values. On the other hand, a mechanism that lists a requirement as None announces that it accepts any value for this requirement as long as it exist in the agent

classmethod create(*args, **kwargs)

Creates an object and returns a proxy to it.

discrete_outcomes(n_max=None, astype=<class 'dict'>)

A discrete set of outcomes that spans the outcome space

Parameters

- n_max (Optional[int]) – The maximum number of outcomes to return. If None, all outcomes will be returned for discrete issues
- 100 if any of the issues was continuous (and) –
- astype (Type[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]) – A type to cast the resulting outcomes to.

Returns

List of n or less outcomes
### extra_state()

Returns any extra state information to be kept in the `state` and `history` properties.

**Return type** `Optional[Dict[str, Any]]`

### classmethod get_info(id)

Returns the mechanism information which contains its static config plus methods to access current state.

**Return type** `AgentMechanismInterface`

### is_satisfying(capabilities)

Checks if the given capabilities are satisfying mechanism requirements.

**Parameters**
- `capabilities (dict)` – capabilities to check

**Return type** `bool`

**Returns**
- `bool` are the requirements satisfied by the capabilities.

**Remarks:**
- Requirements are also a dict with the following meanings:
  - `tuple`: Min and max acceptable values
  - `list/set`: Any value in the iterable is acceptable
  - `Single value`: The capability must match this value
- Capabilities can also have the same three possibilities.

### is_valid(outcome)

Checks whether the outcome is valid given the issues.

### on_mechanism_error()

Called when there is a mechanism error.

**Remarks:**
- When overriding this function you **MUST** call the base class version

**Return type** `None`

### on_negotiation_end()

Called at the end of each negotiation.

**Remarks:**
- When overriding this function you **MUST** call the base class version

**Return type** `None`

### on_negotiation_start()

Called before starting the negotiation. If it returns False then negotiation will end immediately.

**Return type** `bool`

### pareto_frontier(n_max=None, sort_by_welfare=True)

**Return type**
- `Tuple[
  List[Tuple[float]],
  List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]`

### plot(plot-utils=True, plot_outcomes=True)

### random_outcomes(n=1, astype=<class 'dict'>)

Returns random offers
register_listener (event_type, listener)

remove (agent, **kwargs)

Parameters
agent (Negotiator) –

Return type
Optional[bool]

Returns
• True if the agent was removed.
• False if the agent was not in the negotiation already.
• None if the agent cannot be removed.

remove_requirements (requirements)

Adds requirements.

Return type
None

round ()

Implements a single step of the mechanism. Override this!

Return type
MechanismRoundResult

Returns
MechanismRoundResult giving whether the negotiation was broken or timedout
and the agreement if any.

run (timeout=None)

Return type
MechanismState

step ()

Runs a single step of the mechanism.

Returns
The state of the negotiation after the round is conducted

Return type
MechanismState

Remarks:
• Every call yields the results of one round (see round ())
• If the mechanism was yet to start, it will start it and runs one round
• There is another function (run ()) that runs the whole mechanism in blocking mode

Protocol

negmas.mechanisms.Protocol
An alias for Mechanism

alias of negmas.mechanisms.Mechanism

MechanismRoundResult

class negmas.mechanisms.MechanismRoundResult (broken=False, timedout=False, agreement=None, error=False, error_details=“”)

Bases: object
Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agreement</td>
<td>The agreement if any</td>
</tr>
<tr>
<td>broken</td>
<td>True only if END_NEGOTIATION was selected by one agent</td>
</tr>
<tr>
<td>error</td>
<td>True if an error occurred in the mechanism</td>
</tr>
<tr>
<td>error_details</td>
<td>Error message</td>
</tr>
<tr>
<td>timedout</td>
<td>True if a timeout occurred. Usually not used</td>
</tr>
</tbody>
</table>

Attributes Documentation

```python
agreement = None
    The agreement if any

broken = False
    True only if END_NEGOTIATION was selected by one agent

error = False
    True if an error occurred in the mechanism

error_details = ''
    Error message

timedout = False
    True if a timeout occurred. Usually not used
```

5.3.2 Class Inheritance Diagram

```
MechanismRoundResult

ABC
    └── EventSource
        └── Mechanism
            └── NamedObject
```

5.3. negmas.mechanisms Module
5.4 negmas.common Module

Common data-structures and classes used by all other modules.
This module does not import anything from the library except during type checking

5.4.1 Functions

```
register_all_mechanisms(mechanisms) registers the running mechanisms.
```

```
register_all_mechanisms
negmas.common.register_all_mechanisms(mechanisms)
registers the running mechanisms. Used internally. DO NOT CALL THIS.
Return type: None
```

5.4.2 Classes

```
NamedObject([name]) The base class of all named entities.
AgentMechanismInterface(id, n_outcomes, ...) All information of a negotiation visible to negotiators.
MechanismState([running, started, step, ...]) Encapsulates the mechanism state at any point
NegotiatorInfo(name, id, type)
_ShadowAgentMechanismInterface(ami) Used to represent an AMI to Java.
```

```
NamedObject

class negmas.common.NamedObject(name=None)
    Bases: object
    The base class of all named entities.
    All named entities need to call this class’s __init__() somewhere during initialization.
    Parameters name (str) -- The given name of the entity. Notice that the class will add this to a base that depends on the child’s class name.

    Attributes Summary

    id The unique ID of this entity
    name A convenient name of the entity (intended primarily for printing/logging/debugging).
    uuid The unique ID of this entity

    Methods Summary

    create(*args, **kwargs) Creates an object and returns a proxy to it.

    Attributes Documentation

    id The unique ID of this entity
```
name
A convenient name of the entity (intended primarily for printing/logging/debugging).

uuid
The unique ID of this entity

Methods Documentation

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

AgentMechanismInterface

class negmas.common.AgentMechanismInterface(id, n_outcomes, issues, outcomes, time_limit, step_time_limit, n_steps, dynamic_entry, max_n_agents, annotation=<factory>)

Bases: object
All information of a negotiation visible to negotiators.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_negotiators</td>
<td>Syntactic sugar for state.n_agents</td>
</tr>
<tr>
<td>participants</td>
<td>rtype List[NegotiatorInfo]</td>
</tr>
<tr>
<td>requirements</td>
<td>The protocol requirements</td>
</tr>
<tr>
<td>state</td>
<td>Access the current state of the mechanism.</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asdict()</td>
<td>Converts the object to a dict containing all fields</td>
</tr>
<tr>
<td>discrete_outcomes([n_max, astype])</td>
<td>A discrete set of outcomes that spans the outcome space</td>
</tr>
<tr>
<td>keys()</td>
<td></td>
</tr>
<tr>
<td>outcome_index(outcome)</td>
<td>The index of an outcome</td>
</tr>
<tr>
<td>random_outcomes([n, astype])</td>
<td>A set of random outcomes from the issues of this negotiation</td>
</tr>
<tr>
<td>values()</td>
<td></td>
</tr>
</tbody>
</table>

Attributes Documentation

n_negotiators
Syntactic sugar for state.n_agents

Return type int

participants

Return type List[NegotiatorInfo]

requirements
The protocol requirements

Return type dict

Returns
NegMAS Documentation, Release 0.3.2

• A dict of str/Any pairs giving the requirements

**state**

Access the current state of the mechanism.

Remarks:

• Whenever a method receives a `AgentMechanismInterface` object, it can always access the current state of the protocol by accessing this property.

  **Return type** `MechanismState`

**Methods Documentation**

**asdict** ()

Converts the object to a dict containing all fields

**discrete_outcomes** *(n_max=None, astype=<class 'dict'>)*

A discrete set of outcomes that spans the outcome space

Parameters

- `n_max` *(int)* – The maximum number of outcomes to return. If None, all outcomes will be returned for discrete issues
- `astype` *(Type[_Forwardref])* – A type to cast the resulting outcomes to.

**Returns** List of n or less outcomes

  **Return type** `List[Outcome]`

**keys** ()

**outcome_index** *(outcome)*

The index of an outcome

Parameters `outcome` *(Outcome)* – The outcome asked about

**Returns** The index of this outcome in the list of outcomes. Only valid if n_outcomes is finite and not None.

  **Return type** `int`

**random_outcomes** *(n=1, astype=<class 'dict'>)*

A set of random outcomes from the issues of this negotiation

Parameters

- `n` *(int)* – number of outcomes requested
- `astype` *(Type[_Forwardref])* – A type to cast the resulting outcomes to.

**Returns** List of n or less outcomes

  **Return type** `List[Outcome]`

**values** ()

**MechanismState**

`class negmas.common.MechanismState` *(running=False, started=False, step=0, time=0.0, relative_time=0.0, broken=False, timeout=False, agreement=None, n_negotiators=0, has_error=False, error_details=“”)*

Bases: `object`

Encapsulates the mechanism state at any point
Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agreement</td>
<td>Agreement at the end of the negotiation (it is always None until an agreement is reached)</td>
</tr>
<tr>
<td>broken</td>
<td>True if the negotiation has started and ended with an END_NEGOTIATION</td>
</tr>
<tr>
<td>ended</td>
<td>True if the negotiation has started and ended with an END_NEGOTIATION</td>
</tr>
<tr>
<td>error_details</td>
<td>Details of the error if any</td>
</tr>
<tr>
<td>has_error</td>
<td>Does the mechanism have any errors</td>
</tr>
<tr>
<td>n_negotiators</td>
<td>Number of agents currently in the negotiation.</td>
</tr>
<tr>
<td>relative_time</td>
<td>A number in the period [0, 1] giving the relative time of the negotiation.</td>
</tr>
<tr>
<td>running</td>
<td>Whether the negotiation has started and did not yet finish</td>
</tr>
<tr>
<td>started</td>
<td>Whether the negotiation has started</td>
</tr>
<tr>
<td>step</td>
<td>The current round of the negotiation</td>
</tr>
<tr>
<td>time</td>
<td>The current real time of the negotiation.</td>
</tr>
<tr>
<td>timedout</td>
<td>True if the negotiation was timed out</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asdict()</td>
<td>Converts the outcome to a dict containing all fields</td>
</tr>
<tr>
<td>keys()</td>
<td></td>
</tr>
<tr>
<td>values()</td>
<td></td>
</tr>
</tbody>
</table>

Attributes Documentation

```python
agreement = None
    Agreement at the end of the negotiation (it is always None until an agreement is reached)

broken = False
    True if the negotiation has started and ended with an END_NEGOTIATION

ended

error_details = ''
    Details of the error if any

has_error = False
    Does the mechanism have any errors

n_negotiators = 0
    Number of agents currently in the negotiation. Notice that this may change over time if the mechanism supports dynamic entry

relative_time = 0.0
    A number in the period [0, 1] giving the relative time of the negotiation. Relative time is calculated as max(step/n_steps, time/time_limit).

running = False
    Whether the negotiation has started and did not yet finish

started = False
    Whether the negotiation has started

step = 0
    The current round of the negotiation

time = 0.0
    The current real time of the negotiation.
```

5.4. negmas.common Module
timedout = False
    True if the negotiation was timedout

Methods Documentation

asdict()
    Converts the outcome to a dict containing all fields

Keys

values()

NegotiatorInfo

class negmas.common.NegotiatorInfo(name, id, type)
    Bases: object

_ShadowAgentMechanismInterface

class negmas.common._ShadowAgentMechanismInterface(ami)
    Bases: object
    Used to represent an AMI to Java.

Methods Summary

discreteOutcomes(nMax)
    rtype int

getNNegotiators()

getOutcomes()
    rtype List[NegotiatorInfo]

getParticipants()
    rtype List[NegotiatorInfo]

getRequirements()
    rtype Dict[str, Any]

getState()
    rtype MechanismState

outcomeIndex(outcome)
    rtype int

randomOutcomes(n)

to_java()

Methods Documentation

discreteOutcomes(nMax)

getNNegotiators()
    Return type int

getOutcomes()

getParticipants()
5.4.3 Class Inheritance Diagram

_SharAgentMechanismInterface

NegotiatorInfo

NamedObject

MechanismState

AgentMechanismInterface

5.5 negmas.negotiators Module

This module defines the interfaces to all negotiation agents (negotiators) in the platform.
5.5.1 Classes

**Negotiator**

```
(name, ufun, parent)
```

Abstract negotiation agent. Base class for all negotiators.

**AspirationMixin**

```
Adds aspiration level calculation.
```

**Controller**

```
(default_negotiator_type, . . . )
```

Controls the behavior of multiple negotiators in multiple negotiations.

**PassThroughNegotiator**

```
(name, ufun, parent)
```

A negotiator that can be used to pass all method calls to a parent (Controller).

---

### Negotiator

**class** negmas.negotiators.Negotiator

```
(name=None, ufun=None, parent=None)
```

Bases: negmas.common.NamedObject, negmas.events.Notifiable, abc.ABC

Abstract negotiation agent. Base class for all negotiators

** Parameters **

- **name** *(Optional[str]*) – Negotiator name. If not given it is assigned by the system (unique 16 characters).

** Returns:**

- **bool**: True if participating in the given negotiation (or any negotiation if it was None)

** Remarks:**

---

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarly for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

---

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_capabilities(capabilities)</td>
<td>Adds named capabilities to the agent.</td>
</tr>
<tr>
<td>before_death(cntxt)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td>join(ami, state, <em>[ufun, role]</em>)</td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td>on_leave(state)</td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td>on_mechanism_error(state)</td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td>on_negotiation_end(state)</td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td>on_negotiation_start(state)</td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td>on_notification(notification, notifier)</td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td>on_round_end(notification, notifier)</td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td>on_round_start(state)</td>
<td>A call back called at each negotiation round start</td>
</tr>
</tbody>
</table>

Continued on next page
Table 54 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_ufun_changed()</td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

**capabilities**  
Agent capabilities  

**Return type**  
Dict[str,Any]

**id**  
The unique ID of this entity

**name**  
A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**  
Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**  
The unique ID of this entity

### Methods Documentation

**add_capabilities(capabilities)**  
Adds named capabilities to the agent.

**Parameters**  
capabilities (dict) – The capabilities to be added as a dict

**Return type**  
None

**Remarks:**  
It is the responsibility of the caller to be really capable of added capabilities.

**before_death(cntxt)**  
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

**Return type**  
bool

**compare(first, second)**  
Compares two offers using the ufun

**Parameters**

- first (Outcome) – First outcome to be compared
- second (Outcome) – Second outcome to be compared

**Returns**  
An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type**  
UtilityValue

**classmethod create(*args, **kwargs)**  
Creates an object and returns a proxy to it.

**isin(negotiation_id)**  
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters**  
negotiation_id (Optional[str]) – The negotiation ID tested. If None, it means ANY negotiation
Returns

True if participating in the given negotiation (or any negotiation if it was None)

Return type  bool

join (ami, state, *, ufun=None, role='agent')
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters

• ami (AgentMechanismInterface) – The negotiation.
• state (MechanismState) – The current state of the negotiation
• ufun (UtilityFunction) – The ufun function to use before any discounting.
• role (str) – role of the agent.

Return type  bool

Returns  bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

on_leave (state)
A call back called after leaving a negotiation.

Parameters  state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

• MUST call the baseclass on_leave using super () if you are going to override this.
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_mechanism_error (state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state

Parameters  state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_negotiation_end (state)
A call back called at each negotiation end

Parameters  state (MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

Remarks:

• The default behavior is to do nothing.
• Override this to hook some action

Return type  None
on_negotiation_start\(\text{(state)}\)
A call back called at each negotiation start

**Parameters**
\(\text{state} \ (\text{MechanismState}) – \text{MechanismState}\) giving current state of the negotiation.

**Remarks:**
- You MUST call the super() version of this function either before or after your code when you are overriding it.

**Return type** None

on_notification\(\text{(notification, notifier)}\)
Called whenever the agent receives a notification

**Parameters**
- \(\text{notification} \ (\text{Notification}) – \text{The notification}!!\)
- \(\text{notifier} \ (\text{str}) – \text{The notifier}!!\)

**Returns** None

**Remarks:**
- You MUST call the super() version of this function either before or after your code when you are overriding it.

on_round_end\(\text{(state)}\)
A call back called at each negotiation round end

**Parameters**
\(\text{state} \ (\text{MechanismState}) – \text{MechanismState}\) giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

on_round_start\(\text{(state)}\)
A call back called at each negotiation round start

**Parameters**
\(\text{state} \ (\text{MechanismState}) – \text{MechanismState}\) giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

on_ufun_changed()
Called to inform the agent that its ufun has changed.

**Remarks:**
- You MUST call the super() version of this function either before or after your code when you are overriding it.
AspirationMixin

```python
class negmas.negotiators.AspirationMixin
    Bases: object

    Adds aspiration level calculation. This Mixin MUST be used with a Negotiator class.
```

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aspiration(t)</code></td>
<td>The aspiration level</td>
</tr>
<tr>
<td><code>aspiration_init</code></td>
<td>The aspiration level</td>
</tr>
</tbody>
</table>

#### Methods Documentation

**aspiration(t)**

The aspiration level

**Parameters**

- `t` *(float)* – relative time (a number between zero and one)

**Return type**

float

**Returns**

aspiration level

**aspiration_init** *(max_aspiration, aspiration_type, above_reserved_value=True)*

**Parameters**

- `max_aspiration` *(float)* –
- `aspiration_type` *(Union[str, int, float])* –
- `above_reserved_value` –

Controller

```python
class negmas.negotiators.Controller (default_negotiator_type=None, default_negotiator_params=None, name=None)
    Bases: negmas.common.NamedObject

    Controls the behavior of multiple negotiators in multiple negotiations

    The controller class MUST implement any methods of the negotiator class it is controlling with one added argument negotiator_id (str) which represents ID of the negotiator on which the method is being invoked (passed first).

    Controllers for specific classes should inherit from this class and implement whatever methods they want to override on their PassThroughNegotiator objects. For example, the SAO module defines SAOController that needs only to implement propose and respond.

    Remarks:

    - Controllers should always call negotiator methods using the call method defined in this class. Direct calls may lead to infinite loops

```

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>id</code></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Continued on next page
Table 56 – continued from previous page

<table>
<thead>
<tr>
<th>name</th>
<th>A convenient name of the entity (intended primarily for printing/logging/debugging).</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(negotiator, method, *args, **kwargs)</td>
</tr>
<tr>
<td>Calls the given method on the given negotiator safely without causing recursion.</td>
</tr>
<tr>
<td><strong>create</strong>(args, **kwargs)</td>
</tr>
<tr>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><strong>create_negotiator</strong>(negotiator_type, name, cntxt)</td>
</tr>
<tr>
<td>Creates a negotiator passing it the context</td>
</tr>
<tr>
<td><strong>join</strong>(negotiator_id, ami, state, *, ufun, role)</td>
</tr>
<tr>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td><strong>kill_negotiator</strong>(negotiator_id[, force])</td>
</tr>
<tr>
<td>Kills the negotiator sending it an before_death message.</td>
</tr>
<tr>
<td><strong>on_leave</strong>(negotiator_id, state)</td>
</tr>
<tr>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td><strong>on_mechanism_error</strong>(negotiator_id, state)</td>
</tr>
<tr>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td><strong>on_negotiation_end</strong>(negotiator_id, state)</td>
</tr>
<tr>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td><strong>on_negotiation_start</strong>(negotiator_id, state)</td>
</tr>
<tr>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td><strong>on_notification</strong>(negotiator_id, notification, ...)</td>
</tr>
<tr>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td><strong>on_round_end</strong>(negotiator_id, state)</td>
</tr>
<tr>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td><strong>on_round_start</strong>(negotiator_id, state)</td>
</tr>
<tr>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td><strong>on_ufun_changed</strong>(negotiator_id)</td>
</tr>
<tr>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong></td>
</tr>
<tr>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td><strong>name</strong></td>
</tr>
<tr>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td><strong>uuid</strong></td>
</tr>
<tr>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Documentation

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(negotiator, method, *args, **kwargs)</td>
</tr>
<tr>
<td>Calls the given method on the given negotiator safely without causing recursion. The controller MUST use this function to access any callable on the negotiator</td>
</tr>
</tbody>
</table>

#### Parameters

- **negotiator** (*PassThroughNegotiator* )
- **method** (*str*)
- ***args**
- ****kwargs**

#### Returns:

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>classmethod create</strong>(args, **kwargs)</td>
</tr>
<tr>
<td>Creates an object and returns a proxy to it.</td>
</tr>
</tbody>
</table>
create_negotiator(negotiator_type=None, name=None, cntxt=None, **kwargs)

Creates a negotiator passing it the context

Parameters

- **negotiator_type** (Union[types[str, Type[PassThroughNegotiator]], None]) – Type of the negotiator to be created
- **name** (Optional[str]) – negotiator name
- **cntxt** (Optional[Dict[str, None]]) – The context to be associated with this negotiator. It will not be passed to the negotiator
- **constructor** –
- ****kwargs – any key-value pairs to be passed to the negotiator constructor

Returns The negotiator to be controlled

Return type PassThroughNegotiator

join(negotiator_id, ami, state, *, ufun=None, role='agent')

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters

- **negotiator_id** (str) – The negotiator ID
- **ami** (AgentMechanismInterface) – The negotiation.
- **state** (MechanismState) – The current state of the negotiation
- **ufun** (UtilityFunction) – The ufun function to use before any discounting.
- **role** (str) – role of the agent.

Return type bool

Returns bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation.

kill_negotiator(negotiator_id, force=False)

Kills the negotiator sending it an before_death message.

Parameters

- **negotiator_id** (str) – The ID of the negotiator to kill.
- **force** (bool) – Whether to kill the negotiator in case it refused to die.

Remarks:

- Killing a negotiator amounts to nothing more than removing it from the list of negotiators maintained by the controller.

Return type None

on_leave(negotiator_id, state)

A call back called after leaving a negotiation.

Parameters

- **negotiator_id** (str) – The negotiator ID
- **state** (MechanismState) – MechanismState giving current state of the negotiation.

Return type None
on_mechanism_error (negotiator_id, state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state.

Parameters

- negotiator_id (str) – The negotiator ID
- state (MechanismState) – MechanismState giving current state of the negotiation.

Return type None

on_negotiation_end (negotiator_id, state)
A call back called at each negotiation end

Parameters

- negotiator_id (str) – The negotiator ID
- state (MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

Return type None

on_negotiation_start (negotiator_id, state)
A call back called at each negotiation start

Parameters

- negotiator_id (str) – The negotiator ID
- state (MechanismState) – MechanismState giving current state of the negotiation.

Return type None

on_notification (negotiator_id, notification, notifier)

on_round_end (negotiator_id, state)
A call back called at each negotiation round end

Parameters

- negotiator_id (str) – The negotiator ID
- state (MechanismState) – MechanismState giving current state of the negotiation.

Return type None

on_round_start (negotiator_id, state)
A call back called at each negotiation round start

Parameters

- negotiator_id (str) – The negotiator ID
- state (MechanismState) – MechanismState giving current state of the negotiation.

Return type None

on_ufun_changed (negotiator_id)
Called to inform the agent that its ufun has changed.

Parameters negotiator_id (str) – The negotiator ID

Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.
PassThroughNegotiator

class negmas.negotiators.PassThroughNegotiator(name=None, ufun=None, parent=None)

Bases: negmas.negotiators.Negotiator

A negotiator that can be used to pass all method calls to a parent (Controller).

It uses magic dunder methods to implement a general way of passing calls to the parent. This method is slow.

It is recommended to implement a PassThroughNegotiator for each mechanism that does this passing explicitly which will be much faster.

For an example, see the implementation of PassThroughSAONegotiator.

Attributes Summary

capabilities Agent capabilities
id The unique ID of this entity
name A convenient name of the entity (intended primarily for printing/logging/debugging).
reserved_value Reserved value is what the agent gets if no agreement is reached in the negotiation.
utility_function
uuid The unique ID of this entity

Methods Summary

add_capabilities(capabilities) Adds named capabilities to the agent.
before_death(cntxt) Called whenever the parent is about to kill this negotiator.
compare(first, second) Compares two offers using the ufun
create(*args, **kwargs) Creates an object and returns a proxy to it.
isin(negotiation_id) Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.
join(ami, state, *, ufun, role) Called by the mechanism when the agent is about to enter a negotiation.
on_leave(state) A call back called after leaving a negotiation.
on_mechanism_error(state) A call back called whenever an error happens in the mechanism.
on_negotiation_end(state) A call back called at each negotiation end
on_negotiation_start(state) A call back called at each negotiation start
on_notification(notification, notifier) Called whenever the agent receives a notification
on_round_end(state) A call back called at each negotiation round end
on_round_start(state) A call back called at each negotiation round start
on_ufun_changed() Called to inform the agent that its ufun has changed.

Attributes Documentation

capabilities Agent capabilities

   Return type Dict[str,Any]

id
The unique ID of this entity

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**
Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**
The unique ID of this entity

### Methods Documentation

**add_capabilities** *(capabilities)*
Adds named capabilities to the agent.

**Parameters**
- **capabilities** *(dict)* – The capabilities to be added as a dict

**Return type**
None

**Returns**
None

**Remarks:** It is the responsibility of the caller to be really capable of added capabilities.

**before_death** *(cntxt)*
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

**Return type**
bool

**compare** *(first, second)*
Compares two offers using the `ufun`

**Parameters**
- **first** *(Outcome)* – First outcome to be compared
- **second** *(Outcome)* – Second outcome to be compared

**Returns** An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type**
UtilityValue

**classmethod create** *(\*args, **kwargs)*
Creates an object and returns a proxy to it.

**isin** *(negotiation_id)*
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters**
- **negotiation_id** *(Optional[str])* – The negotiation ID tested. If None, it means ANY negotiation

**Returns**
True if participating in the given negotiation (or any negotiation if it was None)

**Return type**
bool

**join** *(ami, state, *, ufun=None, role='agent')*
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**
- **ami** *(AgentMechanismInterface)* – The negotiation.
• **state** (*MechanismState*) – The current state of the negotiation
• **ufun** (*UtilityFunction*) – The ufun function to use before any discounting.
• **role** (*str*) – role of the agent.

**Return type** `bool`

**Returns** `bool` indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

**on_leave** (*state*)
A call back called after leaving a negotiation.

**Parameters**
- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks:**
- MUST call the baseclass `on_leave` using `super()` if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** `None`

**on_mechanism_error** (*state*)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in `state`

**Parameters**
- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** `None`

**on_negotiation_end** (*state*)
A call back called at each negotiation end

**Parameters**
- **state** (*MechanismState*) – MechanismState or one of its descendants giving the state at which the negotiation ended.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** `None`

**on_negotiation_start** (*state*)
A call back called at each negotiation start

**Parameters**
- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks:**
- You MUST call the super() version of this function either before or after your code when you are overriding it.
Return type None

on_notification(notification, notifier)
Called whenever the agent receives a notification

Parameters

• notification(Notification) – The notification!!

• notifier(str) – The notifier!!

Returns None

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

on_round_end(state)
A call back called at each negotiation round end

Parameters state(MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.

• Override this to hook some action

Return type None

on_round_start(state)
A call back called at each negotiation round start

Parameters state(MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.

• Override this to hook some action.

Return type None

on_ufun_changed()
Called to inform the agent that its ufun has changed.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.
5.5.2 Class Inheritance Diagram

AspirationMixin

ABC

Notifiable → Negotiator → PassThroughNegotiator

NamedObject → Controller

5.6 negmas.sao Module

Implements Stacked Alternating Offers (SAO) mechanism and basic negotiators.

5.6.1 Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAOState([running, started, step, time, ...])</td>
<td>An alias for SAOMechanism object</td>
</tr>
<tr>
<td>SACMechanism([issues, outcomes, n_steps, ...])</td>
<td>A negotiation agent that responds randomly in a single negotiation.</td>
</tr>
<tr>
<td>SAOProtocol</td>
<td>A negotiation agent that uses a fixed set of outcomes in a single negotiation.</td>
</tr>
<tr>
<td>LimitedOutcomesNegotiator([name, parent, ...])</td>
<td>A negotiation agent that uses a fixed set of outcomes in a single negotiation.</td>
</tr>
<tr>
<td>LimitedOutcomesAcceptor([name, parent, ...])</td>
<td>A negotiation agent that uses a fixed set of outcomes in a single negotiation.</td>
</tr>
<tr>
<td>AspirationNegotiator([name, ufun, parent, ...])</td>
<td>Implements a generalized tit-for-tat strategy</td>
</tr>
<tr>
<td>ToughNegotiator([name, parent, ...])</td>
<td>A simple tit-for-tat negotiator</td>
</tr>
<tr>
<td>OnlyBestNegotiator([name, parent, ...])</td>
<td>A controller that can manage multiple negotiators taking full or partial control from them.</td>
</tr>
<tr>
<td>NaiveTitForTatNegotiator([name, parent, ...])</td>
<td>A controller that can manage multiple negotiators taking full or partial control from them.</td>
</tr>
<tr>
<td>SimpleTitForTatNegotiator</td>
<td></td>
</tr>
<tr>
<td>NiceNegotiator(*args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>SAOController([default_negotiator_type, ...])</td>
<td></td>
</tr>
</tbody>
</table>
JavaSAONegotiator(java_object, java_class_name)

PassThroughSAONegotiator(...)

A negotiator that acts as an end point to a parent Controller

SAOState

class negmas.sao.SAOState (running=False, started=False, step=0, time=0.0, relative_time=0.0, broken=False, timedout=False, agreement=None, n_negotiators=0, has_error=False, error_details='', current_offer=None, current_proposer=None, n_acceptances=0)

Bases: negmas.common.MechanismState

Attributes Summary

agreement
broken
current_offer
current_proposer
ended
error_details
has_error
n_acceptances
n_negotiators
relative_time
running
started
step
time
timedout

Methods Summary

asdict() Converts the outcome to a dict containing all fields
keys()
values()

Attributes Documentation

agreement = None
broken = False
current_offer = None
current_proposer = None
ended
error_details = ''
has_error = False
n_acceptances = 0
n_negotiators = 0
relative_time = 0.0

5.6. negmas.sao Module
```python
running = False
started = False
step = 0
time = 0.0
timedout = False
```

**Methods Documentation**

```python
class NegMAS:
    def __init__(self, issues=None, outcomes=None, n_steps=None, time_limit=None, step_time_limit=None, max_n_agents=None, dynamic_entry=True, keep_issue_names=True, cache_outcomes=True, max_n_outcomes=1000000, annotation=None, end_on_no_response=True, publish_proposer=True, publish_n_acceptances=False, enable_callbacks=False, avoid_ultimatum=True, check_offers=True, name=None):
```

**Attributes Summary**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Returns a list of all participant names</td>
</tr>
<tr>
<td>dynamic_entry</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>history</td>
<td>Returns relative time or steps.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity.</td>
</tr>
<tr>
<td>issues</td>
<td>Returns the remaining number of steps until the end of the mechanism run.</td>
</tr>
<tr>
<td>name</td>
<td>A dictionary specifying the requirements that must be in the capabilities of any agent to join the mechanism.</td>
</tr>
<tr>
<td>negotiators</td>
<td>Returns remaining time in seconds.</td>
</tr>
<tr>
<td>outcomes</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>participants</td>
<td>Returns a dictionary specifying the requirements that must be in the capabilities of any agent to join the mechanism.</td>
</tr>
<tr>
<td>remaining_time</td>
<td>Returns remaining time in seconds.</td>
</tr>
<tr>
<td>requirements</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>running</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>time</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>step_time_limit</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>step</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>timedout</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>step_time_limit</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>step</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td>timedout</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
</tbody>
</table>
```
Table 63 – continued from previous page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Returns the current state.</td>
</tr>
<tr>
<td>time</td>
<td>Elapsed time since mechanism started in seconds.</td>
</tr>
<tr>
<td>time_limit</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

- **add(negotiator, *, uf, role)**: Add an agent to the negotiation.
- **add_requirements(requirements)**: Adds requirements.
- **announce(event)**: Raises an event and informs all event sinks that are registered for notifications on this event type.
- **can_accept_more_agents()**: Whether the mechanism can currently accept more negotiators.
- **can_enter(agent)**: Whether the agent can enter the negotiation now.
- **can_leave(agent)**: Can the agent leave now?
- **can_participate(agent)**: Checks if the agent can participate in this type of negotiation in general.
- **create(*args, **kwargs)**: Creates an object and returns a proxy to it.
- **discrete_outcomes([n_max, astype])**: A discrete set of outcomes that spans the outcome space.
- **extra_state()**: Returns any extra state information to be kept in the state and history properties.
- **get_info(id)**: Returns the mechanism information which contains its static config plus methods to access current state.
- **is_satisfying(capabilities)**: Checks if the given capabilities are satisfying mechanism requirements.
- **is_valid(outcome)**: Checks whether the outcome is valid given the issues.
- **join(ami, state, *, uf, role)**: Determines if the agent can join the negotiation.
- **on_mechanism_error()**: Called when there is a mechanism error.
- **on_negotiation_end()**: Called at the end of each negotiation.
- **on_negotiation_start()**: Called before starting the negotiation.
- **pareto_frontier([n_max, sort_by_welfare])**: Computes the Pareto front.
- **plot([plot_utils, plot_outcomes])**: Plots the negotiation.
- **random_outcomes([n, astype])**: Returns random offers.
- **register_listener(event_type, listener)**: Registers an event listener.
- **remove(agent, **kwargs)**: Remove the agent from the negotiation.
- **remove_requirements(requirements)**: Adds requirements.
- **round()**: Implements a round of the Stacked Alternating Offers Protocol.
run([timeout])

**rtype** MechanismState

<table>
<thead>
<tr>
<th>Method</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>run()</td>
<td>Runs a single step of the mechanism.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- **agreement**
  - all = {}

- **completed**

- **dynamic_entry**

- **history**

- **id**
  - The unique ID of this entity

- **issues**

- **max_n_agents**

- **n_outcomes**

- **n_steps**

- **name**
  - A convenient name of the entity (intended primarily for printing/logging/debugging).

- **negotiators**

- **outcomes**

- **participants**
  - Returns a list of all participant names

  **Return type** List[NegotiatorInfo]

- **relative_time**
  - Returns a number between 0 and 1 indicating elapsed relative time or steps.

  **Return type** Optional[float]

- **remaining_steps**
  - Returns the remaining number of steps until the end of the mechanism run. None if unlimited

  **Return type** Optional[int]

- **remaining_time**
  - Returns remaining time in seconds. None if no time limit is given.

  **Return type** Optional[float]

- **requirements**
  - A dictionary specifying the requirements that must be in the capabilities of any agent to join the mechanism.

- **running**

- **state**
  - Returns the current state. Override `extra_state` if you want to keep extra state

- **time**
  - Elapsed time since mechanism started in seconds. None if the mechanism did not start running

  **Return type** Optional[float]

- **time_limit**
uuid
The unique ID of this entity

Methods Documentation

**add** *(negotiator, *, ufun=None, role=None, **kwargs)*
Add an agent to the negotiation.

**Parameters**

- **negotiator** *(Negotiator)* – The agent to be added.
- **ufun** *(Optional[UtilityFunction])* – The utility function to use. If None, then the agent must already have a stored utility function otherwise it will fail to enter the negotiation.
- **role** *(Optional[str])* – The role the agent plays in the negotiation mechanism. It is expected that mechanisms inheriting from this class will check this parameter to ensure that the role is a valid role and is still possible for negotiators to join on that role. Roles may include things like moderator, representative etc based on the mechanism.

**Return type** *Optional[bool]*

**Returns**

- True if the agent was added.
- False if the agent was already in the negotiation.
- None if the agent cannot be added.

**add_requirements**(requirements)
Add requirements.

**Return type** *None*

**announce**(event)
Raises an event and informs all event sinks that are registerd for notifications on this event type.

**can_accept_more_agents** ()
Whether the mechanism can currently accept more negotiators.

**Return type** *bool*

**can_enter**(agent)
Whether the agent can enter the negotiation now.

**Return type** *bool*

**can_leave**(agent)
Can the agent leave now?

**Return type** *bool*

**can_participate**(agent)
Checks if the agent can participate in this type of negotiation in general.

**Parameters**

- **agent** *(Negotiator)* –

**Returns**

- True if it can

**Return type** *bool*

**Remarks:** The only reason this may return False is if the mechanism requires some requirements that are not within the capabilities of the agent.

5.6. negmas.sao Module
When evaluating compatibility, the agent is considered incapable of participation if any of the following conditions hold: * A mechanism requirement is not in the capabilities of the agent * A mechanism requirement is in the capabilities of the agent by the values required for it is not in the values announced by the agent.

An agent that lists a None value for a capability is announcing that it can work with all its values. On the other hand, a mechanism that lists a requirement as None announces that it accepts any value for this requirement as long as it exist in the agent

```
classmethod create(*args, **kwargs)
    Creates an object and returns a proxy to it.

discrete_outcomes(n_max=None, astype=<class 'dict'>)
    A discrete set of outcomes that spans the outcome space

    Parameters
    * n_max (Optional[int]) – The maximum number of outcomes to return. If None, all outcomes will be returned for discrete issues
    * 100 if any of the issues was continuous (and) –
    * astype (Type[Union[OutcomeType, Tuple[Union[int, float, str], list]], Dict[Union[int, str], Union[int, float, str, list]]]) – A type to cast the resulting outcomes to.

    Returns List of n or less outcomes
    Return type List[Outcome]

extra_state()
    Returns any extra state information to be kept in the state and history properties

classmethod get_info(id)
    Returns the mechanism information which contains its static config plus methods to access current state

    Return type AgentMechanismInterface

is_satisfying(capabilities)
    Checks if the given capabilities are satisfying mechanism requirements.

    Parameters capabilities (dict) – capabilities to check

    Return type bool

    Returns bool are the requirements satisfied by the capabilities.

    Remarks:
    * Requirements are also a dict with the following meanings:
      – tuple: Min and max acceptable values
      – list/set: Any value in the iterable is acceptable
      – Single value: The capability must match this value
    * Capabilities can also have the same three possibilities.

is_valid(outcome)
    Checks whether the outcome is valid given the issues

join(ami, state, *, ufun=None, role='agent')

    Return type bool

on_mechanism_error()
    Called when there is a mechanism error

    Remarks:
• When overriding this function you **MUST** call the base class version

**Return type** None

**on_negotiation_end()**
Called at the end of each negotiation

**Remarks:**
• When overriding this function you **MUST** call the base class version

**Return type** None

**on_negotiation_start()**
Called before starting the negotiation. If it returns False then negotiation will end immediately

**Return type** bool

**pareto_frontier**(n_max=None, sort_by_welfare=True)

**Return type** Tuple[List[Tuple[float]], List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]]

**plot**(plot_utils=True, plot_outcomes=True)

**random_outcomes**(n=1, astype=<class 'dict'>)
Returns random offers

**Return type** List[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]]

**register_listener**(event_type, listener)

**remove**(agent, **kwargs)
Remove the agent from the negotiation.

**Parameters**
agent (**Negotiator**) –

**Return type** Optional[bool]

**Returns**
• True if the agent was removed.
• False if the agent was not in the negotiation already.
• None if the agent cannot be removed.

**remove_requirements**(requirements)
Adds requirements.

**Return type** None

**round()**
implements a round of the Stacked Alternating Offers Protocol.

**Return type** MechanismRoundResult

**run**(timeout=None)

**Return type** MechanismState

**step()**
Runs a single step of the mechanism.

**Returns**
The state of the negotiation after the round is conducted

**Return type** MechanismState

Remarks:
• Every call yields the results of one round (see `round()`)  
• If the mechanism was yet to start, it will start it and runs one round  
• There is another function (`run()`) that runs the whole mechanism in blocking mode

**SAOProtocol**

```python
def round():  # Method signature  
    # Implementation details
```

An alias for `SAOMechanism` object

```python
def run():  # Method signature  
    # Implementation details
```

**SAONegotiator**

class SAONegotiator(assume_normalized=True, ufun=None, name=None, rational_proposal=True, parent=None):

    Bases: negmas.negotiators.Negotiator

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>eu</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_capabilities(capabilities)</td>
<td>Adds named capabilities to the agent.</td>
</tr>
<tr>
<td>before_death(ctx)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>counter(state, offer)</td>
<td>Called to counter an offer</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td>join(ami, state, *[ufun, role])</td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td>on_leave(state)</td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td>on_mechanism_error(state)</td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td>on_negotiation_end(state)</td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td>on_negotiation_start(state)</td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td>on_notification(notification, notifier)</td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td>on_partner_proposal(state, agent_id, offer)</td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
</tbody>
</table>

Continued on next page
Table 66 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_partner_refused_to_propose(state, agent_id)</code></td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td><code>on_partner_response(state, agent_id, ...)</code></td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td><code>on_round_end(state)</code></td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start(state)</code></td>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed()</code></td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>propose(state)</code></td>
<td>Propose a set of offers</td>
</tr>
<tr>
<td><code>propose_(state)</code></td>
<td></td>
</tr>
<tr>
<td><code>rtype</code></td>
<td><code>Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]</code></td>
</tr>
<tr>
<td><code>respond(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
<tr>
<td><code>respond_(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
</tbody>
</table>

Attributes Documentation

`capabilities`  
Agent capabilities

**Return type** `Dict[str, Any]`

`eu`  
The utility function in the given negotiation taking opponent model into account.

**Remarks:**
- If no `utility_function` is internally stored, `eu` still returns a valid callable that returns `None` for everything.

**Return type** `Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]], None]]`

`id`  
The unique ID of this entity

`name`  
A convenient name of the entity (intended primarily for printing/logging/debugging).

`reserved_value`  
Reserved value is what the agent gets if no agreement is reached in the negotiation.

`utility_function`  

`uuid`  
The unique ID of this entity

Methods Documentation

`add_capabilities(capabilities)`  
Adds named capabilities to the agent.

**Parameters**  
- `capabilities (dict)` – The capabilities to be added as a dict

**Return type** `None`
Returns None

Remarks: It is the responsibility of the caller to be really capable of added capabilities.

**before_death** (cntxt)
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

Return type bool

**compare** (first, second)
Compares two offers using the ufun

Parameters

- **first** (Outcome) – First outcome to be compared
- **second** (Outcome) – Second outcome to be compared

Returns An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

Return type UtilityValue

**counter** (state, offer)
Called to counter an offer

Parameters

- **state** (MechanismState) – MechanismState giving current state of the negotiation.
- **offer** (Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list], None]], None]) – The offer to be countered. None means no offer and the agent is requested to propose an offer

Returns The response to the given offer with a counter offer if the response is REJECT

Return type Tuple[ResponseType, Outcome]

**classmethod create** (*args, **kwargs)
Creates an object and returns a proxy to it.

**isin** (negotiation_id)
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

Parameters negotiation_id (Optional[str]) – The negotiation ID tested. If None, it means ANY negotiation

Returns True if participating in the given negotiation (or any negotiation if it was None)

Return type bool

**join** (ami, state, *, ufun=None, role='agent')
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters

- **ami** (AgentMechanismInterface) – The negotiation.
- **state** (MechanismState) – The current state of the negotiation
- **ufun** (UtilityFunction) – The ufun function to use before any discounting.
- **role** (str) – role of the agent.
Return type  
(bool)

Returns  bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

on_leave(state)
A call back called after leaving a negotiation.

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• MUST call the baseclass on_leave using super() if you are going to override this.
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_mechanism_error(state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state.

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_negotiation_end(state)
A call back called at each negotiation end

Parameters state (MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_negotiation_start(state)
A call back called at each negotiation start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

Return type  None

on_notification(notification, notifier)
Called whenever the agent receives a notification
Parameters

- **notification** (*Notification*) – The notification!!
- **notifier** (*str*) – The notifier!!

Returns None

Remarks:

- You MUST call the super() version of this function either before or after your code when you are overriding it.

**on_partner_proposal** (*state, agent_id, offer*)

A callback called by the mechanism when a partner proposes something

Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the offer was proposed.
- **agent_id** (*str*) – The ID of the agent who proposed
- **offer** (*Union*[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.

Return type None

Returns None

**on_partner_refused_to_propose** (*state, agent_id*)

A callback called by the mechanism when a partner refuses to propose

Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner refused to offer.
- **agent_id** (*str*) – The ID of the agent who refused to propose

Return type None

Returns None

**on_partner_response** (*state, agent_id, outcome, response*)

A callback called by the mechanism when a partner responds to some offer

Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner responded.
- **agent_id** (*str*) – The ID of the agent who responded
- **outcome** (*Union*[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal being responded to.
- **response** (*SAOResponse*) – The response

Return type None

Returns None

**on_round_end** (*state*)

A call back called at each negotiation round end

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
• Override this to hook some action

Return type None

\textbf{on\_round\_start} (\textit{state})

A call back called at each negotiation round start

Parameters \textit{state} (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action.

Return type None

\textbf{on\_ufun\_changed}()

Called to inform the agent that its ufun has changed.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

\textbf{propose} (\textit{state})

Propose a set of offers

Parameters \textit{state} (MechanismState) – MechanismState giving current state of the negotiation.

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int,str], Union[int, float, str, list]], None]

Returns The outcome being proposed or None to refuse to propose

Remarks:
• This function guarantees that no agents can propose something with a utility value

\textbf{propose\_} (\textit{state})

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int,str], Union[int, float, str, list]], None]

\textbf{respond} (\textit{state, offer})

Respond to an offer.

Parameters
• \textit{state} (MechanismState) – MechanismState giving current state of the negotiation.
• \textit{offer} (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
• The default implementation never ends the negotiation
• The default implementation asks the negotiator to propose\_() and accepts the `offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).
respond_ (state, offer)
Respond to an offer.

Parameters

• **state** (MechanismState) – MechanismState giving current state of the negotiation.

• **offer** (Outcome) – offer being tested

Returns The response to the offer

Return type **ResponseType**

Remarks:

• The default implementation never ends the negotiation except if an earlier end_negotiation notification is sent to the negotiator

• The default implementation asks the negotiator to propose`()` and accepts the `offer` if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

**RandomNegotiator**

class negmas.sao.RandomNegotiator (outcomes, name=None, parent=None, reserved_value=None, p_acceptance=0.15, p_rejection=0.25, p_ending=0.1, can_propose=True, ufum=None)


A negotiation agent that responds randomly in a single negotiation.

**Attributes Summary**

<table>
<thead>
<tr>
<th>capabilities</th>
<th>Agent capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>add_capabilities(capabilities)</th>
<th>Adds named capabilities to the agent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>before_death(cntxt)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufum</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>init_random_proposal()</td>
<td>Constructor</td>
</tr>
<tr>
<td>init_random_response([p_acceptance, ...])</td>
<td></td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
</tbody>
</table>
Table 68 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>join(ami, state, *[, ufun, role])</code></td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td><code>on_leave(state)</code></td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td><code>on_mechanism_error(state)</code></td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td><code>on_negotiation_end(state)</code></td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td><code>on_negotiation_start(state)</code></td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td><code>on_notification(notification, notifier)</code></td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td><code>on_round_end(state)</code></td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start(state)</code></td>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed()</code></td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>propose(state)</code></td>
<td>rtype Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]</td>
</tr>
<tr>
<td><code>respond(state, offer)</code></td>
<td>rtype ResponseType</td>
</tr>
</tbody>
</table>

Attributes Documentation

**capabilities**
Agent capabilities

Return type Dict[str, Any]

**id**
The unique ID of this entity

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**
Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**
The unique ID of this entity

Methods Documentation

**add_capabilities(capabilities)**
Add named capabilities to the agent.

Parameters capabilities (dict) – The capabilities to be added as a dict

Return type None

Returns None

Remarks: It is the responsibility of the caller to be really capable of added capabilities.
**before_death** *(cntxt)*
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it.

**Return type** `bool`

**compare** *(first, second)*
Compares two offers using the `ufun`

**Parameters**
- `first` *(Outcome)* – First outcome to be compared
- `second` *(Outcome)* – Second outcome to be compared

**Returns** An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type** `UtilityValue`

**classmethod create** *(\*args, \*\*kwargs)*
Creates an object and returns a proxy to it.

**init_random_proposal** *(\*)
**init_random_response** *(p_acceptance=0.15, p_rejection=0.25, p_ending=0.1)*

**Constructor**

**Parameters**
- `p_acceptance` *(float)* – probability of accepting offers
- `p_rejection` *(float)* – probability of rejecting offers
- `p_ending` *(float)* – probability of ending negotiation

**Return type** `None`

**Returns** `None`

**Remarks:**
- If the summation of acceptance, rejection and ending probabilities is less than 1.0 then with the remaining probability a NO_RESPONSE is returned from `respond()`.

**isin** *(negotiation_id)*
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters** `negotiation_id` *(Optional[\str])* – The negotiation ID tested. If None, it means ANY negotiation

**Returns**
- `True if participating in the given negotiation (or any negotiation if it was None)`

**Return type** `bool`

**join** *(ami, state, *, ufun=None, role='agent')*
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**
- `ami` *(AgentMechanismInterface)* – The negotiation.
- `state` *(MechanismState)* – The current state of the negotiation
- `ufun` *(UtilityFunction)* – The `ufun` function to use before any discounting.
- `role` *(\str)* – role of the agent.
Return type  bool

Returns  bool indicating whether or not the agent accepts to enter. If False is returned it will
not enter the negotiation

on_leave(state)
A call back called after leaving a negotiation.

Parameters  state (MechanismState) – MechanismState giving current state of
the negotiation.

Remarks:
• MUST call the baseclass on_leave using super () if you are going to override this.
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_mechanism_error(state)
A call back called whenever an error happens in the mechanism. The error and its explanation are
accessible in state

Parameters  state (MechanismState) – MechanismState giving current state of
the negotiation.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_negotiation_end(state)
A call back called at each negotiation end

Parameters  state (MechanismState) – MechanismState or one of its descendants
giving the state at which the negotiation ended.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action

Return type  None

on_negotiation_start(state)
A call back called at each negotiation start

Parameters  state (MechanismState) – MechanismState giving current state of
the negotiation.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are
overriding it.

Return type  None

on_notification(notification, notifier)
Called whenever the agent receives a notification
Parameters

- **notification** (*Notification*) – The notification!!
- **notifier** (*str*) – The notifier!!

**Returns** None

Remarks:

- You MUST call the super() version of this function either before or after your code when you are overriding it.

**on_round_end** (*state*)
A call back called at each negotiation round end

**Parameters**
- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks**:

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_round_start** (*state*)
A call back called at each negotiation round start

**Parameters**
- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks**:

- The default behavior is to do nothing.
- Override this to hook some action.

**Return type** None

**on_ufun_changed**()
Called to inform the agent that its ufun has changed.

**Remarks**:

- You MUST call the super() version of this function either before or after your code when you are overriding it.

**propose** (*state*)

**Return type**

- `Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]`

**respond** (*state, offer*)

**Return type** `ResponseType`

**LimitedOutcomesNegotiator**

```
class negmas.sao.LimitedOutcomesNegotiator
```
A negotiation agent that uses a fixed set of outcomes in a single negotiation.

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>eu</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_capabilities(capabilities)</td>
<td>Adds named capabilities to the agent.</td>
</tr>
<tr>
<td>before_death(ctx)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>counter(state, offer)</td>
<td>Called to counter an offer</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>init_limited_outcomes([outcomes,...])</td>
<td>Constructor</td>
</tr>
<tr>
<td>init_limited_outcomes_acceptor([outcomes,...])</td>
<td>Constructor</td>
</tr>
<tr>
<td>init_limited_outcomes_proposer([...])</td>
<td>Constructor</td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td>join(ami, state, *[, ufun, role])</td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td>on_leave(state)</td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td>on_mechanism_error(state)</td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td>on_negotiation_end(state)</td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td>on_negotiation_start(state)</td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td>on_notification(notification, notifier)</td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td>on_partner_proposal(state, agent_id, offer)</td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td>on_partner_refused_to_propose(state, agent_id)</td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td>on_partner_response(state, agent_id, ...)</td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td>on_round_end(state)</td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td>on_round_start(state)</td>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td>on_ufun_changed()</td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 70 – continued from previous page

propose(state)

| rtype   | Union[OutcomeType, |
|         | Tuple[Union[int, |
|         | float, str, list]], |
|         | Dict[Union[int, str], |
|         | Union[int, float, str, |
|         | list]], None |

propose_(state)

| rtype   | Union[OutcomeType, |
|         | Tuple[Union[int, |
|         | float, str, list]], |
|         | Dict[Union[int, str], |
|         | Union[int, float, str, |
|         | list]], None |

respond(state, offer)

<table>
<thead>
<tr>
<th>Respond to an offer.</th>
</tr>
</thead>
</table>

respond_(state, offer)

<table>
<thead>
<tr>
<th>Respond to an offer.</th>
</tr>
</thead>
</table>

Attributes Documentation

capabilities

Agent capabilities

| Return type | Dict[str, Any] |

eu

The utility function in the given negotiation taking opponent model into account.

Remarks:

- If no utility_function is internally stored, eu still returns a valid callable that returns None for everything.

| Return type | Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Union[Distribution, float, None]] |

id

The unique ID of this entity

name

A convenient name of the entity (intended primarily for printing/logging/debugging).

reserved_value

Reserved value is what the agent gets if no agreement is reached in the negotiation.

utility_function

uuid

The unique ID of this entity

Methods Documentation

add_capabilities(capabilities)

Adds named capabilities to the agent.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>capabilities (dict) – The capabilities to be added as a dict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return type</td>
<td>None</td>
</tr>
<tr>
<td>Returns</td>
<td>None</td>
</tr>
</tbody>
</table>
Remarks: It is the responsibility of the caller to be really capable of added capabilities.

**before_death**(cntxt)
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it.

Return type: `bool`

**compare**(first, second)
Compares two offers using the ufun.

Parameters:

- `first` (**Outcome**) – First outcome to be compared
- `second` (**Outcome**) – Second outcome to be compared

Returns: An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

Return type: `UtilityValue`

**counter**(state, offer)
Called to counter an offer.

Parameters:

- `state` (**MechanismState**) – MechanismState giving current state of the negotiation.
- `offer` (**Union**

  - `Outcome`

  - `Tuple`

  - `Dict`

  - `None`) – The offer to be countered. None means no offer and the agent is requested to propose an offer.

Returns: The response to the given offer with a counter offer if the response is REJECT

Return type: `Tuple[ResponseType, Outcome]`

**classmethod create**(*args, **kwargs)
Creates an object and returns a proxy to it.

**init_limited_outcomes**(outcomes=None, acceptable_outcomes=None, acceptance_probabilities=None, proposable_outcomes=None, p_ending=0.0, p_no_response=0.0)
Constructor

Parameters:

- `acceptable_outcomes` (**Optional[Outcomes]**) – the set of acceptable outcomes. If None then it is assumed to be all the outcomes of the negotiation.
- `acceptance_probabilities` (**Sequence[Float]**) – probability of accepting each acceptable outcome. If None then it is assumed to be unity.
- `proposable_outcomes` (**Optional[Outcomes]**) – the set of outcomes from which the agent is allowed to propose. If None, then it is the same as acceptable outcomes with nonzero probability
- `p_no_response` (**float**) – probability of refusing to respond to offers
- `p_ending` (**float**) – probability of ending negotiation

Return type: `None`

Returns: None
init_limited_outcomes_acceptor(outcomes=None, acceptable_outcomes=None, acceptance_probabilities=None, p_ending=0.05, p_no_response=0.0)

Constructor

Parameters

• acceptable_outcomes (Optional[Floats]) – the set of acceptable outcomes. If None then it is assumed to be all the outcomes of the negotiation.

• acceptance_probabilities (Sequence[int]) – probability of accepting each acceptable outcome. If None then it is assumed to be unity.

• p_no_response (float) – probability of refusing to respond to offers

• p_ending (float) – probability of ending negotiation

Return type None

Returns None

init_limited_outcomes_proposer(proposable_outcomes=None)

Return type None

isin(negotiation_id)

Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

Parameters negotiation_id (Optional[str]) – The negotiation ID tested. If None, it means ANY negotiation

Returns

True if participating in the given negotiation (or any negotiation if it was None)

Return type bool

join(ami, state, *, ufun=None, role='agent')

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters

• ami (AgentMechanismInterface) – The negotiation.

• state (MechanismState) – The current state of the negotiation

• ufun (UtilityFunction) – The ufun function to use before any discounting.

• role (str) – role of the agent.

Return type bool

Returns bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

on_leave(state)

A call back called after leaving a negotiation.

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

• MUST call the baseclass on_leave using super() if you are going to override this.

• The default behavior is to do nothing.

• Override this to hook some action

Return type None
on_mechanism_error(state)
   A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state

   Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

   Remarks:
   • The default behavior is to do nothing.
   • Override this to hook some action

   Return type None

on_negotiation_end(state)
   A call back called at each negotiation end

   Parameters state (MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

   Remarks:
   • The default behavior is to do nothing.
   • Override this to hook some action

   Return type None

on_negotiation_start(state)
   A call back called at each negotiation start

   Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

   Remarks:
   • You MUST call the super() version of this function either before or after your code when you are overriding it.

   Return type None

on_notification(notification, notifier)
   Called whenever the agent receives a notification

   Parameters
   • notification (Notification) – The notification!!
   • notifier (str) – The notifier!!

   Returns None

   Remarks:
   • You MUST call the super() version of this function either before or after your code when you are overriding it.

on_partner_proposal(state, agent_id, offer)
   A callback called by the mechanism when a partner proposes something

   Parameters
   • state (MechanismState) – MechanismState giving the state of the negotiation when the offer was proposed.
   • agent_id (str) – The ID of the agent who proposed
• offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.

Return type None
Returns None

on_partner_refused_to_propose (state, agent_id)
A callback called by the mechanism when a partner refuses to propose

Parameters
• state (MechanismState) – MechanismState giving the state of the negotiation when the partner refused to offer.
• agent_id (str) – The ID of the agent who refused to propose

Return type None
Returns None

on_partner_response (state, agent_id, outcome, response)
A callback called by the mechanism when a partner responds to some offer

Parameters
• state (MechanismState) – MechanismState giving the state of the negotiation when the partner responded.
• agent_id (str) – The ID of the agent who responded
• outcome (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal being responded to.
• response (SAOResponse) – The response

Return type None
Returns None

on_round_end (state)
A call back called at each negotiation round end

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action

Return type None

on_round_start (state)
A call back called at each negotiation round start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action.

Return type None
on_ufun_changed()
Called to inform the agent that its ufun has changed.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are
overriding it.

propose (state)

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]],
                  Dict[Union[int, str], Union[int, float, str, list]], None]

propose_(state)

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]],
                  Dict[Union[int, str], Union[int, float, str, list]], None]

respond (state, offer)
Respond to an offer.

Parameters offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

respond_ (state, offer)
Respond to an offer.

Parameters
• state (MechanismState) – MechanismState giving current state of the ne-
gotiation.
• offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
• The default implementation never ends the negotiation except if an earlier end_negotiation
notification is sent to the negotiator
• The default implementation asks the negotiator to propose `()` and accepts the
`offer` if its utility was at least as good as the offer that it would have proposed (and above
the reserved value).

LimitedOutcomesAcceptor

class negmas.sao.LimitedOutcomesAcceptor (name=None, parent=None, outcomes=None,
                                          acceptable_outcomes=None,
                                          acceptance_probabilities=None, p_ending=0.0,
                                          p_no_response=0.0, ufun=None)

Bases: negmas.sao.SAONegotiator, negmas.sao.LimitedOutcomesAcceptorMixin

A negotiation agent that uses a fixed set of outcomes in a single negotiation.

Attributes Summary

<table>
<thead>
<tr>
<th>capabilities</th>
<th>Agent capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
</tbody>
</table>

Continued on next page
### Methods Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_capabilities</code></td>
<td>(capabilities) Adds named capabilities to the agent.</td>
</tr>
<tr>
<td><code>before_death</code></td>
<td>(ctxt) Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td><code>compare</code></td>
<td>(first, second) Compares two offers using the ufun</td>
</tr>
<tr>
<td><code>counter</code></td>
<td>(state, offer) Called to counter an offer</td>
</tr>
<tr>
<td><code>create</code></td>
<td>(*args, **kwargs) Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>init_limited_outcomes_acceptor</code></td>
<td>([outcomes, ...]) Constructor</td>
</tr>
<tr>
<td><code>isin</code></td>
<td>(negotiation_id) Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td><code>join</code></td>
<td>(ami, state, *[, ufun, role]) Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td><code>on_leave</code></td>
<td>(state) A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td><code>on_mechanism_error</code></td>
<td>(state) A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td><code>on_negotiation_end</code></td>
<td>(state) A call back called at each negotiation end</td>
</tr>
<tr>
<td><code>on_negotiation_start</code></td>
<td>(state) A call back called at each negotiation start</td>
</tr>
<tr>
<td><code>on_notification</code></td>
<td>(notification, notifier) Called whenever the agent receives a notification</td>
</tr>
<tr>
<td><code>on_partner_proposal</code></td>
<td>(state, agent_id, offer) A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td><code>on_partner_refused_to_propose</code></td>
<td>(state, agent_id) A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td><code>on_partner_response</code></td>
<td>(state, agent_id, ...) A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td><code>on_round_end</code></td>
<td>(state) A call back called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start</code></td>
<td>(state) A call back called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed</code></td>
<td>() Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>proposed</code></td>
<td>Propose a set of offers</td>
</tr>
<tr>
<td><code>propose</code></td>
<td>(state)</td>
</tr>
<tr>
<td><code>respond</code></td>
<td>(state, offer) Respond to an offer.</td>
</tr>
<tr>
<td><code>respond_</code></td>
<td>(state, offer) Respond to an offer.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

**Capabilities**

- `capabilities` Agent capabilities
Return type: Dict[str, Any]

eu

The utility function in the given negotiation taking opponent model into account.

Remarks:

• If no utility_function is internally stored, eu still returns a valid callable that returns None
  for everything.

Return type: Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str, Union[int, float, str, list]]]], Union[Distribution, float, None]]

id

The unique ID of this entity

name

A convenient name of the entity (intended primarily for printing/logging/debugging).

reserved_value

Reserved value is what the agent gets if no agreement is reached in the negotiation.

utility_function

uuid

The unique ID of this entity

Methods Documentation

add_capabilities(capabilities)

Adds named capabilities to the agent.

Parameters:

capabilities (dict) – The capabilities to be added as a dict

Return type: None

Returns: None

Remarks: It is the responsibility of the caller to be really capable of added capabilities.

before_death(cntxt)

Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does
not want to be killed but the controller can still force-kill it

Return type: bool

counter(state, offer)

Called to counter an offer

Parameters:

Remarks:
• **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

• **offer** (*Union*[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be countered. None means no offer and the agent is requested to propose an offer.

**Returns** The response to the given offer with a counter offer if the response is REJECT

**Return type** Tuple[*ResponseType*, Outcome]

**classmethod create** (*args*, **kwargs*)

Creates an object and returns a proxy to it.

**init_limited_outcomes_acceptor** (*outcomes=None*, *acceptable_outcomes=None*, *acceptance_probabilities=None*, *p_ending=0.05*, *p_no_response=0.0*)

**Constructor**

**Parameters**

• **acceptable_outcomes** (*Optional*[Floats]) – the set of acceptable outcomes. If None then it is assumed to be all the outcomes of the negotiation.

• **acceptance_probabilities** (*Sequence*[int]) – probability of accepting each acceptable outcome. If None then it is assumed to be unity.

• **p_no_response** (*float*) – probability of refusing to respond to offers

• **p_ending** (*float*) – probability of ending negotiation

**Return type** None

**Returns** None

**isin** (*negotiation_id*)

Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters** *negotiation_id* (*Optional*[str]) – The negotiation ID tested. If None, it means ANY negotiation

**Returns**

True if participating in the given negotiation (or any negotiation if it was None)

**Return type** bool

**join** (*ami*, *state*, *, *ufun=None*, *role='agent'*)

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**

• **ami** (*AgentMechanismInterface*) – The negotiation.

• **state** (*MechanismState*) – The current state of the negotiation

• **ufun** (*UtilityFunction*) – The ufun function to use before any discounting.

• **role** (*str*) – role of the agent.

**Return type** bool

**Returns** bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

**on_leave** (*state*)

A call back called after leaving a negotiation.
Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- MUST call the baseclass **on_leave** using *super()* if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_mechanism_error**(state)

A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in *state*

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_negotiation_end**(state)

A call back called at each negotiation end

Parameters **state** (*MechanismState*) – *MechanismState* or one of its descendants giving the state at which the negotiation ended.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_negotiation_start**(state)

A call back called at each negotiation start

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- You MUST call the supert() version of this function either before or after your code when you are overriding it.

**Return type** None

**on_notification**(notification, notifier)

Called whenever the agent receives a notification

Parameters

- **notification** (*Notification*) – The notification!!
- **notifier** (*str*) – The notifier!!

**Returns** None

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

**on_partner_proposal**(state, agent_id, offer)
A callback called by the mechanism when a partner proposes something

**Parameters**

- **state** *(MechanismState)* – MechanismState giving the state of the negotiation when the offer was proposed.
- **agent_id** *(str)* – The ID of the agent who proposed
- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])* – The proposal.

**Return type** None

**Returns** None

**on_partner_refused_to_propose**(state, agent_id)
A callback called by the mechanism when a partner refuses to propose

**Parameters**

- **state** *(MechanismState)* – MechanismState giving the state of the negotiation when the partner refused to offer.
- **agent_id** *(str)* – The ID of the agent who refused to propose

**Return type** None

**Returns** None

**on_partner_response**(state, agent_id, outcome, response)
A callback called by the mechanism when a partner responds to some offer

**Parameters**

- **state** *(MechanismState)* – MechanismState giving the state of the negotiation when the partner responded.
- **agent_id** *(str)* – The ID of the agent who responded
- **outcome** *(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])* – The proposal being responded to.
- **response** *(SAOResponse)* – The response

**Return type** None

**Returns** None

**on_round_end**(state)
A call back called at each negotiation round end

**Parameters** state *(MechanismState)* – MechanismState giving current state of the negotiation.

**Remarks:**

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_round_start**(state)
A call back called at each negotiation round start
Parameters `state` *(MechanismState)* – MechanismState giving current state of the negotiation.

Remarks:
- The default behavior is to do nothing.
- Override this to hook some action.

**Return type** `None`

`on_ufun_changed()`
Called to inform the agent that its ufun has changed.

Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.

`propose(state)`
Propose a set of offers

Parameters
- `state` *(MechanismState)* – MechanismState giving current state of the negotiation.

Return type `Union [OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]`

Returns The outcome being proposed or None to refuse to propose

Remarks:
- This function guarantees that no agents can propose something with a utility value

`propose_(state)`

Return type `Union [OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]`

`respond(state, offer)`
Respond to an offer.

Parameters
- `state` *(MechanismState)* – MechanismState giving current state of the negotiation.
- `offer` *(Outcome)* – offer being tested

Returns The response to the offer

Return type *ResponseType*

Remarks:
- The default implementation never ends the negotiation
- The default implementation asks the negotiator to propose`() and accepts the `offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).”

`respond_(state, offer)`
Respond to an offer.

Parameters
- `state` *(MechanismState)* – MechanismState giving current state of the negotiation.
• offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
• The default implementation never ends the negotiation except if an earlier end_negotiation
  notification is sent to the negotiator
• The default implementation asks the negotiator to propose`()` and accepts the
  offer if its utility was at least as good as the offer that it would have proposed (and above
  the reserved value).

AspirationNegotiator

class negmas.sao.AspirationNegotiator(name=None, ufun=None, parent=None, max_aspiration=1.0, aspiration_type='boulware', dynamic_ufun=True, randomize_offer=False, can_propose=True, assume_normalized=False, ranking=False)

Bases: negmas.sao.SAONegotiator, negmas.negotiators.AspirationMixin

Attributes Summary

- capabilities Agent capabilities
- eu The utility function in the given negotiation taking opponent model into account.
- id The unique ID of this entity
- name A convenient name of the entity (intended primarily for printing/logging/debugging).
- reserved_value Reserved value is what the agent gets if no agreement is reached in the negotiation.
- utility_function
- uuid The unique ID of this entity

Methods Summary

- add_capabilities(capabilities) Adds named capabilities to the agent.
- aspiration(t) The aspiration level
- aspiration_init(max_aspiration, aspiration_type) type max_aspiration float
- before_death(cntxt) Called whenever the parent is about to kill this negotiator.
- compare(first, second) Compares two offers using the ufun
- counter(state, offer) Called to counter an offer
- create(*args, **kwargs) Creates an object and returns a proxy to it.
- isin(negotiation_id) Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.
- join(ami, state, *[., ufun, role]) Called by the mechanism when the agent is about to enter a negotiation.
- on_leave(state) A call back called after leaving a negotiation.
Table 74 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_mechanism_error(state)</code></td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td><code>on_negotiation_end(state)</code></td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td><code>on_negotiation_start(state)</code></td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td><code>on_notification(notification, notifier)</code></td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td><code>on_partner_proposal(state, agent_id, offer)</code></td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td><code>on_partner_refused_to_propose(state, agent_id)</code></td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td><code>on_partner_response(state, agent_id, ...)</code></td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td><code>on_round_end(state)</code></td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start(state)</code></td>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed()</code></td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>propose(state)</code></td>
<td>Propose a set of offers</td>
</tr>
<tr>
<td><code>propose_(state)</code></td>
<td></td>
</tr>
<tr>
<td><code>rtype</code></td>
<td>Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]</td>
</tr>
<tr>
<td><code>respond(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
<tr>
<td><code>respond_(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
</tbody>
</table>

Attributes Documentation

`capabilities`  
Agent capabilities

  **Return type** Dict[str, Any]

`eu`

The utility function in the given negotiation taking opponent model into account.

**Remarks:**

- If no utility_function is internally stored, `eu` still returns a valid callable that returns None for everything.

  **Return type** Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Union[Distribution, float, None]]

`id`

The unique ID of this entity

`name`

A convenient name of the entity (intended primarily for printing/logging/debugging).

`reserved_value`

Reserved value is what the agent gets if no agreement is reached in the negotiation.

`utility_function`

`uuid`

The unique ID of this entity
Methods Documentation

**add_capabilities** (*capabilities*)

Adds named capabilities to the agent.

- **Parameters**
  - **capabilities** (*dict*) – The capabilities to be added as a dict

- **Return type** None

- **Returns** None

  **Remarks:** It is the responsibility of the caller to be really capable of added capabilities.

**aspiration** (*t*)

The aspiration level

- **Parameters**
  - **t** (*float*) – relative time (a number between zero and one)

- **Return type** float

- **Returns** aspiration level

**aspiration_init** (*max_aspiration, aspiration_type, above_reserved_value=True*)

- **Parameters**
  - **max_aspiration** (*float*) –
  - **aspiration_type** (*Union*[str, int, float]*) –
  - **above_reserved_value** –

**before_death** (*ctx*)

Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

- **Return type** bool

**compare** (*first, second*)

Compares two offers using the ufun

- **Parameters**
  - **first** (*Outcome*) – First outcome to be compared
  - **second** (*Outcome*) – Second outcome to be compared

- **Returns** An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

- **Return type** UtilityValue

**counter** (*state, offer*)

Called to counter an offer

- **Parameters**
  - **state** (*MechanismState*) – MechanismState giving current state of the negotiation.
  - **offer** (*Union*[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be countered. None means no offer and the agent is requested to propose an offer

- **Returns** The response to the given offer with a counter offer if the response is REJECT

- **Return type** Tuple[ResponseType, Outcome]

**classmethod create** (*args, **kwargs*)

Creates an object and returns a proxy to it.
**isin**(*negotiation_id*)
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters**
- **negotiation_id** *(Optional*[str]*) – The negotiation ID tested. If None, it means ANY negotiation

**Returns**
- True if participating in the given negotiation (or any negotiation if it was None)

**Return type** bool

**join**(*ami, state, *, ufun=None, role='agent'*)
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**
- **ami** *(AgentMechanismInterface)* – The negotiation.
- **state** *(MechanismState)* – The current state of the negotiation
- **ufun** *(UtilityFunction)* – The ufun function to use before any discounting.
- **role** *(str)* – role of the agent.

**Return type** bool

**Returns** bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

**on_leave**(*state*)
A call back called after leaving a negotiation.

**Parameters**
- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.

**Remarks:**
- MUST call the baseclass on_leave using super () if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_mechanism_error**(*state*)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state

**Parameters**
- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_negotiation_end**(*state*)
A call back called at each negotiation end

**Parameters**
- **state** *(MechanismState)* – MechanismState or one of its descendants giving the state at which the negotiation ended.
Remarks:
  • The default behavior is to do nothing.
  • Override this to hook some action

Return type None

`on_negotiation_start (state)`
A call back called at each negotiation start

Parameters `state (MechanismState) – MechanismState` giving current state of the negotiation.

Remarks:
  • You MUST call the super() version of this function either before or after your code when you are overriding it.

Return type None

`on_notification (notification, notifier)`
Called whenever the agent receives a notification

Parameters
  • `notification (Notification) – The notification!!`
  • `notifier (str) – The notifier!!`

Returns None

Remarks:
  • You MUST call the super() version of this function either before or after your code when you are overriding it.

`on_partner_proposal (state, agent_id, offer)`
A callback called by the mechanism when a partner proposes something

Parameters
  • `state (MechanismState) – MechanismState` giving the state of the negotiation when the offer was proposed.
  • `agent_id (str) – The ID of the agent who proposed`
  • `offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.`

Return type None

Returns None

`on_partner_refused_to_propose (state, agent_id)`
A callback called by the mechanism when a partner refuses to propose

Parameters
  • `state (MechanismState) – MechanismState` giving the state of the negotiation when the partner refused to offer.
  • `agent_id (str) – The ID of the agent who refused to propose`

Return type None

Returns None

`on_partner_response (state, agent_id, outcome, response)`
A callback called by the mechanism when a partner responds to some offer
Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner responded.
- **agent_id** (*str*) – The ID of the agent who responded
- **outcome** (*Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]*) – The proposal being responded to.
- **response** (*SAOResponse*) – The response

Return type None

Returns None

**on_round_end** (*state*)

A call back called at each negotiation round end

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type None

**on_round_start** (*state*)

A call back called at each negotiation round start

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action.

Return type None

**on_ufun_changed**

Called to inform the agent that its ufun has changed.

Remarks:

- You MUST call the super() version of this function either before or after your code when you are overriding it.

**propose** (*state*)

Propose a set of offers

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Return type *Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]*

Returns The outcome being proposed or None to refuse to propose

Remarks:

- This function guarantees that no agents can propose something with a utility value.
propose_\(\text{(state)}\)

Return type  \(\text{Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]}\)

respond\(\text{(state, offer)}\)

Respond to an offer.

Parameters

- \(\text{state (MechanismState)}\) – MechanismState giving current state of the negotiation.
- \(\text{offer (Outcome)}\) – offer being tested

Returns  The response to the offer

Return type  \(\text{ResponseType}\)

Remarks:
- The default implementation never ends the negotiation
- The default implementation asks the negotiator to \text{propose()}\ and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

ToughNegotiator

\text{class negmas.sao.ToughNegotiator (name=\text{None}, parent=\text{None}, dynamic_ufun=\text{True}, can_propose=\text{True}, ufun=\text{None})}

Bases: \text{negmas.sao.SAONegotiator}

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>\text{\texttt{eu}}</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td>\text{\texttt{id}}</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Continued on next page
Table 75 – continued from previous page

<table>
<thead>
<tr>
<th>name</th>
<th>A convenient name of the entity (intended primarily for printing/logging/debugging).</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>add_capabilities(capabilities)</th>
<th>Adds named capabilities to the agent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>before_death(ctx)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>counter(state, offer)</td>
<td>Called to counter an offer</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td>join(ami, state, *[, ufun, role])</td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td>on_leave(state)</td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td>on_mechanism_error(state)</td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td>on_negotiation_end(state)</td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td>on_negotiation_start(state)</td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td>on_notification(notification, notifier)</td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td>on_partner_proposal(state, agent_id, offer)</td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td>on_partner_refused_to_propose(state, agent_id)</td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td>on_partner_response(state, agent_id, ...)</td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td>on_round_end(state)</td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td>on_round_start(state)</td>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td>on_ufun_changed()</td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
</tbody>
</table>

| propose(state)                 | Propose a set of offers                                                         |
| propose_(state)                |                                                                                  |

**Attributes Documentation**

- capabilities
  - Agent capabilities
    - Return type: `Dict[str, Any]`

---

5.6. negmas.sao Module 245
The utility function in the given negotiation taking opponent model into account.

**Remarks:**

- If no utility_function is internally stored, `eu` still returns a valid callable that returns None for everything.

**Return type** Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]], Union[Distribution, float, None]]

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**

Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**

The unique ID of this entity

### Methods Documentation

**add_capabilities** *(capabilities)*

Adds named capabilities to the agent.

**Parameters**

- **capabilities** *(dict)* – The capabilities to be added as a dict

**Return type** None

**Returns** None

**Remarks:** It is the responsibility of the caller to be really capable of added capabilities.

**before_death** *(cntxt)*

Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

**Return type** bool

**compare** *(first, second)*

Compares two offers using the `ufun`

**Parameters**

- **first** *(Outcome)* – First outcome to be compared
- **second** *(Outcome)* – Second outcome to be compared

**Returns** An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type** UtilityValue

**counter** *(state, offer)*

Called to counter an offer

**Parameters**

- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.
• offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be countered. None means no offer and the agent is requested to propose an offer.

**Returns**  The response to the given offer with a counter offer if the response is REJECT

**Return type**  Tuple[ResponseType, Outcome]

**classmethod create (**args, **kwargs)**

Creates an object and returns a proxy to it.

**isin (negotiation_id)**

Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters**  negotiation_id (Optional[str]) – The negotiation ID tested. If None, it means ANY negotiation.

**Returns**

**True if participating in the given negotiation (or any negotiation if it was None)**

**Return type**  bool

**join (ami, state, *, ufun=None, role='agent')**

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering.

**Parameters**

• ami (AgentMechanismInterface) – The negotiation.

• state (MechanismState) – The current state of the negotiation.

• ufun (UtilityFunction) – The ufun function to use before any discounting.

• role (str) – role of the agent.

**Return type**  bool

**Returns**  bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation.

**on_leave (state)**

A callback called after leaving a negotiation.

**Parameters**  state (MechanismState) – MechanismState giving current state of the negotiation.

**Remarks:**

• MUST call the baseclass on_leave using super () if you are going to override this.

• The default behavior is to do nothing.

• Override this to hook some action

**Return type**  None

**on_mechanism_error (state)**

A callback called whenever an error happens in the mechanism. The error and its explanation are accessible in state.

**Parameters**  state (MechanismState) – MechanismState giving current state of the negotiation.

**Remarks:**

• The default behavior is to do nothing.
- Override this to hook some action

Return type  None

on_negotiation_end(state)
A call back called at each negotiation end

Parameters state(MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

Remarks:
- The default behavior is to do nothing.
- Override this to hook some action

Return type  None

on_negotiation_start(state)
A call back called at each negotiation start

Parameters state(MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.

Return type  None

on_notification(notification, notifier)
Called whenever the agent receives a notification

Parameters
- notification(Notification) – The notification!!
- notifier(str) – The notifier!!

Returns  None

Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.

on_partner_proposal(state, agent_id, offer)
A callback called by the mechanism when a partner proposes something

Parameters
- state(MechanismState) – MechanismState giving the state of the negotiation when the offer was proposed.
- agent_id(str) – The ID of the agent who proposed
- offer(Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.

Return type  None

Returns  None

on_partner_refused_to_propose(state, agent_id)
A callback called by the mechanism when a partner refuses to propose

Parameters
• **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner refused to offer.

• **agent_id** (*str*) – The ID of the agent who refused to propose

**Returns** None

**on_partner_response** (*state*, *agent_id*, *outcome*, *response*)

A callback called by the mechanism when a partner responds to some offer

**Parameters**

• **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner responded.

• **agent_id** (*str*) – The ID of the agent who responded

• **outcome** (*Union*[OutcomeType, Tuple[*Union*[int, float, str, list]], Dict [*Union*[int, str], Union[int, float, str, list]]]) – The proposal being responded to.

• **response** (*SAOResponse*) – The response

**Returns** None

**on_round_end** (*state*)

A call back called at each negotiation round end

**Parameters**

*state* (*MechanismState*) – *MechanismState* giving current state of the negotiation.

**Remarks:**

• The default behavior is to do nothing.

• Override this to hook some action

**Return type** None

**on_round_start** (*state*)

A call back called at each negotiation round start

**Parameters**

*state* (*MechanismState*) – *MechanismState* giving current state of the negotiation.

**Remarks:**

• The default behavior is to do nothing.

• Override this to hook some action.

**Return type** None

**on_ufun_changed** ()

Called to inform the agent that its ufun has changed.

**Remarks:**

• You MUST call the super() version of this function either before or after your code when you are overriding it.

**propose** (*state*)

Propose a set of offers
Parameters `state` (MechanismState) – MechanismState giving current state of the negotiation.

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

Returns The outcome being proposed or None to refuse to propose

Remarks:

• This function guarantees that no agents can propose something with a utility value

`propose(state)`

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

`respond(state, offer)`

Respond to an offer.

Parameters

• `state` (MechanismState) – MechanismState giving current state of the negotiation.

• `offer` (Outcome) – offer being tested

Returns The response to the offer

Return typeResponseType

Remarks:

• The default implementation never ends the negotiation

• The default implementation asks the negotiator to propose `()` and accepts the `offer` if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

`respond(state, offer)`

Respond to an offer.

Parameters

• `state` (MechanismState) – MechanismState giving current state of the negotiation.

• `offer` (Outcome) – offer being tested

Returns The response to the offer

Return typeResponseType

Remarks:

• The default implementation never ends the negotiation except if an earlier end_negotiation notification is sent to the negotiator

• The default implementation asks the negotiator to propose `()` and accepts the `offer` if its utility was at least as good as the offer that it would have proposed (and above the reserved value).
**OnlyBestNegotiator**

```python
class negmas.sao.OnlyBestNegotiator(name=None, parent=None, dynamic_ufun=True, min_utility=0.95, top_fraction=0.05, best_first=False, probabilisic_offering=True, can_propose=True, ufun=None)
```

Bases: negmas.sao.SAONegotiator

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>eu</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_capabilities(capabilities)</td>
<td>Adds named capabilities to the agent.</td>
</tr>
<tr>
<td>before_death(cntxt)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>counter(state, offer)</td>
<td>Called to counter an offer</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td>join(ami, state, <em>[ufun, role]</em>)</td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td>on_leave(state)</td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td>on_mechanism_error(state)</td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td>on_negotiation_end(state)</td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td>on_negotiation_start(state)</td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td>on_notification(notification, notifier)</td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td>on_partner_proposal(state, agent_id, offer)</td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td>on_partner_refused_to_propose(state, agent_id)</td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td>on_partner_response(state, agent_id, ...)</td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td>on_round_end(state)</td>
<td>A call back called at each negotiation round end</td>
</tr>
<tr>
<td>on_round_start(state)</td>
<td>A call back called at each negotiation round start</td>
</tr>
<tr>
<td>on_ufun_changed()</td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td>propose(state)</td>
<td>Propose a set of offers</td>
</tr>
</tbody>
</table>

Continued on next page
propose_

**rtype** Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

respond(state, offer) Respond to an offer.
respond_(state, offer) Respond to an offer.

**Attributes Documentation**

capabilities
Agent capabilities

    **Return type** Dict[str, Any]

eu
The utility function in the given negotiation taking opponent model into account.

    **Remarks:**
    - If no utility_function is internally stored, **eu** still returns a valid callable that returns None for everything.

    **Return type** Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]] Union[Distribution, float, None]]

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

reserved_value
Reserved value is what the agent gets if no agreement is reached in the negotiation.

utility_function

uuid
The unique ID of this entity

**Methods Documentation**

**add_capabilities** (capabilities)
Adds named capabilities to the agent.

    **Parameters** capabilities (dict) – The capabilities to be added as a dict
    
    **Return type** None
    
    **Returns** None

    **Remarks:** It is the responsibility of the caller to be really capable of added capabilities.

**before_death** (cntxt)
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

    **Return type** bool
**compare**(first, second)

Compares two offers using the $ufun$

**Parameters**

- **first** *(Outcome)* – First outcome to be compared
- **second** *(Outcome)* – Second outcome to be compared

**Returns**

An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type** UtilityValue

**counter**(state, offer)

Called to counter an offer

**Parameters**

- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.
- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]], None]])* – The offer to be countered. None means no offer and the agent is requested to propose an offer

**Returns**

The response to the given offer with a counter offer if the response is REJECT

**Return type** Tuple[ResponseType, Outcome]

**classmethod create**(\*args, **kwargs)

Creates an object and returns a proxy to it.

**isin**(negotiation_id)

Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters** negotiation_id *(Optional[str]*) – The negotiation ID tested. If None, it means ANY negotiation

**Returns**

True if participating in the given negotiation (or any negotiation if it was None)

**Return type** bool

**join**(ami, state, *, ufun=None, role='agent')

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**

- **ami** *(AgentMechanismInterface)* – The negotiation.
- **state** *(MechanismState)* – The current state of the negotiation
- **ufun** *(UtilityFunction)* – The ufun function to use before any discounting.
- **role** *(str)* – role of the agent.

**Return type** bool

**Returns**

bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

**on_leave**(state)

A call back called after leaving a negotiation.

**Parameters** state *(MechanismState)* – MechanismState giving current state of the negotiation.
Remarks:

- MUST call the baseclass on_leave using super() if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_mechanism_error(state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state.

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_negotiation_end(state)
A call back called at each negotiation end

Parameters state (MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_negotiation_start(state)
A call back called at each negotiation start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

- You MUST call the super() version of this function either before or after your code when you are overriding it.

Return type None

on_notification(notification, notifier)
Called whenever the agent receives a notification

Parameters

- notification (Notification) – The notification!!
- notifier (str) – The notifier!!

Returns None

Remarks:

- You MUST call the super() version of this function either before or after your code when you are overriding it.
on_partner_proposal (state, agent_id, offer)
A callback called by the mechanism when a partner proposes something

Parameters

- state (MechanismState) – MechanismState giving the state of the negotiation when the offer was proposed.
- agent_id (str) – The ID of the agent who proposed
- offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.

Return type None
Returns None

on_partner_refused_to_propose (state, agent_id)
A callback called by the mechanism when a partner refuses to propose

Parameters

- state (MechanismState) – MechanismState giving the state of the negotiation when the partner refused to offer.
- agent_id (str) – The ID of the agent who refused to propose

Return type None
Returns None

on_partner_response (state, agent_id, outcome, response)
A callback called by the mechanism when a partner responds to some offer

Parameters

- state (MechanismState) – MechanismState giving the state of the negotiation when the partner responded.
- agent_id (str) – The ID of the agent who responded
- outcome (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal being responded to.
- response (SAOResponse) – The response

Return type None
Returns None

on_round_end (state)
A call back called at each negotiation round end

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_round_start (state)
A call back called at each negotiation round start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.
Remarks:

- The default behavior is to do nothing.
- Override this to hook some action.

Return type: None

on_ufun_changed()
Called to inform the agent that its ufun has changed.
Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.

propose(state)
Propose a set of offers
Parameters:
- state (MechanismState) – MechanismState giving current state of the negotiation.

Return type: Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int,str], Union[int, float, str, list]], None]

Returns: The outcome being proposed or None to refuse to propose.
 Remarks:
- This function guarantees that no agents can propose something with a utility value.

propose_(state)

Return type: Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int,str], Union[int, float, str, list]], None]

respond(state, offer)
Respond to an offer.
Parameters:
- state (MechanismState) – MechanismState giving current state of the negotiation.
- offer (Outcome) – offer being tested

Returns: The response to the offer.

Return type: ResponseType

Remarks:
- The default implementation never ends the negotiation.
- The default implementation asks the negotiator to propose`()` and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

respond_(state, offer)
Respond to an offer.
Parameters:
- state (MechanismState) – MechanismState giving current state of the negotiation.
- offer (Outcome) – offer being tested

Returns: The response to the offer.
Return type **ResponseType**

**Remarks:**
- The default implementation never ends the negotiation except if an earlier end_negotiation notification is sent to the negotiator
- The default implementation asks the negotiator to `propose()` and accepts the `offer` if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

### NaiveTitForTatNegotiator

**class** `negmas.sao.NaiveTitForTatNegotiator` *(name=None, parent=None, ufun=None, kindness=0.0, randomize_offer=False, always_concede=True, initial_concession='min')*

**Bases:** `negmas.sao.SAONegotiator`

Implements a generalized tit-for-tat strategy

### Attributes Summary

<table>
<thead>
<tr>
<th><strong>capabilities</strong></th>
<th>Agent capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>eu</strong></td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td><strong>id</strong></td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td><strong>reserved_value</strong></td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td><strong>utility_function</strong></td>
<td></td>
</tr>
<tr>
<td><strong>uuid</strong></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

| **add_capabilities** *(capabilities)* | Adds named capabilities to the agent. |
| **before_death** *(cntxt)* | Called whenever the parent is about to kill this negotiator. |
| **compare** *(first, second)* | Compares two offers using the ufun |
| **counter** *(state, offer)* | Called to counter an offer |
| **create** *(args, **kwargs)* | Creates an object and returns a proxy to it. |
| **is_in** *(negotiation_id)* | Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation. |
| **join** *(ami, state, *[ufun, role]*) | Called by the mechanism when the agent is about to enter a negotiation. |
| **on_leave** *(state)* | A call back called after leaving a negotiation. |
| **on_mechanism_error** *(state)* | A call back called whenever an error happens in the mechanism. |
| **on_negotiation_end** *(state)* | A call back called at each negotiation end |
| **on_negotiation_start** *(state)* | A call back called at each negotiation start |
| **on_notification** *(notification, notifier)* | Called whenever the agent receives a notification |
| **on_partner_proposal** *(state, agent_id, offer)* | A callback called by the mechanism when a partner proposes something |
Table 80 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_partner_refused_to_propose(state, agent_id)</code></td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td><code>on_partner_response(state, agent_id, . . . )</code></td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td><code>on_round_end(state)</code></td>
<td>A callback called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start(state)</code></td>
<td>A callback called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed()</code></td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>propose(state)</code></td>
<td>Propose a set of offers</td>
</tr>
<tr>
<td><code>propose_(state)</code></td>
<td></td>
</tr>
<tr>
<td><code>rtype</code></td>
<td><code>Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]</code></td>
</tr>
<tr>
<td><code>respond(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
<tr>
<td><code>respond_(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
</tbody>
</table>

Attributes Documentation

**capabilities**
Agent capabilities

*Return type* `Dict[str, Any]`

**eu**
The utility function in the given negotiation taking opponent model into account.

*Remarks:*

• If no `utility_function` is internally stored, `eu` still returns a valid callable that returns `None` for everything.

*Return type* `Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Union[Distribution, float, None]]`

**id**
The unique ID of this entity

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**
Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**
The unique ID of this entity

Methods Documentation

**add_capabilities**(capabilities)
Adds named capabilities to the agent.

*Parameters* capabilities (dict) – The capabilities to be added as a dict

*Return type* `None`
Returns None

Remarks: It is the responsibility of the caller to be really capable of added capabilities.

before_death (cntxt)
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

Return type bool

compare (first, second)
Compares two offers using the ufun

Parameters
  • first (Outcome) – First outcome to be compared
  • second (Outcome) – Second outcome to be compared

Returns An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

Return type UtilityValue

counter (state, offer)
Called to counter an offer

Parameters
  • state (MechanismState) – MechanismState giving current state of the negotiation.
  • offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be countered. None means no offer and the agent is requested to propose an offer

Returns The response to the given offer with a counter offer if the response is REJECT

Return type Tuple[ResponseType, Outcome]

classmethod create (*args, **kwargs)
Creates an object and returns a proxy to it.

isin (negotiation_id)
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

Parameters negotiation_id (Optional[str]) – The negotiation ID tested. If None, it means ANY negotiation

Returns

True if participating in the given negotiation (or any negotiation if it was None)

Return type bool

join (ami, state, *, ufun=None, role='agent')
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters
  • ami (AgentMechanismInterface) – The negotiation.
  • state (MechanismState) – The current state of the negotiation
  • ufun (UtilityFunction) – The ufun function to use before any discounting.
  • role (str) – role of the agent.
Return type bool

Returns bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

on_leave(state)
A call back called after leaving a negotiation.

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
- MUST call the baseclass on_leave using super() if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_mechanism_error(state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_negotiation_end(state)
A call back called at each negotiation end

Parameters state (MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

Remarks:
- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_negotiation_start(state)
A call back called at each negotiation start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.

Return type None

on_notification(notification, notifier)
Called whenever the agent receives a notification
Parameters

- `notification (Notification)` – The notification!!
- `notifier (str)` – The notifier!!

Returns None

Remarks:

- You MUST call the super() version of this function either before or after your code when you are overriding it.

`on_partner_proposal (state, agent_id, offer)`

A callback called by the mechanism when a partner proposes something

Parameters

- `state (MechanismState)` – MechanismState giving the state of the negotiation when the offer was proposed.
- `agent_id (str)` – The ID of the agent who proposed
- `offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])` – The proposal.

Returns None

`on_partner_refused_to_propose (state, agent_id)`

A callback called by the mechanism when a partner refuses to propose

Parameters

- `state (MechanismState)` – MechanismState giving the state of the negotiation when the partner refused to offer.
- `agent_id (str)` – The ID of the agent who refused to propose

Returns None

`on_partner_response (state, agent_id, outcome, response)`

A callback called by the mechanism when a partner responds to some offer

Parameters

- `state (MechanismState)` – MechanismState giving the state of the negotiation when the partner responded.
- `agent_id (str)` – The ID of the agent who responded
- `outcome (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]])` – The proposal being responded to.
- `response (SAOResponse)` – The response

Returns None

`on_round_end (state)`

A call back called at each negotiation round end

Parameters `state (MechanismState)` – MechanismState giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
• Override this to hook some action

Return type None

```python
on_round_start(state)
```
A call back called at each negotiation round start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• The default behavior is to do nothing.
• Override this to hook some action.

Return type None

```python
on_ufun_changed()
```
Called to inform the agent that its ufun has changed.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

```python
propose(state)
```
Propose a set of offers

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

Returns The outcome being proposed or None to refuse to propose

Remarks:
• This function guarantees that no agents can propose something with a utility value

```python
propose_(state)
```

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

```python
respond(state, offer)
```
Respond to an offer.

Parameters

• state (MechanismState) – MechanismState giving current state of the negotiation.

• offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
• The default implementation never ends the negotiation

• The default implementation asks the negotiator to propose`() and accepts the `offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).
**respond_**(state, offer)

Respond to an offer.

**Parameters**

- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.
- **offer** *(Outcome)* – offer being tested

**Returns** The response to the offer

**Return type** *ResponseType*

**Remarks:**

- The default implementation never ends the negotiation except if an earlier end_negotiation notification is sent to the negotiator
- The default implementation asks the negotiator to propose() and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

---

**SimpleTitForTatNegotiator**
	negmas.sao.SimpleTitForTatNegotiator

A simple tit-for-tat negotiator

**alias of** negmas.sao.NaiveTitForTatNegotiator

**NiceNegotiator**

class negmas.sao.NiceNegotiator(*args, **kwargs)

**Bases:** negmas.sao.SAONegotiator, negmas.sao.RandomProposalMixin

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>eu</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_capabilities(capabilities)</td>
<td>Adds named capabilities to the agent.</td>
</tr>
<tr>
<td>before_death(cntxt)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>counter(state, offer)</td>
<td>Called to counter an offer</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>init_random_proposal()</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 82 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isin(negotiation_id)</code></td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td><code>join(ami, state, *, ufun, role)</code></td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td><code>on_leave(state)</code></td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td><code>on_mechanism_error(state)</code></td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td><code>on_negotiation_end(state)</code></td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td><code>on_negotiation_start(state)</code></td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td><code>on_notification(notification, notifier)</code></td>
<td>Called whenever the agent receives a notification</td>
</tr>
<tr>
<td><code>on_partner_proposal(state, agent_id, offer)</code></td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td><code>on_partner_refused_to_propose(state, agent_id)</code></td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td><code>on_partner_response(state, agent_id, ...)</code></td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td><code>on_round_end(state)</code></td>
<td>A callback called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start(state)</code></td>
<td>A callback called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed()</code></td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>propose(state)</code></td>
<td>Propose a set of offers</td>
</tr>
<tr>
<td><code>propose_(state)</code></td>
<td></td>
</tr>
<tr>
<td><code>rtype Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]</code></td>
<td></td>
</tr>
<tr>
<td><code>respond(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
<tr>
<td><code>respond_(state, offer)</code></td>
<td>Respond to an offer.</td>
</tr>
</tbody>
</table>

Attributes Documentation

**capabilities**

Agent capabilities

**Return type** `Dict[str, Any]`

**eu**

The utility function in the given negotiation taking opponent model into account.

**Remarks:**

- If no utility function is internally stored, `eu` still returns a valid callable that returns `None` for everything.

**Return type** `Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Union[Distribution, float, None]]`

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).
reserved_value
Reserved value is what the agent gets if no agreement is reached in the negotiation.

utility_function

uuid
The unique ID of this entity

Methods Documentation

add_capabilities(capabilities)
Adds named capabilities to the agent.

Parameters
   capabilities (dict) – The capabilities to be added as a dict

Return type
   None

Remarks:
   It is the responsibility of the caller to be really capable of added capabilities.

before_death(cntxt)
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

Return type
   bool

compare(first, second)
Compares two offers using the ufun

Parameters
   • first (Outcome) – First outcome to be compared
   • second (Outcome) – Second outcome to be compared

Returns
   An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

Return type
   UtilityValue

counter(state, offer)
Called to counter an offer

Parameters
   • state (MechanismState) – MechanismState giving current state of the negotiation.
   • offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]) – The offer to be countered. None means no offer and the agent is requested to propose an offer

Returns
   The response to the given offer with a counter offer if the response is REJECT

Return type
   Tuple[ResponseType, Outcome]

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

init_random_proposal()

isin(negotiation_id)
Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.
Parameters **negotiation_id** *(Optional[str])* – The negotiation ID tested. If None, it means ANY negotiation

Returns

True if participating in the given negotiation (or any negotiation if it was None)

Return type **bool**

**join**(ami, state, *, ufun=None, role='agent')

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters

- **ami**(AgentMechanismInterface) – The negotiation.
- **state**(MechanismState) – The current state of the negotiation
- **ufun**(UtilityFunction) – The ufun function to use before any discounting.
- **role**(str) – role of the agent.

Return type **bool**

Returns **bool** indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

**on_leave**(state)

A call back called after leaving a negotiation.

Parameters **state**(MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

- MUST call the baseclass **on_leave** using **super** () if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

Return type **None**

**on_mechanism_error**(state)

A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in **state**

Parameters **state**(MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type **None**

**on_negotiation_end**(state)

A call back called at each negotiation end

Parameters **state**(MechanismState or one of its descendants) giving the state at which the negotiation ended.

Remarks:

- The default behavior is to do nothing.
• Override this to hook some action

Return type None

on_negotiation_start (state)
A callback called at each negotiation start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

Return type None

on_notification (notification, notifier)
Called whenever the agent receives a notification

Parameters
• notification (Notification) – The notification!!
• notifier (str) – The notifier!!

Returns None

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

on_partner_proposal (state, agent_id, offer)
A callback called by the mechanism when a partner proposes something

Parameters
• state (MechanismState) – MechanismState giving the state of the negotiation when the offer was proposed.
• agent_id (str) – The ID of the agent who proposed
• offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.

Return type None

Returns None

on_partner_refused_to_propose (state, agent_id)
A callback called by the mechanism when a partner refuses to propose

Parameters
• state (MechanismState) – MechanismState giving the state of the negotiation when the partner refused to offer.
• agent_id (str) – The ID of the agent who refused to propose

Return type None

Returns None

on_partner_response (state, agent_id, outcome, response)
A callback called by the mechanism when a partner responds to some offer

Parameters
• state (MechanismState) – MechanismState giving the state of the negotiation when the partner responded.
• **agent_id** (*str*) – The ID of the agent who responded

• **outcome** (*Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]*) – The proposal being responded to.

• **response** (*SAOResponse*) – The response

**Return type** None

**Returns** None

### on_round_end (*state*)

A callback called at each negotiation round end

**Parameters**

- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks:**

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

### on_round_start (*state*)

A callback called at each negotiation round start

**Parameters**

- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Remarks:**

- The default behavior is to do nothing.
- Override this to hook some action.

**Return type** None

### on_ufun_changed ()

Called to inform the agent that its ufun has changed.

**Remarks:**

- You MUST call the super() version of this function either before or after your code when you are overriding it.

### propose (*state*)

Propose a set of offers

**Parameters**

- **state** (*MechanismState*) – MechanismState giving current state of the negotiation.

**Return type** *Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]*

**Returns** The outcome being proposed or None to refuse to propose

**Remarks:**

- This function guarantees that no agents can propose something with a utility value

### propose_ (*state*)

**Return type** *Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]*
**respond** (state, offer)

Respond to an offer.

**Parameters**

- **state** (MechanismState) – MechanismState giving current state of the negotiation.
- **offer** (Outcome) – offer being tested

**Returns** The response to the offer

**Return type** ResponseType

**Remarks:**

- The default implementation never ends the negotiation
- The default implementation asks the negotiator to propose() and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

**respond_** (state, offer)

Respond to an offer.

**Parameters**

- **state** (MechanismState) – MechanismState giving current state of the negotiation.
- **offer** (Outcome) – offer being tested

**Returns** The response to the offer

**Return type** ResponseType

**Remarks:**

- The default implementation never ends the negotiation except if an earlier end_negotiation notification is sent to the negotiator
- The default implementation asks the negotiator to propose() and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

### SAOController

**class** negmas.sao.SAOController (default_negotiator_type=None, de-

- **default_negotiator_params=None, name=None**) 

**Bases:** negmas.negotiators.Controller

A controller that can manage multiple negotiators taking full or partial control from them.

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

5.6. negmas.sao Module
### call(negotiator, method, *args, **kwargs)
Calls the given method on the given negotiator safely without causing recursion.

- **negotiator** (*PassThroughNegotiator*)
- **method** (*str*)
- **args**
- **kwargs**

### create(*args, **kwargs)
Creates an object and returns a proxy to it.

### create_negotiator([negotiator_type, name, cnxt])
Creates a negotiator passing it the context

### join(negotiator_id, ami, state, *[ufun, role]*)
Called by the mechanism when the agent is about to enter a negotiation.

### kill_negotiator(negotiator_id[, force])
Kills the negotiator sending it an `before_death` message.

### on_leave(negotiator_id, state)
A call back called after leaving a negotiation.

### on_mechanism_error(negotiator_id, state)
A call back called whenever an error happens in the mechanism.

### on_negotiation_end(negotiator_id, state)
A call back called at each negotiation end

### on_negotiation_start(negotiator_id, state)
A call back called at each negotiation start

### on_notification(negotiator_id, notification, ...)

### on_round_end(negotiator_id, state)
A call back called at each negotiation round end

### on_round_start(negotiator_id, state)
A call back called at each negotiation round start

### on_ufun_changed(negotiator_id)
Called to inform the agent that its ufun has changed.

### propose(negotiator_id, state)

```
rtype Union[OutcomeType,  
           Tuple[Union[int,  
                  float, str, list]],  
           Dict[Union[int, str],  
                Union[int, float, str, list]], None]
```

### respond(negotiator_id, state, offer)

```
rtype ResponseType
```

### Attributes Documentation

**id**
The unique ID of this entity

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).

**uuid**
The unique ID of this entity

### Methods Documentation

**call(negotiator, method, *args, **kwargs)**
Calls the given method on the given negotiator safely without causing recursion. The controller MUST use this function to access any callable on the negotiator

**Parameters**

- **negotiator** (*PassThroughNegotiator*)
- **method** (*str*)
- **args**
- **kwargs**
Returns:

```python
classmethod create(*args, **kwargs)
```

Creates an object and returns a proxy to it.

```python
create_negotiator(negotiator_type=None, name=None, cntxt=None, **kwargs)
```

Creates a negotiator passing it the context.

**Parameters**

- `negotiator_type` (Union[Union[str, Type[PassThroughNegotiator]], None]) – Type of the negotiator to be created
- `name` (Optional[Union[str]]) – negotiator name
- `cntxt` (Optional[Dict[str, None]]) – The context to be associated with this negotiator. It will not be passed to the negotiator
- `**kwargs` – any key-value pairs to be passed to the negotiator constructor

**Returns**

The negotiator to be controlled

**Return type**

`PassThroughNegotiator`

```python
join(negotiator_id, ami, state, *, ufun=None, role='agent')
```

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering.

**Parameters**

- `negotiator_id` (str) – The negotiator ID
- `ami` (AgentMechanismInterface) – The negotiation.
- `state` (MechanismState) – The current state of the negotiation
- `ufun` (UtilityFunction) – The ufun function to use before any discounting.
- `role` (str) – role of the agent.

**Return type**

`bool`

**Returns**

bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation.

```python
kill_negotiator(negotiator_id, force=False)
```

Kills the negotiator sending it an before_death message.

**Parameters**

- `negotiator_id` (str) – The ID of the negotiator to kill.
- `force` (bool) – Whether to kill the negotiator in case it refused to die.

**Remarks:**

- Killing a negotiator amounts to nothing more than removing it from the list of negotiators maintained by the controller.

**Return type**

`None`

```python
on_leave(negotiator_id, state)
```

A call back called after leaving a negotiation.

**Parameters**

- `negotiator_id` (str) – The negotiator ID
- `state` (MechanismState) – MechanismState giving current state of the negotiation.
Return type None

**on_mechanism_error** *(negotiator_id, state)*
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in `state`

Parameters
- `negotiator_id (str)` – The negotiator ID
- `state (MechanismState)` – `MechanismState` giving current state of the negotiation.

Return type None

**on_negotiation_end** *(negotiator_id, state)*
A call back called at each negotiation end

Parameters
- `negotiator_id (str)` – The negotiator ID
- `state (MechanismState)` – `MechanismState` or one of its descendants giving the state at which the negotiation ended.

Return type None

**on_negotiation_start** *(negotiator_id, state)*
A call back called at each negotiation start

Parameters
- `negotiator_id (str)` – The negotiator ID
- `state (MechanismState)` – `MechanismState` giving current state of the negotiation.

Return type None

**on_notification** *(negotiator_id, notification, notifier)*

**on_round_end** *(negotiator_id, state)*
A call back called at each negotiation round end

Parameters
- `negotiator_id (str)` – The negotiator ID
- `state (MechanismState)` – `MechanismState` giving current state of the negotiation.

Return type None

**on_round_start** *(negotiator_id, state)*
A call back called at each negotiation round start

Parameters
- `negotiator_id (str)` – The negotiator ID
- `state (MechanismState)` – `MechanismState` giving current state of the negotiation.

Return type None

**on_ufun_changed** *(negotiator_id)*
Called to inform the agent that its ufun has changed.

Parameters `negotiator_id (str)` – The negotiator ID

Remarks:
• You MUST call the super() version of this function either before or after your code when you are overriding it.

**propose**(negotiator_id, state)

*Return type* Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

**respond**(negotiator_id, state, offer)

*Return type* ResponseType

**JavaSAONegotiator**

class negmas.sao.JavaSAONegotiator(java_object, java_class_name, auto_load_java=False, outcome_type=<class 'dict'>)

*Bases:* negmas.sao.SAONegotiator, negmas.java.JavaCallerMixin

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
<td>Agent capabilities</td>
</tr>
<tr>
<td>eu</td>
<td>The utility function in the given negotiation taking opponent model into account.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_capabilities(capabilities)</td>
<td>Adds named capabilities to the agent.</td>
</tr>
<tr>
<td>before_death(cntxt)</td>
<td>Called whenever the parent is about to kill this negotiator.</td>
</tr>
<tr>
<td>compare(first, second)</td>
<td>Compares two offers using the ufun</td>
</tr>
<tr>
<td>counter(state, offer)</td>
<td>Called to counter an offer</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>from_dict(java_object, *args[, parent])</td>
<td>Creates a Java negotiator from an object returned from the JVM implementing PySAONegotiator</td>
</tr>
<tr>
<td>init_java_bridge(java_object, java_class_name)</td>
<td>initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java</td>
</tr>
<tr>
<td>isin(negotiation_id)</td>
<td>Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.</td>
</tr>
<tr>
<td>join(ami, state, <em>[ufun, role]</em>)</td>
<td>Called by the mechanism when the agent is about to enter a negotiation.</td>
</tr>
<tr>
<td>on_leave(state)</td>
<td>A call back called after leaving a negotiation.</td>
</tr>
<tr>
<td>on_mechanism_error(state)</td>
<td>A call back called whenever an error happens in the mechanism.</td>
</tr>
<tr>
<td>on_negotiation_end(state)</td>
<td>A call back called at each negotiation end</td>
</tr>
<tr>
<td>on_negotiation_start(state)</td>
<td>A call back called at each negotiation start</td>
</tr>
<tr>
<td>on_notification(notification, notifier)</td>
<td>Called whenever the agent receives a notification</td>
</tr>
</tbody>
</table>

Continued on next page
Table 86 – continued from previous page

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_partner_proposal</code> (state, agent_id, offer)</td>
<td>A callback called by the mechanism when a partner proposes something</td>
</tr>
<tr>
<td><code>on_partner_refused_to_propose</code> (state, agent_id)</td>
<td>A callback called by the mechanism when a partner refuses to propose</td>
</tr>
<tr>
<td><code>on_partner_response</code> (state, agent_id, ...)</td>
<td>A callback called by the mechanism when a partner responds to some offer</td>
</tr>
<tr>
<td><code>on_round_end</code> (state)</td>
<td>A callback called at each negotiation round end</td>
</tr>
<tr>
<td><code>on_round_start</code> (state)</td>
<td>A callback called at each negotiation round start</td>
</tr>
<tr>
<td><code>on_ufun_changed</code></td>
<td>Called to inform the agent that its ufun has changed.</td>
</tr>
<tr>
<td><code>propose</code> (state)</td>
<td>Propose a set of offers</td>
</tr>
<tr>
<td><code>propose_</code> (state)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>rtype</code> Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]</td>
</tr>
<tr>
<td><code>respond</code> (state, offer)</td>
<td>Respond to an offer.</td>
</tr>
<tr>
<td><code>respond_</code> (state, offer)</td>
<td>Respond to an offer.</td>
</tr>
</tbody>
</table>

Attributes Documentation

**capabilities**

Agent capabilities

**Return type** Dict[str, Any]

**eu**

The utility function in the given negotiation taking opponent model into account.

**Remarks:**

- If no utility function is internally stored, `eu` still returns a valid callable that returns None for everything.

**Return type** Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Union[Distribution, float, None]]

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**

Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**

The unique ID of this entity

Methods Documentation

**add_capabilities** (capabilities)

Adds named capabilities to the agent.
Parameters **capabilities** *(dict)* – The capabilities to be added as a dict

Returns None

Remarks: It is the responsibility of the caller to be really capable of added capabilities.

**before_death** *(cntxt)*
Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does not want to be killed but the controller can still force-kill it

Return type bool

**compare** *(first, second)*
Compares two offers using the ufun

Parameters

- **first** *(Outcome)* – First outcome to be compared
- **second** *(Outcome)* – Second outcome to be compared

Returns An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

Return type UtilityValue

**counter** *(state, offer)*
Called to counter an offer

Parameters

- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.
- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]], None]])* – The offer to be countered. None means no offer and the agent is requested to propose an offer

Returns The response to the given offer with a counter offer if the response is REJECT

Return type Tuple[ResponseType, Outcome]

**classmethod create** *(*args, **kwargs)*
Creates an object and returns a proxy to it.

**classmethod from_dict** *(java_object, *args, parent=None)*
Creates a Java negotiator from an object returned from the JVM implementing PySAONegotiator

Return type JavaSAONegotiator

**init_java_bridge** *(java_object, java_class_name, auto_load_java=False, python_shadow_object=None)*
Initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

Parameters

- **java_object** – A java object that already exists of the correct type. If given no new objects will be created
- **java_class_name** *(str)* – The type of the Java object to be created
- **auto_load_java** *(bool)* – When true, a JVM will be automatically created (if one is not available)
- **python_shadow_object** *(Optional[Any])* – A python object to shadow the java object. The object will just call the corresponding
on this shadow object whenever it needs. (method) –

Remarks:

• sets a member called java_object that can be used to access the corresponding Java object crated

• if python_shadow_object is given, it must be an object of a type that has an internal class
called Java which has a single member called ‘implements’ which is a list of one string element
representing the Java interface being implemented (it must be either jnegmas.PyCallable or an
extension of it).

isin(negotiation_id)
Is that agent participating in the given negotiation? Tests if the agent is participating in the given
negotiation.

Parameters negotiation_id (Optional[str]) – The negotiation ID tested. If
None, it means ANY negotiation

Returns

True if participating in the given negotiation (or any negotiation if it was None)

Return type bool

join(ami, state, *, ufun=None, role='agent')
Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

Parameters

• ami (AgentMechanismInterface) – The negotiation.

• state (MechanismState) – The current state of the negotiation

• ufun (UtilityFunction) – The ufun function to use before any discounting.

• role (str) – role of the agent.

Return type bool

Returns bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

on_leave(state)
A call back called after leaving a negotiation.

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

• MUST call the baseclass on_leave using super () if you are going to override this.

• The default behavior is to do nothing.

• Override this to hook some action

Return type None

on_mechanism_error(state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

• The default behavior is to do nothing.
• Override this to hook some action

**Return type** None

**on_negotiation_end**(state)

A call back called at each negotiation end

**Parameters**

- **state** (*MechanismState*) – *MechanismState* or one of its descendants giving the state at which the negotiation ended.

**Remarks:**

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_negotiation_start**(state)

A call back called at each negotiation start

**Parameters**

- **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

**Remarks:**

- You MUST call the super() version of this function either before or after your code when you are overriding it.

**Return type** None

**on_notification**(notification, notifier)

Called whenever the agent receives a notification

**Parameters**

- **notification** (*Notification*) – The notification!!
- **notifier** (*str*) – The notifier!!

**Returns** None

**Remarks:**

- You MUST call the super() version of this function either before or after your code when you are overriding it.

**on_partner_proposal**(state, agent_id, offer)

A callback called by the mechanism when a partner proposes something

**Parameters**

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the offer was proposed.
- **agent_id** (*str*) – The ID of the agent who proposed
- **offer** (**OutcomeType**, **Tuple**,**Dict**) – The proposal.

**Return type** None

**on_partner_refused_to_propose**(state, agent_id)

A callback called by the mechanism when a partner refuses to propose

**Parameters**
- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner refused to offer.
- **agent_id** (*str*) – The ID of the agent who refused to propose

**Returns** None

### on_partner_response (*state, agent_id, outcome, response*)
A callback called by the mechanism when a partner responds to some offer

**Parameters**
- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner responded.
- **agent_id** (*str*) – The ID of the agent who responded
- **outcome** (*Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]*) – The proposal being responded to.
- **response** (*SAOResponse*) – The response

**Returns** None

### on_round_end (*state*)
A call back called at each negotiation round end

**Parameters**
- **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Returns** None

### on_round_start (*state*)
A call back called at each negotiation round start

**Parameters**
- **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action.

**Returns** None

### on_ufun_changed()
Called to inform the agent that its ufun has changed.

**Remarks:**
- You MUST call the super() version of this function either before or after your code when you are overriding it.

### propose (*state*)
Propose a set of offers
Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Return type `Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]`

Returns The outcome being proposed or None to refuse to propose

Remarks:

- This function guarantees that no agents can propose something with a utility value

**propose_**(state)

Return type `Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]`

**respond**(state, offer)

Respond to an offer.

Parameters

- **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.
- **offer** (*Outcome*) – offer being tested

Returns The response to the offer

Return type `ResponseType`

Remarks:

- The default implementation never ends the negotiation
- The default implementation asks the negotiator to propose() and accepts the `offer` if its utility was at least as good as the offer that it would have proposed (and above the reserved value).
PassThroughSAONegotiator

class negmas.sao.PassThroughSAONegotiator (assume_normalized=True, uf...)

Bases: negmas.sao.SAONegotiator

A negotiator that acts as an end point to a parent Controller

Attributes Summary

capabilities | Agent capabilities
--- | ---
eu | The utility function in the given negotiation taking opponent model into account.
id | The unique ID of this entity
name | A convenient name of the entity (intended primarily for printing/logging/debugging).
reserved_value | Reserved value is what the agent gets if no agreement is reached in the negotiation.
utility_function | The unique ID of this entity
uuid | 

Methods Summary

add_capabilities | Adds named capabilities to the agent.
before_death | Called whenever the parent is about to kill this negotiator.
compare | Compares two offers using the uf
counter | Called to counter an offer
create | Creates an object and returns a proxy to it.
isin | Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.
join | Called by the mechanism when the agent is about to enter a negotiation.
on_leave | A call back called after leaving a negotiation.
on_mechanism_error | A call back called whenever an error happens in the mechanism.
on_negotiation_end | A call back called at each negotiation end
on_negotiation_start | A call back called at each negotiation start
on_notification | Called whenever the agent receives a notification
on_partner_proposal | A callback called by the mechanism when a partner proposes something
on_partner_refused_to_propose | A callback called by the mechanism when a partner refuses to propose
on_partner_response | A callback called by the mechanism when a partner responds to some offer
on_round_end | A call back called at each negotiation round end
on_round_start | A call back called at each negotiation round start
on_ufun_changed | Called to inform the agent that its uf has changed.
propose | Propose a set of offers

Continued on next page
### Attributes Documentation

**capabilities**

Agent capabilities

**Return type** `Dict[str, Any]`

**eu**

The utility function in the given negotiation taking opponent model into account.

**Remarks:**

- If no utility_function is internally stored, `eu` still returns a valid callable that returns `None` for everything.

**Return type** `Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]]], Union[Distribution, float, None]]]

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**reserved_value**

Reserved value is what the agent gets if no agreement is reached in the negotiation.

**utility_function**

**uuid**

The unique ID of this entity

### Methods Documentation

**add_capabilities(capabilities)**

Adds named capabilities to the agent.

**Parameters**

- `capabilities` (`dict`) – The capabilities to be added as a dict

**Return type** `None`

**Returns** `None`

**Remarks:** It is the responsibility of the caller to be really capable of added capabilities.

**before_death(cntxt)**

Called whenever the parent is about to kill this negotiator. It should return `False` if the negotiator does not want to be killed but the controller can still force-kill it

**Return type** `bool`
**compare** *(first, second)*

Compared two offers using the ufun

**Parameters**

- **first** *(Outcome)* – First outcome to be compared
- **second** *(Outcome)* – Second outcome to be compared

**Returns** An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type** UtilityValue

**counter** *(state, offer)*

Called to counter an offer

**Parameters**

- **state** *(MechanismState)* – MechanismState giving current state of the negotiation.
- **offer** *(Union[OutcomeType, Tuple[Union[int, float, str, list], Dict[Union[int, str], Union[int, float, str, list]], None]])* – The offer to be countered. None means no offer and the agent is requested to propose an offer

**Returns** The response to the given offer with a counter offer if the response is REJECT

**Return type** Tuple[ResponseType, Outcome]

**classmethod create**( *args, **kwargs*

Creates an object and returns a proxy to it.

**isin** *(negotiation_id)*

Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters** negotiation_id *(Optional[str])* – The negotiation ID tested. If None, it means ANY negotiation

**Returns** True if participating in the given negotiation (or any negotiation if it was None)

**Return type** bool

**join** *(ami, state, *, ufun=None, role='agent'*

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**

- **ami** *(AgentMechanismInterface)* – The negotiation.
- **state** *(MechanismState)* – The current state of the negotiation
- **ufun** *(UtilityFunction)* – The ufun function to use before any discounting.
- **role** *(str)* – role of the agent.

**Return type** bool

**Returns** bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

**on_leave** *(state)*

A call back called after leaving a negotiation.

**Parameters** state *(MechanismState)* – MechanismState giving current state of the negotiation.
Remarks:

- **MUST** call the baseclass `on_leave` using `super()` if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

`on_mechanism_error(state)`

A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in `state`.

**Parameters**

- `state (MechanismState)` – `MechanismState` giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

`on_negotiation_end(state)`

A call back called at each negotiation end.

**Parameters**

- `state (MechanismState)` – `MechanismState` or one of its descendants giving the state at which the negotiation ended.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

`on_negotiation_start(state)`

A call back called at each negotiation start.

**Parameters**

- `state (MechanismState)` – `MechanismState` giving current state of the negotiation.

Remarks:

- You **MUST** call the super() version of this function either before or after your code when you are overriding it.

**Return type** None

`on_notification(notification, notifier)`

Called whenever the agent receives a notification.

**Parameters**

- `notification (Notification)` – The notification!!
- `notifier (str)` – The notifier!!

**Returns** None

Remarks:

- You **MUST** call the super() version of this function either before or after your code when you are overriding it.
on_partner_proposal (state, agent_id, offer)
A callback called by the mechanism when a partner proposes something

Parameters

- state (MechanismState) – MechanismState giving the state of the negotiation when the offer was proposed.
- agent_id (str) – The ID of the agent who proposed
- offer (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal.

Return type None

Returns None

on_partner_refused_to_propose (state, agent_id)
A callback called by the mechanism when a partner refuses to propose

Parameters

- state (MechanismState) – MechanismState giving the state of the negotiation when the partner refused to offer.
- agent_id (str) – The ID of the agent who refused to propose

Return type None

Returns None

on_partner_response (state, agent_id, outcome, response)
A callback called by the mechanism when a partner responds to some offer

Parameters

- state (MechanismState) – MechanismState giving the state of the negotiation when the partner responded.
- agent_id (str) – The ID of the agent who responded
- outcome (Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]) – The proposal being responded to.
- response (SAOResponse) – The response

Return type None

Returns None

on_round_end (state)
A callback called at each negotiation round end

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type None

on_round_start (state)
A callback called at each negotiation round start

Parameters state (MechanismState) – MechanismState giving current state of the negotiation.
Remarks:
- The default behavior is to do nothing.
- Override this to hook some action.

Return type None

on_ufun_changed()
Called to inform the agent that its ufun has changed.
Remarks:
- You MUST call the super() version of this function either before or after your code when you are overriding it.

propose(state)
Propose a set of offers
Parameters state (MechanismState) – MechanismState giving current state of the negotiation.

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

Returns The outcome being proposed or None to refuse to propose

Remarks:
- This function guarantees that no agents can propose something with a utility value

propose_(state)

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]

respond(state, offer)
Respond to an offer.
Parameters

• state (MechanismState) – MechanismState giving current state of the negotiation.

• offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
- The default implementation never ends the negotiation
- The default implementation asks the negotiator to propose`()` and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

respond_(state, offer)
Respond to an offer.
Parameters

• state (MechanismState) – MechanismState giving current state of the negotiation.

• offer (Outcome) – offer being tested

Returns The response to the offer
Return type: `ResponseType`

Remarks:
- The default implementation never ends the negotiation except if an earlier `end_negotiation` notification is sent to the negotiator.
- The default implementation asks the negotiator to call `propose()` and accepts the `offer` if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

### 5.6.2 Class Inheritance Diagram

![Class Inheritance Diagram]

### 5.7 `negmas.genius` Module

Genius Negotiator: An agent used to connect to GENIUS agents (ver 8.0.4) and allow them to join negotiation mechanisms.

#### 5.7.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init_genius_bridge(path, port, force)</code></td>
<td>Initializes a genius connection</td>
</tr>
<tr>
<td><code>genius_bridge_is_running(port)</code></td>
<td>Checks whether a Genius Bridge is running.</td>
</tr>
</tbody>
</table>
**init_genius_bridge**

```python
negmas.genius.init_genius_bridge(path=None, port=0, force=False)
```

Initializes a genius connection

**Parameters**

- **path** (Optional[str]) – The path to a JAR file that runs negloader
- **port** (int) – port number to use
- **force** (bool) – Force trial even if an existing bridge is initialized

**Returns**

- **bool**

**genius_bridge_is_running**

```python
negmas.genius.genius_bridge_is_running(port=None)
```

Checks whether a Genius Bridge is running. A genius bridge allows you to use `GeniusNegotiator` objects.

**Remarks:**

You can start a Genius Bridge in at least two ways:

- execute the python function `init_genius_bridge()` in this module
- run “negmas genius” on the terminal

**Return type**

- **bool**

---

5.7.2 Classes

**GeniusNegotiator**

```python
GeniusNegotiator(java_class_name[, port, ...])
```

Encapsulates a Genius Negotiator

**GeniusNegotiator**

```python
class negmas.genius.GeniusNegotiator(java_class_name, port=None, domain_file_name=None, utility_file_name=None, keep_issue_names=True, keep_value_names=True, auto_load_java=False, can_propose=True, genius_bridge_path=None, name=None)
```

**Bases:** `negmas.sao.SAONegotiator`

Encapsulates a Genius Negotiator

**Attributes Summary**

- **capabilities**
  - Agent capabilities

- **eu**
  - The utility function in the given negotiation taking opponent model into account.

- **id**
  - The unique ID of this entity

- **is_connected**

- **java_name**

Continued on next page
### Table 91 – continued from previous page

<table>
<thead>
<tr>
<th>name</th>
<th>A convenient name of the entity (intended primarily for printing/logging/debugging).</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved_value</td>
<td>Reserved value is what the agent gets if no agreement is reached in the negotiation.</td>
</tr>
<tr>
<td>utility_function</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

- **add_capabilities** *(capabilities)* Adds named capabilities to the agent.
- **before_death** *(ctx)* Called whenever the parent is about to kill this negotiator.
- **compare** *(first, second)* Compares two offers using the ufun
- **counter** *(state, offer)* Called to counter an offer
- **create** *(*args, **kwargs)* Creates an object and returns a proxy to it.
- **isin** *(negotiation_id)* Tests if the agent is participating in the given negotiation.
- **join** *(ami, state, *[, ufun, role]*) Called by the mechanism when the agent is about to enter a negotiation.
- **negotiators** *(agent_based, party_based)* Returns a list of all available agents in genius 8.4.0
- **on_leave** *(state)* A call back called after leaving a negotiation.
- **on_mechanism_error** *(state)* A call back called whenever an error happens in the mechanism.
- **on_negotiation_end** *(state)* A call back called at each negotiation end
- **on_negotiation_start** *(state)* Called when the info starts.
- **on_notification** *(notification, notifier)* Called whenever the agent receives a notification
- **on_partner_proposal** *(state, agent_id, offer)* A callback called by the mechanism when a partner proposes something
- **on_partner_refused_to_propose** *(state, agent_id)* A callback called by the mechanism when a partner refuses to propose
- **on_partner_response** *(state, agent_id, ...)* A callback called by the mechanism when a partner responds to some offer
- **on_round_end** *(state)* A call back called at each negotiation round end
- **on_round_start** *(state)* A call back called at each negotiation round start
- **on_ufun_changed** () Called to inform the agent that its ufun has changed.
- **parse** *(action)* Parses an action into and a ResponseType and an Outcome (if one is included) :type action: str
  :param action: ...
- **propose** *(state)* Propose a set of offers
- **propose** *(state)*

#### rtype

- **Union** *(OutcomeType, Tuple*[Union*[int, float, str, list]], Dict*[Union*[int, str], Union*[int, float, str, list]], None]*

- **random_negotiator** *(agent_based, ...)* Returns an agent with a random class name
- **respond** *(state, offer)* Respond to an offer.
- **respond** *(state, offer)* Respond to an offer.

Continued on next page
Attributes Documentation

capabilities
   Agent capabilities
      Return type: Dict[str, Any]

eu
   The utility function in the given negotiation taking opponent model into account.
      Remarks:
         • If no utility_function is internally stored, eu still returns a valid callable that returns None
           for everything.
      Return type: Callable[[Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]]]], Distribution, float, None]]

id
   The unique ID of this entity

is_connected

java_name

name
   A convenient name of the entity (intended primarily for printing/logging/debugging).

reserved_value
   Reserved value is what the agent gets if no agreement is reached in the negotiation.

utility_function

uuid
   The unique ID of this entity

Methods Documentation

add_capabilities (capabilities)
   Adds named capabilities to the agent.
      Parameters: capabilities (dict) – The capabilities to be added as a dict
      Return type: None
      Returns: None
      Remarks: It is the responsibility of the caller to be really capable of added capabilities.

before_death (cntxt)
   Called whenever the parent is about to kill this negotiator. It should return False if the negotiator does
   not want to be killed but the controller can still force-kill it
      Return type: bool

compare (first, second)
   Compares two offers using the ufun
      Parameters
• **first** (*Outcome*) – First outcome to be compared

• **second** (*Outcome*) – Second outcome to be compared

**Returns** An estimate of the differences between the two outcomes. It can be a real number between -1, 1 or a probability distribution over the same range.

**Return type** *UtilityValue*

counter (*state, offer*)

Called to counter an offer

**Parameters**

• **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

• **offer** (*Union[OutcomeType, Tuple[Union[int, float, str, list]], Dict[Union[int, str], Union[int, float, str, list]], None]*) – The offer to be countered. None means no offer and the agent is requested to propose an offer

**Returns** The response to the given offer with a counter offer if the response is REJECT

**Return type** *Tuple[ResponseType, Outcome]*

classmethod create (*args, **kwargs*)

Creates an object and returns a proxy to it.

isin (*negotiation_id*)

Is that agent participating in the given negotiation? Tests if the agent is participating in the given negotiation.

**Parameters** *negotiation_id* (*Optional[str]*) – The negotiation ID tested. If None, it means ANY negotiation

**Returns**

True if participating in the given negotiation (or any negotiation if it was None)

**Return type** *bool*

join (*ami, state, *, ufun=None, role='agent'*)

Called by the mechanism when the agent is about to enter a negotiation. It can prevent the agent from entering

**Parameters**

• **ami** (*AgentMechanismInterface*) – The negotiation.

• **state** (*MechanismState*) – The current state of the negotiation

• **ufun** (*UtilityFunction*) – The ufun function to use before any discounting.

• **role** (*str*) – role of the agent.

**Return type** *bool*

**Returns** bool indicating whether or not the agent accepts to enter. If False is returned it will not enter the negotiation

classmethod negotiators (*agent_based=False, party_based=True*)

Returns a list of all available agents in genius 8.4.0

**Parameters**

• **agent_based** – Old agents based on the Java class Negotiator

• **party_based** – Newer agents based on the Java class AbstractNegotiationParty

**Returns:**

**Return type** *List[str]*
**on_leave**(state)
A call back called after leaving a negotiation.

**Parameters**
- *state*(MechanismState) – MechanismState giving current state of the negotiation.

**Remarks:**
- MUST call the baseclass on_leave using super() if you are going to override this.
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_mechanism_error**(state)
A call back called whenever an error happens in the mechanism. The error and its explanation are accessible in state

**Parameters**
- *state*(MechanismState) – MechanismState giving current state of the negotiation.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_negotiation_end**(state)
A call back called at each negotiation end

**Parameters**
- *state*(MechanismState) – MechanismState or one of its descendants giving the state at which the negotiation ended.

**Remarks:**
- The default behavior is to do nothing.
- Override this to hook some action

**Return type** None

**on_negotiation_start**(state)
Called when the info starts. Connects to the JVM.

**Return type** None

**on_notification**(notification, notifier)
Called whenever the agent receives a notification

**Parameters**
- *notification*(Notification) – The notification!!
- *notifier*(str) – The notifier!!

**Returns** None

**Remarks:**
- You MUST call the super() version of this function either before or after your code when you are overriding it.

**on_partner_proposal**(state, agent_id, offer)
A callback called by the mechanism when a partner proposes something
Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the offer was proposed.
- **agent_id** (*str*) – The ID of the agent who proposed
- **offer** (*Outcome*) – The proposal.

Returns None

`on_partner_refused_to_propose(state, agent_id)`
A callback called by the mechanism when a partner refuses to propose

Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner refused to offer.
- **agent_id** (*str*) – The ID of the agent who refused to propose

Return type None

Returns None

`on_partner_response(state, agent_id, outcome, response)`
A callback called by the mechanism when a partner responds to some offer

Parameters

- **state** (*MechanismState*) – *MechanismState* giving the state of the negotiation when the partner responded.
- **agent_id** (*str*) – The ID of the agent who responded
- **outcome** (*Outcome*) – The proposal being responded to.
- **response** (*ResponseType*) – The response

Returns None

`on_round_end(state)`
A callback called at each negotiation round end

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action

Return type None

`on_round_start(state)`
A callback called at each negotiation round start

Parameters **state** (*MechanismState*) – *MechanismState* giving current state of the negotiation.

Remarks:

- The default behavior is to do nothing.
- Override this to hook some action.

Return type None
on_ufun_changed()
Called to inform the agent that its ufun has changed.

Remarks:
• You MUST call the super() version of this function either before or after your code when you are
overriding it.

parse (action)
Parses an action into and a ResponseType and an Outcome (if one is included) :
type action: str
:param action:

Returns:

propose (state)
Propose a set of offers

Parameters state (MechanismState) – MechanismState giving current state of
the negotiation.

Return type Outcome

Returns The outcome being proposed or None to refuse to propose

Remarks:
• This function guarantees that no agents can propose something with a utility value

propose_ (state)

Return type Union[OutcomeType, Tuple[Union[int, float, str, list]],
Dict[Union[int, str], Union[int, float, str, list]], None]

classmethod random_negotiator (agent_based=True, party_based=True, port=None,
domain_file_name=None, utility_file_name=None, keep_issue_names=True,keep_value_names=True,
auto_load_java=False, can_propose=True, name=None)

Returns an agent with a random class name

Parameters
• agent_based – Old agents based on the Java class Negotiator
• party_based – Newer agents based on the Java class AbstractNegotiationParty

Return type GeniusNegotiator

Returns GeniusNegotiator an agent with a random java class

respond (state, offer)
Respond to an offer.

Parameters
• state (MechanismState) – MechanismState giving current state of the ne-
gotiation.
• offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
• The default implementation never ends the negotiation
• The default implementation asks the negotiator to propose() and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

respond_(state, offer)

Respond to an offer.

Parameters
- state (MechanismState) – MechanismState giving current state of the negotiation.
- offer (Outcome) – offer being tested

Returns The response to the offer

Return type ResponseType

Remarks:
• The default implementation never ends the negotiation except if an earlier end_negotiation notification is sent to the negotiator
• The default implementation asks the negotiator to propose() and accepts the offer if its utility was at least as good as the offer that it would have proposed (and above the reserved value).

test()

Return type str

5.7.3 Class Inheritance Diagram

![Class Inheritance Diagram]

5.8 negmas.situated Module

This module defines the base classes for worlds within which multiple agents engage in situated negotiations.

The Agent class encapsulates the managing entity that creates negotiators to engage in negotiations within a world Simulation in order to maximize its own total utility.
5.8.1 Remarks:

- When immediate_negotiations is true, negotiations start in the same step they are registered in (they may also end in that step) and negotiation_speed_multiple steps of it are conducted. That entails that requesting a negotiation may result in new contracts in the same time-step only if immediate_negotiations is true.

5.8.2 Simulation steps:

1. prepare custom stats (call _pre_step_stats)
2. sign contracts that are to be signed at this step calling on_contract_signed as needed
3. step all existing negotiations negotiation_speed_multiple times handling any failed negotiations and creating contracts for any resulting agreements
4. run all Entity objects registered (i.e. all agents) in the predefined simulation_order.
5. execute contracts that are executable at this time-step handling any breaches
6. allow custom simulation steps to run (call _simulation_step)
7. remove any negotiations that are completed!
8. update basic stats
9. update custom stats (call _post_step_stats)

5.8.3 Functions

```python
save_stats(world, log_dir[, params, ...]) Saves the statistics of a world run.
```

```python
negmas.situated.save_stats(world, log_dir, params=None, stats_file_name=None)
Saves the statistics of a world run.
```

Parameters

- **world** (*World*) -
- **log_dir** (*str*) -
- **params** (*Optional*[*Dict*[str, Any]*) -
- **stats_file_name** (*Optional*[str]*) – File name to use for stats file(s) without extension

Returns:

5.8.4 Classes

```python
Action(type, params) An action that an Agent can execute in a World through the Simulator.
```

```python
Contract([partners, agreement, annotation, ...]) A agreement definition which encapsulates an agreement with partners and extra information
```

```python
Breach(contract, perpetrator, type[, ...])
```

```python
BreachProcessing
```

```python
Agent([name]) Base class for all agents that can run within a World and engage in situated negotiations
```

Continued on next page
Table 94 – continued from previous page

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BulletinBoard(name)</td>
<td>The bulletin-board which carries all public information.</td>
</tr>
<tr>
<td>World(bulletin_board, n_steps, time_limit)</td>
<td>Base world class encapsulating a world that runs a simulation with several agents interacting within some dynamically changing environment.</td>
</tr>
<tr>
<td>Entity(name)</td>
<td>Defines an entity that is a part of the world but does not participate in the simulation.</td>
</tr>
<tr>
<td>AgentWorldInterface(world, agent)</td>
<td>Agent World Interface class</td>
</tr>
<tr>
<td>NegotiationInfo(mechanism, partners, ...)</td>
<td>Saves information about a negotiation</td>
</tr>
<tr>
<td>RenegotiationRequest(publisher, ...)</td>
<td>A request for renegotiation.</td>
</tr>
<tr>
<td>StatsMonitor(name)</td>
<td>A monitor object capable of receiving stats of a world</td>
</tr>
<tr>
<td>WorldMonitor(name)</td>
<td>A monitor object capable of monitoring a world.</td>
</tr>
</tbody>
</table>

Action

class negmas.situated.Action(type, params)

    Bases: object

    An action that an Agent can execute in a World through the Simulator.

Contract

class negmas.situated.Contract(partners=<factory>, agreement=None, annotations=<factory>, issues=<factory>, signed_at=None, concluded_at=None, nullified_at=None, to_be_signed_at=None, signatures=<factory>, mechanism_state=None, id=<factory>)

    Bases: negmas.outcomes.OutcomeType

    A agreement definition which encapsulates an agreement with partners and extra information

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agreement</td>
<td>The actual agreement of the negotiation in the form of an Outcome in the Issue space defined by issues</td>
</tr>
<tr>
<td>concluded_at</td>
<td>The time-step at which the contract was concluded (but it is still not binding until signed)</td>
</tr>
<tr>
<td>mechanism_state</td>
<td>The mechanism state at the contract conclusion</td>
</tr>
<tr>
<td>nullified_at</td>
<td>The time-step at which the contract was nullified after being signed.</td>
</tr>
<tr>
<td>signed_at</td>
<td>The time-step at which the contract was signed.</td>
</tr>
<tr>
<td>to_be_signed_at</td>
<td>The time-step at which the contract should be signed</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asdict()</td>
<td>Converts the outcome to a dict containing all fields</td>
</tr>
<tr>
<td>astuple()</td>
<td>Converts the outcome to a tuple where the order of items is the same as they are defined as fields</td>
</tr>
<tr>
<td>get(name[, default])</td>
<td>Acts like dict.get</td>
</tr>
</tbody>
</table>

Continued on next page
### Attributes Documentation

**agreement = None**
- The actual agreement of the negotiation in the form of an `Outcome` in the `Issue` space defined by `issues`

**concluded_at = None**
- The time-step at which the contract was concluded (but it is still not binding until signed)

**mechanism_state = None**
- The mechanism state at the contract conclusion

**nullified_at = None**
- The time-step at which the contract was nullified after being signed. That can happen if a partner declares bankruptcy

**signed_at = None**
- The time-step at which the contract was signed

**to_be_signed_at = None**
- The time-step at which the contract should be signed

### Methods Documentation

**asdict()**
- Converts the outcome to a dict containing all fields

**astuple()**
- Converts the outcome to a tuple where the order of items is the same as they are defined as fields

**get(name, default=None)**
- Acts like dict.get

**keys()**
- Return type `List[str]`

**values()**
- Return type `List[str]`

### Breach

**class negmas.situated.Breach(contract, perpetrator, type, victims=<factory>, level=1.0, step=-1, id=<factory>)**

**Bases:** `object`

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>level</code></td>
<td>Breach level defaulting to full breach (a number between 0 and 1)</td>
</tr>
<tr>
<td><code>step</code></td>
<td>The simulation step at which the breach occurred</td>
</tr>
</tbody>
</table>
Methods Summary

```python
as_dict()
```

Attributes Documentation

```python
level = 1.0
    Breach level defaulting to full breach (a number between 0 and 1)
step = -1
    The simulation step at which the breach occurred
```

Methods Documentation

```python
as_dict()
```

BreachProcessing

```python
class negmas.situated.BreachProcessing
    Bases: enum.Enum

    The way breaches are to be handled
```

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>META_NEGOTIATION</td>
<td>A meta negotiation is instantiated between victim and perpetrator to set re-negotiation issues.</td>
</tr>
<tr>
<td>NONE</td>
<td>The breach should always be reported in the breach list and no re-negotiation is allowed.</td>
</tr>
<tr>
<td>VICTIM_THEN_PERPETRATOR</td>
<td>The victim is asked to set the re-negotiation agenda then the perpetrator.</td>
</tr>
</tbody>
</table>

Attributes Documentation

```python
META_NEGOTIATION = 2
    A meta negotiation is instantiated between victim and perpetrator to set re-negotiation issues.
NONE = 0
    The breach should always be reported in the breach list and no re-negotiation is allowed.
VICTIM_THEN_PERPETRATOR = 1
    The victim is asked to set the re-negotiation agenda then the perpetrator.
```

Agent

```python
class negmas.situated.Agent(name=None)

    Base class for all agents that can run within a World and engage in situated negotiations
```

Attributes Summary
awi
id
name
requested_negotiations
running_negotiations
short_type_name
type_name
unsigned_contracts
uuid

Methods Summary

create(*args, **kwargs)
from_config(config[, section, ...])

init()
init_()

notify(notifiable, notification)
on_contract_breacheds(contract, breaches, ...)
on_contract_cancelleds(contract, rejectors)
on_contract_cancelleds_(contract, rejectors)
on_contract_executeds(contract)
on_contract_signeds(contract)
on_contract_signeds_(contract)
on_event(event, sender)
on_neg_request_accepteds(req_id, mechanism)
on_neg_request_accepteds_(req_id, mechanism)
on_neg_request_rejecteds(req_id, by)
on_neg_request_rejecteds_(req_id, by)
on_negotiation_failureps(partners, annotation, ...)
on_negotiation_failureps_(partners, ...)
on_negotiation_successes(contract, mechanism)
on_negotiation_successes_(contract, mechanism)
read_config(config[, section])
respond_to_negotiation_request_(initiator, ...)
respond_to_renegotiation_request_(contract, ...)

Continued on next page
Table 101 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set_renegotiation_agenda</code> (contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>sign_contract</code> (contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step</td>
</tr>
</tbody>
</table>

Attributes Documentation

**awi**

Gets the Agent-world interface.

**Return type** `AgentWorldInterface`

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**requested_negotiations**

The negotiations currently requested by the agent.

**Return type** `List[NegotiationRequestInfo]`

**Returns** A list of negotiation request information objects (`NegotiationRequestInfo`)

**running_negotiations**

The negotiations currently requested by the agent.

**Return type** `List[RunningNegotiationInfo]`

**Returns** A list of negotiation information objects (`RunningNegotiationInfo`)

**short_type_name**

Returns a short name of the type of this entity

**type_name**

Returns the name of the type of this entity

**unsigned_contracts**

All contracts that are not yet signed.

**Return type** `List[Contract]`

**uuid**

The unique ID of this entity

Methods Documentation

**classmethod create(**args**, **kwargs**)**

Creates an object and returns a proxy to it.

**classmethod from_config(**config**, **section=None**, **ignore_children=True**, **try_parsing_children=True**, **scope=None**)**

Creates an object of this class given the configuration info

**Parameters**

- **config** (`Union[str, dict]`) – Either a file name or a dictionary
- **section** (`Optional[str]`) – A section in the file or a key in the dictionary to use for loading params
• **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return

• **try_parsing_children** *(bool)* – If true the children will first be parsed as `ConfigReader` classes if they are not

• **types** *(e.g. int, str, float, Iterable[int|str|float]** *(simple)* –

• **scope** – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are

• **any children that are to be parsed.** *(having)* –

Returns An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing `scope=globals()` to this function is to get around the fact that in python `eval()` will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when `eval()` is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an `undefined_name` exception, the caller must pass her `globals()` as the scope.

**init**()
Called to initialize the agent after the world is initialized. The AWI is accessible at this point.

**init_()**
Called to initialize the agent after the world is initialized. The AWI is accessible at this point.

**notify** *(notifiable, notification)*

**on_contract_breached** *(contract, breaches, resolution)*
Called after complete processing of a contract that involved a breach.

Parameters

• **contract** *(Contract)* – The contract

• **breaches** *(List[Breach])* – All breaches committed (even if they were resolved)

• **resolution** *(Optional[Contract])* – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

**on_contract_cancelled** *(contract, rejectors)*
 Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_cancelled_**(contract, rejectors)
 Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_executed** *(contract)*
 Called after successful contract execution for which the agent is one of the partners.

**Return type** None

**on_contract_signed** *(contract)*
 Called whenever a contract is signed by all partners

**Return type** None
on_contract_signed_(contract)
Called whenever a contract is signed by all partners

    Return type None

on_event (event, sender)

on_neg_request_accepted (req_id, mechanism)
Called when a requested negotiation is accepted

on_neg_request_accepted_ (req_id, mechanism)
Called when a requested negotiation is accepted

on_neg_request_rejected (req_id, by)
Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if
the failure was for another reason

on_neg_request_rejected_ (req_id, by)
Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if
the failure was for another reason

on_negotiation_failure (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

    Return type None

on_negotiation_failure_ (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

    Return type None

on_negotiation_success (contract, mechanism)
Called whenever a negotiation ends with agreement

    Return type None

on_negotiation_success_ (contract, mechanism)
Called whenever a negotiation ends with agreement

    Return type None

classmethod read_config (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use
  for loading params

    Return type Dict[str, Any]

    Returns A dict ready to be parsed by from_config

Remarks:

respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism,
role, req_id)
Called when a negotiation request is received
respond_to_renegotiation_request

Called to respond to a renegotiation request

Parameters

• agenda (RenegotiationRequest) –
• contract (Contract) –
• breaches (List[Breach]) –

Returns:

Return type Optional[Negotiator]

set_renegotiation_agenda

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters

• contract (Contract) – The contract being breached
• breaches (List[Breach]) – All breaches on contract

Return type Optional[RenegotiationRequest]

Returns Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

sign_contract

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type Optional[str]

step

Called by the simulator at every simulation step

step_

Called at every time-step. This function is called directly by the world.

BulletinBoard

class BulletinBoard(name=None)


The bulletin-board which carries all public information. It consists of sections each with a dictionary of records.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>This property is intended for use only by the world manager.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>
Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_section(name)</td>
<td>Adds a section to the bulletin Board</td>
</tr>
<tr>
<td>announce(event)</td>
<td>Raises an event and informs all event sinks that are registered for notifications on this event type</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>from_config(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td>query(section, query[, query_keys])</td>
<td>Returns all records in the given section/sections of the bulletin-board that satisfy the query</td>
</tr>
<tr>
<td>read(section, key)</td>
<td>Reads the value associated with given key</td>
</tr>
<tr>
<td>read_config(config[, section])</td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td>record(section, value[, key])</td>
<td>Records data in the given section of the bulletin-board</td>
</tr>
<tr>
<td>register_listener(event_type, listener)</td>
<td></td>
</tr>
<tr>
<td>remove(section, *[, query, key, query_keys, ...])</td>
<td>Removes a value or a set of values from the bulletin Board</td>
</tr>
<tr>
<td>satisfies(value, query)</td>
<td></td>
</tr>
</tbody>
</table>

Attributes Documentation

data
This property is intended for use only by the world manager. No other agent is allowed to use it

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

short_type_name
Returns a short name of the type of this entity

type_name
Returns the name of the type of this entity

uuid
The unique ID of this entity

Methods Documentation

add_section(name)
Adds a section to the bulletin Board

Parameters:
- name (str) – Section name

Returns:
- Return type: None

announce(event)
 Raises an event and informs all event sinks that are registered for notifications on this event type

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)
Creates an object of this class given the configuration info
Parameters

- **config** (`Union[str, dict]`) – Either a file name or a dictionary
- **section** (`Optional[str]`) – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** (`bool`) – If true then children will be ignored and there will be a single return
- **try_parsing_children** (`bool`) – If true the children will first be parsed as `ConfigReader` classes if they are not
- **types** (`e.g. int, str, float, Iterable[int|str|float]`) –
- **scope** – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are
- **any children that are to be parsed.**

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**query** (`section, query, query_keys=False`)

Returns all records in the given section/sections of the bulletin-board that satisfy the query

Parameters

- **section** (`Union[str, List[str], None]`) – Either a section name, a list of sections or None specifying ALL public sections (see remarks)
- **query** (`Any`) – The query which is USUALLY a dict with conditions on it when querying values and a RegExp when
- **keys** (`querying`) –
- **query_keys** – Whether the query is to be applied to the keys or values.

**Returns** value pairs giving all records that satisfied the given requirements.

**Return type**

- A dictionary with key

Remarks:

- A public section is a section with a name that does not start with an underscore
- If a set of sections is given, and two records in different sections had the same key, only one of them will be returned
- Key queries use regular expressions and match from the beginning using the standard re.match function

**read** (`section, key`)

Reads the value associated with given key

Parameters
• **section** (*str*) – section name
• **key** (*str*) – key

**Return type**  Any

**Returns**  Content of that key in the bulletin-board

**classmethod read_config** (*config, section=None*)

Reads the configuration from a file or a dict and prepares it for parsing

**Parameters**

• **config** (*Union*[str, dict]*) – Either a file name or a dictionary
• **section** (*Optional*[str]*) – A section in the file or a key in the dictionary to use for loading params

**Return type**  Dict[str, Any]

**Returns**  A dict ready to be parsed by from_config

**Remarks:**

**record** (*section, value, key=None*)

Records data in the given section of the bulletin-board

**Parameters**

• **section** (*str*) – section name (can contain subsections separated by '/')
• **key** (*Optional*[str]*) – The key
• **value** (*Any*) – The value

**Return type**  None

**register_listener** (*event_type, listener*)

**remove** (*section, *, query=None, key=None, query_keys=False, value=None*)

Removes a value or a set of values from the bulletin Board

**Parameters**

• **section** (*Union*[str, List[str], None]*) – The section
• **query** (*Optional*[Any]*) – the query to use to select what to remove
• **key** (*Optional*[str]*) – the key to remove (no need to give a full query)
• **query_keys** (*bool*) – Whether to apply the query (if given) to keys or values
• **value** (*Optional*[Any]*) – Value to be removed

**Returns**  Success of failure

**Return type**  bool

**classmethod satisfies** (*value, query*)

**Return type**  bool
World

class negmas.situated.World(bulletin_board=None, n_steps=10000, time_limit=3600, negotiation_speed=None, neg_n_steps=100, neg_time_limit=180, neg_step_time_limit=60, default_signing_delay=0, breach_processing=<BreachProcessing.VICTIM_THEN_PERPETRATOR: 1>, log_folder=None, log_to_file=True, log_ufuns=False, log_negotiations=False, log_step_time_limit=60, log_stats_every=0, log_file_level=10, log_screen_level=40, log_file_name='log.txt', mechanisms=None, awi_type='negmas.situated.AgentWorldInterface', start_negotiations_immediately=False, save_signed_contracts=True, save_cancelled_contracts=True, save_negotiations=True, save_resolved_breaches=True, ignore_agent_exceptions=False, ignore_contract_execution_exceptions=False, safe_stats_monitoring=False, name=None)

Bases: negmas.events.EventSink, negmas.events.EventSource, negmas.helpers.ConfigReader, abc.ABC

Base world class encapsulating a world that runs a simulation with several agents interacting within some dynamically changing environment.

A world maintains its own session.

Attributes Summary

cancelled_contracts

type List[Dict[str, Any]]

relative_time

Returns a number between 0 and 1 indicating elapsed relative time or steps.

remaining_steps

Returns the remaining number of steps until the end of the mechanism run.

remaining_time

Returns remaining time in seconds.

resolved_breaches

type List[Dict[str, Any]]

saved_breaches

type List[Dict[str, Any]]

saved_contracts

type List[Dict[str, Any]]

saved_negotiations

type List[Dict[str, Any]]

signed_contracts

type List[Dict[str, Any]]

stats

type Dict[str, Any]

time

Elapsed time since mechanism started in seconds.

Continued on next page
Table 104 – continued from previous page

| unresolved_breaches | rtype | List[Dict[str, Any]] |

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>announce(event)</code></td>
<td>Raises an event and informs all event sinks that are registered for notifications on this event type</td>
</tr>
<tr>
<td><code>append_stats()</code></td>
<td></td>
</tr>
<tr>
<td><code>execute(action, agent[, callback])</code></td>
<td>Executes the given action by the given agent</td>
</tr>
<tr>
<td><code>from_config(config[, section,...])</code></td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td><code>get_private_state(agent)</code></td>
<td>Reads the private state of the given agent</td>
</tr>
<tr>
<td><code>join(x[, simulation_priority])</code></td>
<td>Add an agent to the world.</td>
</tr>
<tr>
<td><code>logdebug(s)</code></td>
<td>logs debug-level information</td>
</tr>
<tr>
<td><code>logerror(s)</code></td>
<td>logs error-level information</td>
</tr>
<tr>
<td><code>loginfo(s)</code></td>
<td>logs info-level information</td>
</tr>
<tr>
<td><code>logwarning(s)</code></td>
<td>logs warning-level information</td>
</tr>
<tr>
<td><code>on_contract_cancelled(contract)</code></td>
<td>Called whenever a concluded contract is not signed (cancelled)</td>
</tr>
<tr>
<td><code>on_contract_concluded(contract, to_be_signed_at)</code></td>
<td>Called to add a contract to the existing set of contract after it is signed</td>
</tr>
<tr>
<td><code>on_contract_signed(contract)</code></td>
<td>Called to add a contract to the existing set of contract after it is signed</td>
</tr>
<tr>
<td><code>on_event(event, sender)</code></td>
<td></td>
</tr>
<tr>
<td><code>read_config(config[, section])</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td><code>register(x[, simulation_priority])</code></td>
<td>Registers an entity in the world so it can be looked up by name.</td>
</tr>
<tr>
<td><code>register_listener(event_type, listener)</code></td>
<td></td>
</tr>
<tr>
<td><code>register_stats_monitor(m)</code></td>
<td></td>
</tr>
<tr>
<td><code>register_world_monitor(m)</code></td>
<td></td>
</tr>
<tr>
<td><code>request_negotiation_about(req_id, caller,...)</code></td>
<td>Requests to start a negotiation with some other agents</td>
</tr>
<tr>
<td><code>run()</code></td>
<td>Runs the simulation until it ends</td>
</tr>
<tr>
<td><code>run_negotiation(caller, issues[, partners[, ...]])</code></td>
<td>Requests to start a negotiation with some other agents</td>
</tr>
<tr>
<td><code>save_config(file_name)</code></td>
<td>Saves the config of the world as a yaml file</td>
</tr>
<tr>
<td><code>set_bulletin_board(bulletin_board)</code></td>
<td></td>
</tr>
<tr>
<td><code>step()</code></td>
<td>A single simulation step</td>
</tr>
<tr>
<td><code>unregister_stats_monitor(m)</code></td>
<td></td>
</tr>
<tr>
<td><code>unregister_world_monitor(m)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Attributes Documentation

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Return type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cancelled_contracts</code></td>
<td>List[Dict[str, Any]]</td>
<td></td>
</tr>
<tr>
<td><code>relative_time</code></td>
<td>float</td>
<td>Returns a number between 0 and 1 indicating elapsed relative time or steps.</td>
</tr>
<tr>
<td><code>remaining_steps</code></td>
<td>float</td>
<td>Returns the remaining number of steps until the end of the mechanism run. None if unlimited</td>
</tr>
</tbody>
</table>
**Return type**  Optional[int]

**remaining_time**
Returns remaining time in seconds. None if no time limit is given.

**Return type**  Optional[float]

**resolved_breaches**

**Return type**  List[Dict[str, Any]]

**saved_breaches**

**Return type**  List[Dict[str, Any]]

**saved_contracts**

**Return type**  List[Dict[str, Any]]

**saved_negotiations**

**Return type**  List[Dict[str, Any]]

**signed_contracts**

**Return type**  List[Dict[str, Any]]

**stats**

**Return type**  Dict[Union[str, int], Any]

**time**
Elapsed time since mechanism started in seconds. None if the mechanism did not start running

**Return type**  Optional[float]

**unresolved_breaches**

**Return type**  List[Dict[str, Any]]

### Methods Documentation

**announce** *(event)*
Raises an event and informs all event sinks that are registered for notifications on this event type

**append_stats** ()

**execute** *(action, agent, callback=None)*
Executes the given action by the given agent

**Return type**  bool

**classmethod from_config** *(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)*
Creates an object of this class given the configuration info

**Parameters**

- **config** *(Union[str, dict])* – Either a file name or a dictionary
- **section** *(Optional[str])* – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return
- **try_parsing_children** *(bool)* – If true the children will first be parsed as ConfigReader classes if they are not
- **types** *(e.g. int, str, float, Iterable[int|str|float])* *(simple)* –
• **scope** – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are

• **any children that are to be parsed.** *(having)* –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing `scope=globals()` to this function is to get around the fact that in python `eval()` will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when `eval()` is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her `globals()` as the scope.

**get_private_state** *(agent)*

Reads the private state of the given agent

**Return type** `dict`

**join** *(x, simulation_priority=0)*

Add an agent to the world.

**Parameters**

• **x** *(Agent)* – The agent to be registered

• **simulation_priority** *(int)* – The simulation priority. Entities with lower priorities will be stepped first during

**Returns:**

**logdebug** *(s)*

logs debug-level information

**Parameters** **s** *(str)* – The string to log

**Return type** `None`

**logerror** *(s)*

logs error-level information

**Parameters** **s** *(str)* – The string to log

**Return type** `None`

**loginfo** *(s)*

logs info-level information

**Parameters** **s** *(str)* – The string to log

**Return type** `None`

**logwarning** *(s)*

logs warning-level information

**Parameters** **s** *(str)* – The string to log

**Return type** `None`

**on_contract_cancelled** *(contract)*

Called whenever a concluded contract is not signed (cancelled)

**Parameters** **contract** – The contract to add

Remarks:
• By default this function just adds the contract to the set of contracts maintained by the world.
• You should ALWAYS call this function when overriding it.

**on_contract_concluded** *(contract, to_be_signed_at)*
Called to add a contract to the existing set of contract after it is signed

**Parameters**
- **contract** *(Contract)* – The contract to add
- **to_be_signed_at** *(int)* – The timestep at which the contract is to be signed

**Remarks:**
- By default this function just adds the contract to the set of contracts maintained by the world.
- You should ALWAYS call this function when overriding it.

**Return type** None

**on_contract_signed** *(contract)*
Called to add a contract to the existing set of contract after it is signed

**Parameters**
- **contract** *(Contract)* – The contract to add

**Remarks:**
- By default this function just adds the contract to the set of contracts maintained by the world.
- You should ALWAYS call this function when overriding it.

**Return type** None

**on_event** *(event, sender)*

**classmethod read_config** *(config, section=None)*
Reads the configuration from a file or a dict and prepares it for parsing

**Parameters**
- **config** *(Union[Union[str, dict]])* – Either a file name or a dictionary
- **section** *(Optional[str])* – A section in the file or a key in the dictionary to use for loading params

**Return type** Dict[Dict][str, Any]

**Returns** A dict ready to be parsed by from_config

**Remarks:**

**register** *(x, simulation_priority=0)*
Registers an entity in the world so it can be looked up by name. Should not be called directly

**Parameters**
- **x** *(Entity)* – The entity to be registered
- **simulation_priority** *(int)* – The simulation priority. Entities with lower priorities will be stepped first during

**Returns:**

**register_listener** *(event_type, listener)*

**register_stats_monitor** *(m)*

**register_world_monitor** *(m)*
requests a negotiation about

\( \text{request_negotiation_about}(\text{req_id, caller, issues, partners, roles=None, annotation=None, mechanism_name=None, mechanism_params=None}) \)

Requests to start a negotiation with some other agents

**Parameters**

- **req_id** (str) – An ID for the request that is unique to the caller
- **caller** (Agent) – The agent requesting the negotiation
- **partners** (List[Agent]) – The list of partners that the agent wants to negotiate with. Roles will be determined by these agents.
- **issues** (List[Issue]) – Negotiation issues
- **annotation** (Optional[Dict[str, Any]]) – Extra information to be passed to the partners when asking them to join the negotiation
- **partners** – A list of partners to participate in the negotiation
- **roles** (Optional[List[str]]) – The roles of different partners. If None then each role for each partner will be None
- **mechanism_name** (Optional[str]) – Name of the mechanism to use. It must be one of the mechanism_names that are supported by the
- **or None which means that the World should select the mechanism. If None, then roles and my_role(World) –
- **also be None(must) –
- **mechanism_params** (Optional[Dict[str, Any]]) – A dict of parameters used to initialize the mechanism object

**Return type** bool

**Returns** None. The caller will be informed by a callback function `on_neg_request_accepted` or `on_neg_request_rejected` about the status of the negotiation.

**run()**

Runs the simulation until it ends

**run_negotiation** (caller, issues, partners, roles=None, annotation=None, mechanism_name=None, mechanism_params=None)

Requests to start a negotiation with some other agents

**Parameters**

- **caller** (Agent) – The agent requesting the negotiation
- **partners** (Collection[Agent]) – The list of partners that the agent wants to negotiate with. Roles will be determined by these agents.
- **issues** (Collection[Issue]) – Negotiation issues
- **annotation** (Optional[Dict[str, Any]]) – Extra information to be passed to the partners when asking them to join the negotiation
- **partners** – A list of partners to participate in the negotiation
- **roles** (Optional[Collection[str]]) – The roles of different partners. If None then each role for each partner will be None
- **mechanism_name** (Optional[str]) – Name of the mechanism to use. It must be one of the mechanism_names that are supported by the
- **or None which means that the World should select the mechanism. If None, then roles and my_role(World) –
• also be None (must) –
• mechanism_params (Optional[Dict[str, Any]]) – A dict of parameters used
to initialize the mechanism object

Returns The agreed upon contract if negotiation was successful otherwise, None.

Return type Contract

save_config (file_name)
Saves the config of the world as a yaml file

Parameters file_name (str) – Name of file to save the config to

Returns:

set_bulletin_board (bulletin_board)

step ()
A single simulation step

Return type bool

unregister_stats_monitor (m)

unregister_world_monitor (m)

Entity
class negmas.situated.Entity (name=None)
Bases: negmas.common.NamedObject

Defines an entity that is a part of the world but does not participate in the simulation

Attributes Summary

<table>
<thead>
<tr>
<th>id</th>
<th>The unique ID of this entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

create(*args, **kwargs) Creates an object and returns a proxy to it.

Attributes Documentation

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

short_type_name
Returns a short name of the type of this entity

type_name
Returns the name of the type of this entity

uuid
The unique ID of this entity

**Methods Documentation**

```python
classmethod create(*args, **kwargs)
```

Creates an object and returns a proxy to it.

**AgentWorldInterface**

```python
class negmas.situated.AgentWorldInterface(world, agent)
```

Agent World Interface class

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_step</td>
<td>Current simulation step</td>
</tr>
<tr>
<td>default_signing_delay</td>
<td>Return type int</td>
</tr>
<tr>
<td>n_steps</td>
<td>Number of steps in a simulation</td>
</tr>
<tr>
<td>relative_time</td>
<td>Relative time of the simulation going from 0 to 1</td>
</tr>
<tr>
<td>state</td>
<td>Returns the private state of the agent in that world</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb_query(section, query[, query_keys])</td>
<td>Returns all records in the given section/sections of the bulletin-board that satisfy the query</td>
</tr>
<tr>
<td>bb_read(section, key)</td>
<td>Reads the value associated with given key from the bulletin board</td>
</tr>
<tr>
<td>bb_record(section, value[, key])</td>
<td>Records data in the given section of the bulletin board</td>
</tr>
<tr>
<td>bb_remove(section, *[query, key, ...])</td>
<td>Removes a value or a set of values from the bulletin Board</td>
</tr>
<tr>
<td>execute(action[, callback])</td>
<td>Executes an action in the world simulation</td>
</tr>
<tr>
<td>logdebug(msg)</td>
<td>Logs a WARNING message</td>
</tr>
<tr>
<td>logerror(msg)</td>
<td>Logs a WARNING message</td>
</tr>
<tr>
<td>loginfo(msg)</td>
<td>Logs an INFO message</td>
</tr>
<tr>
<td>logwarning(msg)</td>
<td>Logs a WARNING message</td>
</tr>
<tr>
<td>request_negotiation_about(issues, partners, ...)</td>
<td>Requests to start a negotiation with some other agents</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_step</td>
<td>Current simulation step</td>
</tr>
<tr>
<td>Return type</td>
<td>int</td>
</tr>
<tr>
<td>default_signing_delay</td>
<td>Return type int</td>
</tr>
<tr>
<td>n_steps</td>
<td>Number of steps in a simulation</td>
</tr>
</tbody>
</table>
Return type int
def relative_time:
    Relative time of the simulation going from 0 to 1
    Return type float
def state:
    Returns the private state of the agent in that world
    Return type Any

Methods Documentation
def bb_query(section, query, query_keys=False):
    Returns all records in the given section/sections of the bulletin-board that satisfy the query
    Parameters
        • section (Union[str, List[str], None]) – Either a section name, a list of sections or None specifying ALL public sections (see remarks)
        • query (Any) – The query which is USUALLY a dict with conditions on it when querying values and a RegExp when
        • keys (querying) –
        • query_keys – Whether the query is to be applied to the keys or values.
    Returns value pairs giving all records that satisfied the given requirements.
    Return type • A dictionary with key

Remarks:
• A public section is a section with a name that does not start with an underscore
• If a set of sections is given, and two records in different sections had the same key, only one of them will be returned
• Key queries use regular expressions and match from the beginning using the standard re.match function

def bb_read(section, key):
    Reads the value associated with given key from the bulletin board
    Parameters
        • section (str) – section name
        • key (str) – key
    Return type Optional[Any]
    Returns Content of that key in the bulletin-board

def bb_record(section, value, key=None):
    Records data in the given section of the bulletin board
    Parameters
        • section (str) – section name (can contain subsections separated by ‘/’)
        • key (Optional[None]) – The key
        • value (Any) – The value
    Return type None
**bb_remove** (*section*, *, query=None, key=None, query_keys=False, value=None)

Removes a value or a set of values from the bulletin board.

**Parameters**

- **section** (*Union[str, List[str], None]*) – The section
- **query** (*Optional[Any]*) – The query to use to select what to remove
- **key** (*Optional[str]*) – The key to remove (no need to give a full query)
- **query_keys** (*bool*) – Whether to apply the query (if given) to keys or values
- **value** (*Optional[Any]*) – Value to be removed

**Returns** Success or failure

**Return type** *bool*

**execute** (*action*, *callback=None*)

Executes an action in the world simulation.

**Return type** *bool*

**logdebug** (*msg*)

Logs a warning message.

**Parameters**

- **msg** (*str*) – The message to log

**Returns**:

**Return type** *None*

**logerror** (*msg*)

Logs a warning message.

**Parameters**

- **msg** (*str*) – The message to log

**Returns**:

**Return type** *None*

**loginfo** (*msg*)

Logs an info message.

**Parameters**

- **msg** (*str*) – The message to log

**Returns**:

**Return type** *None*

**logwarning** (*msg*)

Logs a warning message.

**Parameters**

- **msg** (*str*) – The message to log

**Returns**:

**Return type** *None*

**request_negotiation_about** (*issues*, *partners*, *req_id*, *roles=None*, *annotation=None*, *mechanism_name=None*, *mechanism_params=None*)

Requests to start a negotiation with some other agents.

**Parameters**

- **req_id** (*str*) –
- **issues** (*List[Issue]*) – Negotiation issues
- **annotation** (*Optional[Dict[str, Any]]*) – Extra information to be passed to the partners when asking them to join the negotiation
- **partners** (*List[str]*) – A list of partners to participate in the negotiation
• **roles** *(Optional[List[str]])* – The roles of different partners. If None then each role for each partner will be None

• **mechanism_name** *(Optional[str]*) – Name of the mechanism to use. It must be one of the mechanism_names that are supported by the

• **or** None which means that the World should select the mechanism. If None, then roles and my_role *(World)* –

• **also be None** *(must)* –

• **mechanism_params** *(Optional[Dict[str, Any]])* – A dict of parameters used to initialize the mechanism object

**Return type**  bool

**Returns**  List[“Agent”] the list of partners who rejected the negotiation if any. If None then the negotiation was accepted. If empty then the negotiation was not started from the world manager

Remarks:

• The function will create a request ID that will be used in callbacks
  ```python
  on_neg_request_accepted
  and
  on_neg_request_rejected
  ```

### NegotiationInfo

**class**  negmas.situated.NegotiationInfo *(mechanism, partners, annotation, issues, requested_at, rejectors=None)*

**Bases:**  object

Saves information about a negotiation

**Attributes Summary**

<table>
<thead>
<tr>
<th><strong>rejectors</strong></th>
</tr>
</thead>
</table>

**Attributes Documentation**

rejectors  =  None

### RenegotiationRequest

**class**  negmas.situated.RenegotiationRequest *(publisher, issues, annotation=<factory>)*

**Bases:**  object

A request for renegotiation.

The issues can be any or all of the following:

immediate_delivery: int immediate_unit_price: float later_quantity: int later_unit_price: int later_penalty: float later_time: int

### StatsMonitor

**class**  negmas.situated.StatsMonitor *(name=None)*

**Bases:**  negmas.situated.Entity

A monitor object capable of receiving stats of a world

5.8. negmas.situated Module
Attributes Summary

<table>
<thead>
<tr>
<th>attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create(*args,**kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>init(stats,world_name)</td>
<td>Called to initialize the monitor before running first step</td>
</tr>
<tr>
<td>step(stats,world_name)</td>
<td>Called at the END of every simulation step</td>
</tr>
</tbody>
</table>

Attributes Documentation

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**short_type_name**

Returns a short name of the type of this entity

**type_name**

Returns the name of the type of this entity

**uuid**

The unique ID of this entity

Methods Documentation

**classmethod create(*args,**kwargs)***

Creates an object and returns a proxy to it.

**init(stats,world_name)**

Called to initialize the monitor before running first step

**step(stats,world_name)**

Called at the END of every simulation step

WorldMonitor

**class negmas.situated.WorldMonitor(name=None)**

Bases: negmas.situated.Entity

A monitor object capable of monitoring a world. It has read/write access to the world

Attributes Summary

<table>
<thead>
<tr>
<th>attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Continued on next page
### Attributes Documentation

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>id</code></td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td><code>short_type_name</code></td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td><code>type_name</code></td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td><code>uuid</code></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Documentation

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create(*args, **kwargs)</code></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>init(world)</code></td>
<td>Called to initialize the monitor before running first step</td>
</tr>
<tr>
<td><code>step(world)</code></td>
<td>Called at the END of every simulation step</td>
</tr>
</tbody>
</table>

```python
classmethod create(*args, **kwargs)
    Creates an object and returns a proxy to it.

init(world)
    Called to initialize the monitor before running first step

step(world)
    Called at the END of every simulation step
```
5.9 negmas.events Module

Implements Event management

5.9.1 Classes

<table>
<thead>
<tr>
<th>Event(type, data)</th>
<th>EventSource()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An object capable of raising events</td>
</tr>
</tbody>
</table>

Continued on next page
Table 115 – continued from previous page

<table>
<thead>
<tr>
<th>EventSink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notification(type, data)</td>
</tr>
<tr>
<td>Notifier([name])</td>
</tr>
</tbody>
</table>

**Event**

class negmas.events.Event (type, data)

Bases: object

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>

**EventSource**

class negmas.events.EventSource

Bases: object

An object capable of raising events

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>announce(event)</td>
<td>Raises an event and informs all event sinks that are registered for notifications on this event type</td>
</tr>
<tr>
<td>register_listener(event_type, listener)</td>
<td></td>
</tr>
</tbody>
</table>

**Methods Documentation**

**announce (event)**

Raises an event and informs all event sinks that are registered for notifications on this event type

**register_listener (event_type, listener)**

**EventSink**

class negmas.events.EventSink

Bases: object

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_event(event, sender)</td>
<td></td>
</tr>
</tbody>
</table>
Methods Documentation

on_event(event, sender)

Notification

class negmas.events.Notification(type, data)
    Bases: object

Attributes Summary

data

type

Attributes Documentation

data

type

Notifier

class negmas.events.Notifier(name=None)
    Bases: negmas.common.NamedObject

Attributes Summary

id The unique ID of this entity

name A convenient name of the entity (intended primarily for printing/logging/debugging).

uuid The unique ID of this entity

Methods Summary

create(*args, **kwargs) Creates an object and returns a proxy to it.

notify(notifiable, notification)

Attributes Documentation

id The unique ID of this entity

name A convenient name of the entity (intended primarily for printing/logging/debugging).

uuid The unique ID of this entity

Methods Documentation

classmethod create(*args, **kwargs) Creates an object and returns a proxy to it.
notify (notifiable, notification)

Notifiable

class negmas.events.Notifiable
   Bases: object

   Methods Summary

   on_notification(notification, notifier)  
      rtype  None

   Methods Documentation

   on_notification (notification, notifier) 
      Return type  None
5.9.2 Class Inheritance Diagram

Notification

Notifiable

NamedObject → Notifier

EventSource

EventSink

Event
This part of the documentation describes specific application modules that are based on the base situated module.

6.1 negmas.apps.scml Package

The implementation file for all entities needed for ANAC-SCML 2019.
Participants need to provide a class inherited from FactoryManager that implements all of its abstract functions.
Participants can optionally override any other methods of this class or implement new NegotiatorUtility class.

6.1.1 Simulation steps:

1. prepare custom stats (call _pre_step_stats)
2. sign contracts that are to be signed at this step calling on_contract_signed as needed
3. step all existing negotiations negotiation_speed_multiple times handling any failed negotiations and creating contracts for any resulting agreements
4. run all Entity objects registered (i.e. all agents), Consumers run first then FactoryManager s then Miners
5. execute contracts that are executable at this time-step handling any breaches
6. Custom Simulation Steps:
   1. step all factories (Factory objects) running any pre-scheduled commands
   2. Apply interests and pay loans
   3. remove expired CFP s
   4. Deliver any products that are in transportation
7. remove any negotiations that are completed!
8. update basic stats
9. update custom stats (call `_post_step_stats`)

6.1.2 Remarks about re-negotiation on breaches:

- The victim is asked first to specify the negotiation agenda (issues) then the perpetrator
- renegotiations for breaches run immediately to completion independent from settings of `negotiation_speed_multiplier` and `immediate_negotiations`. That include conclusion and signing of any resulting agreements.

6.1.3 Remarks about timing:

- The order of events within a single time-step are as follows:
  1. Contracts scheduled to be signed are signed.
  2. Scheduled negotiations run for the predefined number of steps. Any negotiation that result in a contract or fail may trigger other negotiations.
  3. If `immediate_negotiations`, some of the newly added negotiations may be concluded/failed.
  4. Any newly concluded contracts that are to be signed at this step are signed.
  5. Contracts are executed including full execution of any re-negotiations and breaches are handled. Notice that if re-negotiation leads to new contracts, these will be concluded and signed immediately at this step. Please note the following about contract execution:
     - Products are moved from the seller’s storage to a temporary truck as long as they are available at the time of contract execution. Because contract execution happens before actual production, outputs from production processes CANNOT be sold at the same time-step.
  6. Production is executed on all factories. For a Process to start/continue on a Line, all its inputs required at this time-step MUST be available in storage of the corresponding factory by this point. This implies that it is impossible for any processes to start at time-step 0 except if initial storage was nonzero. FactoryManagers are informed about processes that cannot start due to storage or fund shortage (or cannot continue due to storage shortage) through an `on_production_failure` call.
  7. Outputs of the Process are generated at the end of the corresponding time-step. It is immediately moved to storage. Because outputs are generated at the end of the step and inputs are consumed at the beginning, a factory cannot use outputs of a process as inputs to another process that starts at the same time-step.
  8. Products are moved from the temporary truck to the buyer’s storage after the transportation_delay have passed at the end of the time-step. Transportation completes at the end of the time-step no matter what is the value for transportation_delay. This means that if a FactoryManager believes that it can produce some product at time \( t \), it should never contract to sell it before \( t+d+1 \) where \( d \) is the transportation_delay (the \( I \) comes from the fact that contract execution happens before production). Even for a zero transportation delay, you cannot produce something and sell it in the same time-step. Moreover, the buyer should never use the product to be delivered at time \( t \) as an input to a production process that needs it before step \( t+1 \).
  9. When contracts are executed, the funds are deducted from the buyer’s wallet at the beginning of the simulation step and deposited in the seller’s wallet at the end of that step (similar to what happens to the products). This means that a factory manager cannot use funds it receives from sales at time \( t \) for buying products before \( t+1 \).

6.1.4 Remarks about ANAC 2019 SCML League:

Given the information above, and settings for the ANAC 2019 SCML you can confirm for yourself that the following rules are all correct:

1. No agents except miners should contract on delivery at time 0.
2. *FactoryManager* s should never sign contracts to sell the output of their production with delivery at \( t \) except if this production starts at step \( t \) and the contract is signed no later than \( t-1 \).

3. If not all inputs are available in storage, *FactoryManager* s should never sign contracts to sell the output of production with delivery at \( t \) later than \( t-2 \) (and that is optimistic).

### 6.1.5 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>transaction(simulator)</code></td>
<td>Runs the simulated actions then confirms them if they are not rolled back</td>
</tr>
<tr>
<td><code>temporary_transaction(simulator)</code></td>
<td>Runs the simulated actions then rolls them back</td>
</tr>
<tr>
<td><code>anac2019_world((competitors, params, ...))</code></td>
<td>Creates a world compatible with the ANAC 2019 competition.</td>
</tr>
<tr>
<td><code>anac2019_tournament(competitors[, ...])</code></td>
<td>The function used to run ANAC 2019 SCML tournament (collusion track).</td>
</tr>
<tr>
<td><code>anac2019_collusion(competitors[, ...])</code></td>
<td>The function used to run ANAC 2019 SCML tournament (collusion track).</td>
</tr>
<tr>
<td><code>anac2019_std(competitors[, ...])</code></td>
<td>The function used to run ANAC 2019 SCML tournament (standard track).</td>
</tr>
<tr>
<td><code>balance_calculator(worlds, scoring_context, ...)</code></td>
<td>A scoring function that scores factory managers’ performance by the final balance only ignoring whatever still in their inventory.</td>
</tr>
<tr>
<td><code>anac2019_sabotage(competitors[, ...])</code></td>
<td>The function used to run ANAC 2019 SCML tournament (collusion track).</td>
</tr>
<tr>
<td><code>pos_gauss(mu, sigma)</code></td>
<td>Returns a sample from a rectified gaussian</td>
</tr>
<tr>
<td><code>_safe_max(a, b)</code></td>
<td></td>
</tr>
<tr>
<td><code>zero_runs(a)</code></td>
<td>Finds all runs of zero in an array</td>
</tr>
</tbody>
</table>

### `transaction`

`negmas.apps.scml.transaction(simulator)`

Runs the simulated actions then confirms them if they are not rolled back

### `temporary_transaction`

`negmas.apps.scml.temporary_transaction(simulator)`

Runs the simulated actions then rolls them back
anac2019_world

negmas.apps.scml.anac2019_world(competitors=(), params=(), randomize=True, log_file_name=None, name=None, agent_names_reveal_type=False, n_intermediate=(1, 4), n_miners=5, n_factories_per_level=11, n_agents_per_competitor=1, n_consumers=5, n_lines_per_factory=10, guaranteed_contracts=False, use_consumer=True, max_insurance_premium=inf, n_retrials=5, negotiator_type='negmas.sao.AspirationNegotiator', transportation_delay=0, default_signing_delay=0, max_storage=922372036854775807, consumption_horizon=15, consumption=(3, 5), negotiation_speed=21, neg_time_limit=240, neg_n_steps=20, n_steps=100, time_limit=8100, n_default_per_level=5, compact=False, **kwargs)

Creates a world compatible with the ANAC 2019 competition. Note that

Parameters

- **n_agents_per_competitor** – Number of instantiations of each competing type.
- **name** *(Optional[str]*) – World name to use
- **agent_names_reveal_type** *(bool)* – If true, a snake_case version of the agent_type will prefix agent names
- **randomize** *(bool)* – If true, managers are assigned to factories randomly otherwise in the order
- **are giving** *(they)* –
- **n_intermediate** *(Tuple[int, int])* –
- **n_default_per_level** *(int)* –
- **competitors** *(Sequence[Union[str, Type[FactoryManager]]])* – A list of class names for the competitors
- **params** *(Sequence[Dict[str, Any]])* – A list of dictionaries giving parameters to pass to the competitors
- **n_miners** – number of miners of the single raw material
- **n_factories_per_level** – number of factories at every production level
- **n_consumers** – number of consumers of the final product
- **n_steps** – number of simulation steps
- **n_lines_per_factory** – number of lines in each factory
- **negotiation_speed** – The number of negotiation steps per simulation step. None means infinite
- **default_signing_delay** – The number of simulation between contract conclusion and signature
- **neg_n_steps** – The maximum number of steps of a single negotiation (that is double the number of rounds)
- **neg_time_limit** – The total time-limit of a single negotiation
- **time_limit** – The total time-limit of the simulation
- **transportation_delay** – The transportation delay
• **n_retrials** – The number of retrials the *Miner* and *GreedyFactoryManager* will try if negotiations fail

• **max_insurance_premium** – The maximum insurance premium accepted by *GreedyFactoryManager*(-1 to disable)

• **use_consumer** – If true, the *GreedyFactoryManager* will use an internal consumer for buying its needs

• **guaranteed_contracts** – If true, the *GreedyFactoryManager* will only sign contracts that it can guarantee not to break.

• **consumption_horizon** – The number of steps for which *Consumer* publishes CFPs

• **consumption** – The consumption schedule will be sampled from a uniform distribution with these limits inclusive

• **log_file_name** *(Optional*[str]*) – File name to store the logs

• **negotiator_type** *(str)* – The negotiation factory used to create all negotiators

• **max_storage** – maximum storage capacity for all factory managers If None then it is unlimited

• **compact** *(bool)* – If True, then compact logs will be created to reduce memory footprint

• **kwargs** – key-value pairs to be passed as argument to chain_world() and then to SCMLWorld()

Return type **SCMLWorld**

Returns **SCMLWorld** ready to run

Remarks:

• Every production level n has one process only that takes n steps to complete

**anac2019_tournament**

negmas.apps.scml.anac2019_tournament *(competitors, agent_names_reveal_type=False, n_configs=5, max_worlds_per_config=1000, n_runs_per_world=5, n_agents_per_competitor=5, tournament_path='./logs/tournaments', total_timeout=None, parallelism='parallel', scheduler_ip=None, scheduler_port=None, tournament_progress_callback=None, world_progress_callback=None, name=None, verbose=False, configs_only=False, compact=False, **kwargs)*

The function used to run ANAC 2019 SCML tournament (collusion track).

Parameters

• **name** *(Optional*[str]*) – Tournament name

• **competitors** *(Sequence[Union*[str, Type*[FactoryManager]*]])* – A list of class names for the competitors

• **agent_names_reveal_type** – If true then the type of an agent should be readable in its name (most likely at its beginning).

• **n_configs** *(int)* – The number of different world configs (up to competitor assignment) to be generated.
• **max_worlds_per_config** *(int)* – The maximum number of worlds to run per config. If None, then all possible assignments of competitors within each config will be tried (all permutations).

• **n_runs_per_world** *(int)* – Number of runs per world. All of these world runs will have identical competitor assignment and identical world configuration.

• **n_agents_per_competitor** *(int)* – Number of agents per competitor

• **total_timeout** *(Optional[int]*) – Total timeout for the complete process

• **tournament_path** *(str)* – Path at which to store all results. A scores.csv file will keep the scores and logs folder will keep detailed logs

• **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed

• **scheduler_port** *(Optional[str]*) – Port of the dask scheduler if parallelism is dask, dist, or distributed

• **scheduler_ip** *(Optional[str]*) – IP Address of the dask scheduler if parallelism is dask, dist, or distributed

• **world_progress_callback** *(Optional[Callable[[Optional[SCMLWorld], None]]]*) – A function to be called after every step of every world run (only allowed for serial evaluation and should be used with caution).

• **tournament_progress_callback** *(Optional[Callable[[Optional[WorldRunResults]], None]])* – A function to be called with WorldRunResults after each world finished processing

• **verbose** *(bool)* – Verbosity

• **configs_only** – If true, a config file for each process

• **compact** – If true, effort will be made to reduce memory footprint including disabling most logs

• **kwargs** – Arguments to pass to the world_generator function

**Return type** `Union[TournamentResults, PathLike]`

**Returns** `TournamentResults` The results of the tournament or a PathLike giving the location where configs were saved

**Remarks:**

Default parameters will be used in the league with the exception of parallelism which may use distributed processing
The function used to run ANAC 2019 SCML tournament (collusion track).

**Parameters**

- **name** *(Optional[str])* – Tournament name
- **competitors** *(Sequence[Union[str, Type[FactoryManager]]])* – A list of class names for the competitors
- **competitor_params** *(Optional[Sequence[Dict[str, Any]]])* – A list of competitor parameters (used to initialize the competitors).
- **agent_names_reveal_type** – If true then the type of an agent should be readable in its name (most likely at its beginning).
- **n_configs** *(int)* – The number of different world configs (up to competitor assignment) to be generated.
- **max_worlds_per_config** *(Optional[int])* – The maximum number of worlds to run per config. If None, then all possible assignments of competitors within each config will be tried (all permutations).
- **n_runs_per_world** *(int)* – Number of runs per world. All of these world runs will have identical competitor assignment and identical world configuration.
- **n_agents_per_competitor** *(int)* – Number of agents per competitor
- **min_factories_per_level** *(int)* – Minimum number of factories for each production level
- **total_timeout** *(Optional[int])* – Total timeout for the complete process
- **tournament_path** *(str)* – Path at which to store all results. A scores.csv file will keep the scores and logs folder will keep detailed logs
- **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed
- **scheduler_port** *(Optional[str])* – Port of the dask scheduler if parallelism is dask, dist, or distributed
- **scheduler_ip** *(Optional[str])* – IP Address of the dask scheduler if parallelism is dask, dist, or distributed
- **world_progress_callback** *(Optional[Callable[[Optional[SCMLWorld]], None]])* – A function to be called after every step of every world run (only allowed for serial evaluation and should be used with caution).
- **tournament_progress_callback** *(Optional[Callable[[Optional[WorldRunResults]], None]])* – A function to be called with WorldRunResults after each world finished processing
• non_competitors (Optional[Sequence[Union[str, Type.FactoryManager]]]) – A list of agent types that will not be competing in the sabotage competition but will exist in the world

• non_competitor_params (Optional[Sequence[Union[str, Type.FactoryManager]]]) – parameters of non competitor agents

• verbose (bool) – Verbosity

• configs_only – If true, a config file for each

• compact – If true, compact logs will be created and effort will be made to reduce the memory footprint

• kwargs – Arguments to pass to the world_generator function

Return type Union[TournamentResults, PathLike]

Returns TournamentResults The results of the tournament or a PathLike giving the location where configs were saved

Remarks:
Default parameters will be used in the league with the exception of parallelism which may use distributed processing

anac2019_std

negmas.apps.scml.anac2019_std(competitors, competitor_params=None, agent_names_reveal_type=False, n_configs=5, max_worlds_per_config=1000, n_runs_per_world=5, min_factories_per_level=5, tournament_path='./logs/tournaments', total_timeout=None, parallelism='parallel', scheduler_ip=None, scheduler_port=None, tournament_progress_callback=None, world_progress_callback=None, non_competitors=None, non_competitor_params=None, name=None, verbose=False, configs_only=False, compact=False, **kwargs)

The function used to run ANAC 2019 SCML tournament (standard track).

Parameters

• name (Optional[str]) – Tournament name

• competitors (Sequence[Union[str, Type.FactoryManager]]) – A list of class names for the competitors

• competitor_params (Optional[Sequence[Dict[str, Any]]]) – A list of competitor parameters (used to initialize the competitors).

• agent_names_reveal_type – If true then the type of an agent should be readable in its name (most likely at its beginning).

• n_configs (int) – The number of different world configs (up to competitor assignment) to be generated.

• max_worlds_per_config (Optional[int]) – The maximum number of worlds to run per config. If None, then all possible assignments of competitors within each config will be tried (all permutations).

• n_runs_per_world (int) – Number of runs per world. All of these world runs will have identical competitor assignment and identical world configuration.

• min_factories_per_level (int) – Minimum number of factories for each production level

• total_timeout (Optional[int]) – Total timeout for the complete process
• **tournament_path** *(str)* – Path at which to store all results. A scores.csv file will keep the scores and logs folder will keep detailed logs.

• **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed.

• **scheduler_port** *(Optional[str]*) – Port of the dask scheduler if parallelism is dask, dist, or distributed.

• **scheduler_ip** *(Optional[str]*) – IP Address of the dask scheduler if parallelism is dask, dist, or distributed.

• **world_progress_callback** *(Optional[Callable[[Optional[SCMLWorld]], None]])* – A function to be called after every step of every world run (only allowed for serial evaluation and should be used with caution).

• **tournament_progress_callback** *(Optional[Callable[[Optional[WorldRunResults]], None]])* – A function to be called with `WorldRunResults` after each world finished processing.

• **non_competitors** *(Optional[Sequence[Union[str, Type[FactoryManager]]]])* – A list of agent types that will not be competing in the sabotage competition but will exist in the world.

• **non_competitor_params** *(Optional[Sequence[Union[str, Type[FactoryManager]]]])* – Parameters of non competitor agents.

• **verbose** *(bool)* – Verbosity

• **configs_only** – If true, a config file for each

• **compact** – If true, compact logs will be created and effort will be made to reduce the memory footprint

• **kwarg** – Arguments to pass to the `world_generator` function

**Return type** `Union[TournamentResults, PathLike]`

**Returns** `TournamentResults` The results of the tournament or a `PathLike` giving the location where configs were saved.

**Remarks:**

Default parameters will be used in the league with the exception of parallelism which may use distributed processing.

balance_calculator

`negmas.apps.scml.balance_calculator(worlds, scoring_context, dry_run, ignore_default=True)`

A scoring function that scores factory managers’ performance by the final balance only ignoring whatever still in their inventory.

**Parameters**

• **worlds** *(List[SCMLWorld])* – The world which is assumed to be run up to the point at which the scores are to be calculated.

• **scoring_context** *(Dict[str, Any])* – A dict of context parameters passed by the world generator or assigner.

• **dry_run** *(bool)* – A boolean specifying whether this is a dry_run. For dry runs, only names and types are expected in the returned `WorldRunResults`

**Return type** `WorldRunResults`

**Returns** `WorldRunResults` giving the names, scores, and types of factory managers.
anac2019_sabotage

def anac2019_sabotage(competitors, competitor_params=None, agent_names_reveal_type=False, n_configs=5, max_worlds_per_config=1000, n_runs_per_world=5, n_agents_per_competitor=5, min_factories_per_level=5, tournament_path='./logs/tournaments', total_timeout=None, parallelism='parallel', scheduler_ip=None, scheduler_port=None, tournament_progress_callback=None, world_progress_callback=None, non_competitors=None, non_competitor_params=None, name=None, verbose=False, configs_only=False, compact=False, **kwargs)

The function used to run ANAC 2019 SCML tournament (collusion track).

Parameters

- **name** *(Optional [str])* – Tournament name
- **competitors** *(Sequence[Union[typing.Type[FactoryManager]]])* – A list of class names for the competitors
- **competitor_params** *(Optional[Sequence[Dict[str, Any]]])* – A list of competitor parameters (used to initialize the competitors).
- **agent_names_reveal_type** – If true then the type of an agent should be readable in its name (most likely at its beginning).
- **n_configs** *(int)* – The number of different world configs (up to competitor assignment) to be generated.
- **max_worlds_per_config** *(Optional[int])* – The maximum number of worlds to run per config. If None, then all possible assignments of competitors within each config will be tried (all permutations).
- **n_runs_per_world** *(int)* – Number of runs per world. All of these world runs will have identical competitor assignment and identical world configuration.
- **n_agents_per_competitor** *(int)* – Number of agents per competitor
- **min_factories_per_level** *(int)* – Minimum number of factories for each production level
- **total_timeout** *(Optional[int])* – Total timeout for the complete process
- **tournament_path** *(str)* – Path at which to store all results. A scores.csv file will keep the scores and logs folder will keep detailed logs
- **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed
- **scheduler_port** *(Optional[str])* – Port of the dask scheduler if parallelism is dask, dist, or distributed
- **scheduler_ip** *(Optional[str])* – IP Address of the dask scheduler if parallelism is dask, dist, or distributed
- **world_progress_callback** *(Optional[Callable[[Optional[SCMLWorld]], None]])* – A function to be called after every step of every world run (only allowed for serial evaluation and should be used with caution).
- **tournament_progress_callback** *(Optional[Callable[[Optional[WorldRunResults]], None]])* – A function to be called with WorldRunResults after each world finished processing
• **non_competitors** (Optional[Sequence[Union[str, Type[FactoryManager]]]]) – A list of agent types that will not be competing in the sabotage competition but will exist in the world

• **non_competitor_params** (Optional[Sequence[Union[str, Type[FactoryManager]]]]) – parameters of non competitor agents

• **verbose** (bool) – Verbosity

• **configs_only** – If true, a config file for each

• **compact** – If true, compact logs will be created and effort will be made to reduce the memory footprint

• **kwargs** – Arguments to pass to the world_generator function

**Return type** Union[TournamentResults, PathLike]

**Returns** TournamentResults The results of the tournament or a PathLike giving the location where configs were saved

**Remarks:**

Default parameters will be used in the league with the exception of parallelism which may use distributed processing

**pos_gauss**

negmas.apps.scml.pos_gauss(mu, sigma)

Returns a sample from a rectified gaussian

**_safe_max**

negmas.apps.scml._safe_max(a, b)

**zero_runs**

negmas.apps.scml.zero_runs(a)

Finds all runs of zero in an array

**Parameters**

a (<built-in function array>) – Input array (assumed to be 1D)

**Returns**

A 2D array giving beginning and end (exclusive) of zero stretches in the input array.

**Return type** np.array

6.1.6 Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product(id, production_level, name, ...)</td>
<td>An input/output to a production process</td>
</tr>
<tr>
<td>Process(id, production_level, name, inputs, ...)</td>
<td>An input/output to a production process</td>
</tr>
<tr>
<td>InputOutput(product, quantity, step)</td>
<td>An input/output to a production process</td>
</tr>
<tr>
<td>RunningCommandInfo(profile, beg, end, step, ...)</td>
<td>An input/output to a production process</td>
</tr>
<tr>
<td>ManufacturingProfile(n_steps, cost, ...)</td>
<td>The costs/time required for running a process on a line (with associated cancellation costs etc).</td>
</tr>
<tr>
<td>ManufacturingProfileCompiled(n_steps, cost, ...)</td>
<td>The costs/time required for running a process on a line (with associated cancellation costs etc).</td>
</tr>
<tr>
<td>ProductManufacturingInfo(profile, quantity, step)</td>
<td>Gives full information about a manufacturing process that can generate or consume a product.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FactoryStatusUpdate</td>
<td>(balance, storage) Describes a job to be run on one production line of a Factory.</td>
</tr>
<tr>
<td>Job</td>
<td>(profile, time, line, action, contract, ...) Describes a job to be run on one production line of a Factory.</td>
</tr>
<tr>
<td>ProductionNeed</td>
<td>(product, needed_for, ...) Describes some quantity of a product that is needed to honor a (sell) contract.</td>
</tr>
<tr>
<td>MissingInput</td>
<td>(product, quantity)</td>
</tr>
<tr>
<td>ProductionReport</td>
<td>(line, started, continuing, ...)</td>
</tr>
<tr>
<td>ProductionFailure</td>
<td>(line, command, ...)</td>
</tr>
<tr>
<td>FinancialReport</td>
<td>(agent, step, cash, ...)</td>
</tr>
<tr>
<td>SCMLAgreement</td>
<td>(time, unit_price, quantity[, ...])</td>
</tr>
<tr>
<td>SCMLAction</td>
<td>(line, profile, action[, time])</td>
</tr>
<tr>
<td>CFP</td>
<td>(is_buy, publisher, product, time, ...[, ...]) A Call for proposal upon which a negotiation can start</td>
</tr>
<tr>
<td>Loan</td>
<td>(amount, starts_at, total, interest, ...)</td>
</tr>
<tr>
<td>InsurancePolicy</td>
<td>(premium, contract, at_time, ...)</td>
</tr>
<tr>
<td>Factory</td>
<td>(initial_storage[, initial_wallet, ...]) Represents a factory within an SCML world.</td>
</tr>
<tr>
<td>FactoryState</td>
<td>(max_storage, line_schedules, ...) Read Only State of a factory</td>
</tr>
<tr>
<td>SCMLAWI</td>
<td>(world, agent) A single contact point between SCML agents and the world simulation.</td>
</tr>
<tr>
<td>FactoryManager</td>
<td>([name, simulator_type]) Base factory manager class that will be inherited by participant negmas in ANAC 2019.</td>
</tr>
<tr>
<td>DoNothingFactoryManager</td>
<td>([name, simulator_type]) The default factory manager that will be implemented by the committee of ANAC-SCML 2019.</td>
</tr>
<tr>
<td>GreedyFactoryManager</td>
<td>([name, simulator_type, ...]) The default factory manager that will be implemented by the committee of ANAC-SCML 2019.</td>
</tr>
<tr>
<td>JavaFactoryManager</td>
<td>([java_object, ...]) Allows factory managers implemented in Java (using jnegmas) to participate in SCML worlds.</td>
</tr>
<tr>
<td>JavaDoNothingFactoryManager</td>
<td>([...])</td>
</tr>
<tr>
<td>JavaGreedyFactoryManager</td>
<td>([auto_load_java, ...])</td>
</tr>
<tr>
<td>JavaDummyMiddleMan</td>
<td>([auto_load_java, name, ...])</td>
</tr>
<tr>
<td>DefaultBank</td>
<td>(minimum_balance, interest_rate, ...) Represents a bank in the world</td>
</tr>
<tr>
<td>Bank</td>
<td>(*args, **kwargs) Base class for all banks</td>
</tr>
<tr>
<td>DefaultInsuranceCompany</td>
<td>(premium, ...[, ...]) Represents an insurance company in the world</td>
</tr>
<tr>
<td>InsuranceCompany</td>
<td>(*args, **kwargs) Base class for all insurance companies</td>
</tr>
<tr>
<td>SCMLAgent</td>
<td>([name]) The base for all SCM Agents</td>
</tr>
<tr>
<td>FactorySimulator</td>
<td>(initial_wallet, ...[, ...]) Simulates a factory allowing for prediction of storage/balance in the future.</td>
</tr>
<tr>
<td>SlowFactorySimulator</td>
<td>(initial_wallet, ...) A slow factory simulator that runs an internal factory to find-out what will happen in the future</td>
</tr>
<tr>
<td>FastFactorySimulator</td>
<td>(initial_wallet, ...) A faster implementation of the FactorySimulator interface (compared with SlowFactorySimulator.</td>
</tr>
<tr>
<td>DefaultGreedyManager</td>
<td>(*args, [...])</td>
</tr>
<tr>
<td>ScheduleInfo</td>
<td>(final_balance[, valid, start, ...])</td>
</tr>
<tr>
<td>Scheduler</td>
<td>(manager_id, awi[, ...]) Base class for all schedulers</td>
</tr>
<tr>
<td>GreedyScheduler</td>
<td>(manager_id, awi[, ...]) Default scheduler used by the DefaultFactoryManager</td>
</tr>
<tr>
<td>SCMLWorld</td>
<td>(products, processes, factories, ...) The World class running a simulation of supply chain management.</td>
</tr>
<tr>
<td>Factory</td>
<td>(initial_storage[, initial_wallet, ...]) Represents a factory within an SCML world.</td>
</tr>
</tbody>
</table>
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer([name])</td>
<td>Base class of all consumer classes</td>
</tr>
<tr>
<td>ConsumptionProfile([schedule, ...])</td>
<td>Consumer class</td>
</tr>
<tr>
<td>ScheduleDrivenConsumer([profiles, ...])</td>
<td>Consumer class</td>
</tr>
<tr>
<td>Miner([name])</td>
<td>Base class of all miners</td>
</tr>
<tr>
<td>MiningProfile([cv, alpha_t, alpha_q, ...])</td>
<td>Raw Material Generator</td>
</tr>
<tr>
<td>ReactiveMiner([profiles, negotiator_type, ...])</td>
<td>Raw Material Generator</td>
</tr>
</tbody>
</table>

**Product**

```python
class negmas.apps.scml.Product (id, production_level, name, expires_in, catalog_price)
```

*Attributes Summary*

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog_price</td>
<td>Catalog price of the product.</td>
</tr>
<tr>
<td>expires_in</td>
<td>Number of steps within which the product must be consumed. None means never</td>
</tr>
<tr>
<td>id</td>
<td>Product index.</td>
</tr>
<tr>
<td>name</td>
<td>Object name</td>
</tr>
<tr>
<td>production_level</td>
<td>The level of this product in the production graph.</td>
</tr>
</tbody>
</table>

*Attributes Documentation*

**catalog_price**

Catalog price of the product.

**expires_in**

Number of steps within which the product must be consumed. None means never

**id**

Product index. Must be set during construction and MUST be unique for products in the same world

**name**

Object name

**production_level**

The level of this product in the production graph.

**Process**

```python
class negmas.apps.scml.Process (id, production_level, name, inputs, outputs, historical_cost)
```

*Attributes Summary*

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>historical_cost</td>
<td>Average cost for running this process in some world.</td>
</tr>
<tr>
<td>id</td>
<td>A manufacturing process.</td>
</tr>
<tr>
<td>inputs</td>
<td>list of input product name + quantity required and time of consumption relative to the time required for production (value from 0 to 1)</td>
</tr>
<tr>
<td>name</td>
<td>Object name</td>
</tr>
</tbody>
</table>

Continued on next page
### Attributes Documentation

**historical_cost**
Average cost for running this process in some world. Filled by the world

**id**
A manufacturing process.

**inputs**
list of input product name + quantity required and time of consumption relative to the time required for production (value from 0 to 1)

**name**
Object name

**outputs**
list of output product names, quantity required and when it becomes available relative to the time required for production (value from 0 to 1)

**production_level**
The level of this process in the production graph

### InputOutput

**class** `negmas.apps.scml.InputOutput` *(product, quantity, step)*

Bases: `object`

An input/output to a production process

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>product</code></td>
<td>Index of the product used as input or output</td>
</tr>
<tr>
<td><code>quantity</code></td>
<td>Quantity needed/produced</td>
</tr>
<tr>
<td><code>step</code></td>
<td>Relative time within the production at which the input is needed (output is produced)</td>
</tr>
</tbody>
</table>

### Attributes Documentation

**product**
Index of the product used as input or output

**quantity**
Quantity needed/produced

**step**
Relative time within the production at which the input is needed (output is produced)

### RunningCommandInfo

**class** `negmas.apps.scml.RunningCommandInfo` *(profile, beg, end, step, paused, action, updates)*

Bases: `object`
### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>The command type. For the current implementation it will always be run or none for no command</td>
</tr>
<tr>
<td>beg</td>
<td>The time the command is to be executed</td>
</tr>
<tr>
<td>end</td>
<td>The number of steps starting at <code>beg</code> for this command to end (it ends at end - 1)</td>
</tr>
<tr>
<td>is_none</td>
<td></td>
</tr>
<tr>
<td>n_steps</td>
<td></td>
</tr>
<tr>
<td>paused</td>
<td>True if the command is paused</td>
</tr>
<tr>
<td>profile</td>
<td>The manufacturing profile associated with this command.</td>
</tr>
<tr>
<td>step</td>
<td>The time-step relative to <code>beg</code> at the factory is currently executing the Process indicated in profile.</td>
</tr>
<tr>
<td>updates</td>
<td>The status updates implied by this command with their times relative to <code>beg</code></td>
</tr>
</tbody>
</table>

| Type        | rtype int |

### Methods Summary

- `do_nothing()`
- `ended_before(t)`
- `started_on_or_after(t)`

### Attributes Documentation

**action**

The command type. For the current implementation it will always be run or none for no command

**beg**

The time the command is to be executed

**end**

The number of steps starting at `beg` for this command to end (it ends at end - 1)

**is_none**

**n_steps**

Return type int

**paused**

True if the command is paused

**profile**

The manufacturing profile associated with this command. Most importantly, it gives the process and line

**step**

The time-step relative to `beg` at the factory is currently executing the Process indicated in profile. Step will always go up by one every simulation step except if the command is paused where it does not change

**updates**

The status updates implied by this command with their times relative to `beg`
Methods Documentation

classmethod do_nothing()
ended_before(t)
started_on_or_after(t)

ManufacturingProfile

class negmas.apps.scml.ManufacturingProfile(n_steps, cost, initial_pause_cost, running_pause_cost, resumption_cost, cancellation_cost, line, process)

Bases: object

The costs/time required for running a process on a line (with associated cancellation costs etc). This data-structure carries full information about the Process es instead of just its index as in ManufacturingProfileCompiled. It is intended to be used to construct factories

See also:
Factory

Attributes Summary

cancellation_cost Cost of cancelling the process before the last step

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>Cost of manufacturing</td>
</tr>
<tr>
<td>initial_pause_cost</td>
<td>Cost of pausing incurred only at the step a pause is started</td>
</tr>
<tr>
<td>line</td>
<td>The line index</td>
</tr>
<tr>
<td>n_steps</td>
<td>Number of steps needed to complete the manufacturing</td>
</tr>
<tr>
<td>process</td>
<td>The Process associated with this profile</td>
</tr>
<tr>
<td>resumption_cost</td>
<td>Cost of resuming a process</td>
</tr>
<tr>
<td>running_pause_cost</td>
<td>Running cost of pausing</td>
</tr>
</tbody>
</table>

Attributes Documentation

cancellation_cost
    Cost of cancelling the process before the last step
cost
    Cost of manufacturing
initial_pause_cost
    Cost of pausing incurred only at the step a pause is started
line
    The line index
n_steps
    Number of steps needed to complete the manufacturing
process
    The Process associated with this profile
resumption_cost
    Cost of resuming a process
running_pause_cost
    Running cost of pausing
ManufacturingProfileCompiled

```python
class negmas.apps.scml.ManufacturingProfileCompiled(n_steps, cost, initial_pause_cost, running_pause_cost, resumption_cost, cancellation_cost, line, process)
```

Bases: `object`

The costs/time required for running a process on a line (with associated cancellation costs etc).

See also:

`Factory`

## Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cancellation_cost</code></td>
<td>Cost of cancelling the process before the last step</td>
</tr>
<tr>
<td><code>cost</code></td>
<td>Cost of manufacturing</td>
</tr>
<tr>
<td><code>initial_pause_cost</code></td>
<td>Cost of pausing incurred only at the step a pause is started</td>
</tr>
<tr>
<td><code>line</code></td>
<td>The line index</td>
</tr>
<tr>
<td><code>n_steps</code></td>
<td>Number of steps needed to complete the manufacturing</td>
</tr>
<tr>
<td><code>process</code></td>
<td>The Process index</td>
</tr>
<tr>
<td><code>resumption_cost</code></td>
<td>Cost of resuming a process</td>
</tr>
<tr>
<td><code>running_pause_cost</code></td>
<td>Running cost of pausing</td>
</tr>
</tbody>
</table>

## Methods Summary

```python
from_manufacturing_profile(profile, process2ind)
```

## Attributes Documentation

### `cancellation_cost`

Cost of cancelling the process before the last step

### `cost`

Cost of manufacturing

### `initial_pause_cost`

Cost of pausing incurred only at the step a pause is started

### `line`

The line index

### `n_steps`

Number of steps needed to complete the manufacturing

### `process`

The Process index

### `resumption_cost`

Cost of resuming a process

### `running_pause_cost`

Running cost of pausing
Methods Documentation

classmethod `from_manufacturing_profile`

```
class ProductManufacturingInfo
    class negmas.apps.scml.ProductManufacturingInfo (profile, quantity, step)
        Bases: object
        Gives full information about a manufacturing process that can generate or consume a product.
        See also:
        consuming and producing of Factory

Attributes Summary

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile</td>
<td>The ManufacturingProfile index</td>
</tr>
<tr>
<td>quantity</td>
<td>The quantity generated/consumed by running this manufacturing info</td>
</tr>
<tr>
<td>step</td>
<td>The step from the beginning at which the Product is received/consumed</td>
</tr>
</tbody>
</table>

Attributes Documentation

- **profile**
  The ManufacturingProfile index

- **quantity**
  The quantity generated/consumed by running this manufacturing info

- **step**
  The step from the beginning at which the Product is received/consumed

FactoryStatusUpdate

class negmas.apps.scml.FactoryStatusUpdate (balance, storage)
    Bases: object

Attributes Summary

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>The update to the balance</td>
</tr>
<tr>
<td>is_empty</td>
<td></td>
</tr>
<tr>
<td>storage</td>
<td>The updates to be applied to the storage after this step</td>
</tr>
</tbody>
</table>

Methods Summary

- **combine**(other)
  Combines this status update with another one in place

Continued on next page
combine_sets(dst, src)

Combines a set of updates over time with another in place (overriding first).

:param dst: First set of updates to be combined into.

:param src: Second set of updates to be combined from.

empty()

make_empty()

Makes the update an empty one.

Attributes Documentation

balance

The update to the balance

is_empty

storage

The updates to be applied to the storage after this step

Methods Documentation

combine(other)

Combines this status update with another one in place.

Parameters

other (FactoryStatusUpdate) – The other status update.

Returns

None

classmethod combine_sets(dst, src)

Combines a set of updates over time with another in place (overriding first).

:param dst: First set of updates to be combined into.

:param src: Second set of updates to be combined from.

Returns:

empty()

make_empty()

Makes the update an empty one.

Return type

None

Job

class negmas.apps.scml.Job (profile, time, line, action, contract, override)

Bases: object

Describes a job to be run on one production line of a Factory.

Attributes Summary

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>The command type.</td>
</tr>
<tr>
<td>contract</td>
<td>The sell contract associated with the command.</td>
</tr>
<tr>
<td>line</td>
<td>Index of the line on which the job is to be scheduled.</td>
</tr>
</tbody>
</table>
Table 14 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>override</td>
<td>Whether to override existing commands when the</td>
</tr>
<tr>
<td></td>
<td>job is to be executed.</td>
</tr>
<tr>
<td>profile</td>
<td>The process for run commands</td>
</tr>
<tr>
<td>time</td>
<td>The time the command is to be executed</td>
</tr>
</tbody>
</table>

### Methods Summary

**is_cancelling**(job) Determines if the given jobs cancels this one

### Attributes Documentation

**action**

The command type. For the current implementation it can be run/pause/resume/stop/cancel with cancel cancelling any other command type.

**contract**

The sell contract associated with the command

**line**

Index of the line on which the job is to be scheduled. Notice that it will be ignored for run actions.

**override**

Whether to override existing commands when the job is to be executed.

**profile**

The process for run commands

**time**

The time the command is to be executed

### Methods Documentation

**is_cancelling**(job)

Determines if the given jobs cancels this one

**Parameters**

- **job** *(Job)*

**Returns:**

- **Return type** *bool*

### ProductionNeed

**class** negmas.apps.scml.ProductionNeed

*Bases:* object

Describes some quantity of a product that is needed to honor a (sell) contract.

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>needed_for</td>
<td>The contract for which the product is needed</td>
</tr>
<tr>
<td>product</td>
<td>The product needed</td>
</tr>
<tr>
<td>quantity_in_storage</td>
<td>The quantity already found in storage</td>
</tr>
<tr>
<td>quantity_to_buy</td>
<td>The quantity need to be bought</td>
</tr>
<tr>
<td>step</td>
<td>The time step at which the product is needed</td>
</tr>
</tbody>
</table>
Attributes Documentation

needed_for
The contract for which the product is needed

product
The product needed

quantity_in_storage
The quantity already found in storage

quantity_to_buy
The quantity need to be bought

step
The time step at which the product is needed

MissingInput

class negmas.apps.scml.MissingInput (product, quantity)
   Bases: object

Attributes Summary

<table>
<thead>
<tr>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantity</td>
</tr>
</tbody>
</table>

Attributes Documentation

product
quantity

ProductionReport

class negmas.apps.scml.ProductionReport (line, started, continuing, finished, failure, updates)
   Bases: object

Attributes Summary

<table>
<thead>
<tr>
<th>failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_empty</td>
</tr>
<tr>
<td>no_production</td>
</tr>
</tbody>
</table>

Attributes Documentation

failed
is_empty
no_production
ProductionFailure

class negmas.apps.scml.ProductionFailure(line, command, missing_inputs, missing_money, missing_space)

Bases: object

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Information about the command that failed</td>
</tr>
<tr>
<td>line</td>
<td>ID of the line that failed</td>
</tr>
<tr>
<td>missing_inputs</td>
<td>The missing inputs if any with their quantities</td>
</tr>
<tr>
<td>missing_money</td>
<td>The amount of money needed for production that is not available</td>
</tr>
<tr>
<td>missing_space</td>
<td>The amount space needed in storage but not found</td>
</tr>
</tbody>
</table>

Attributes Documentation

command
Information about the command that failed

line
ID of the line that failed

missing_inputs
The missing inputs if any with their quantities

missing_money
The amount of money needed for production that is not available

missing_space
The amount space needed in storage but not found

FinancialReport

class negmas.apps.scml.FinancialReport(agent, step, cash, liabilities, inventory, credit_rating)

Bases: object

Reports that financial standing of an agent at a given time in the simulation

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>The balance of the agent defined as the difference between its available cash + inventory and its liabilities</td>
</tr>
</tbody>
</table>

Attributes Documentation

balance
The balance of the agent defined as the difference between its available cash + inventory and its liabilities

Remarks:

- If the inventory was not calculated (due to having at least one product with unknown catalog price), it is used as zero in the equation.
**SCMLAgreement**

```python
class negmas.apps.scml.SCMLAgreement(time, unit_price, quantity, penalty=None, signing_delay=-1)
```

**Bases:** `negmas.outcomes.OutcomeType`

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>penalty</td>
<td>penalty</td>
</tr>
<tr>
<td>signing_delay</td>
<td>Delay between agreement conclusion and signing it to be binding</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asdict()</td>
<td>Converts the outcome to a dict containing all fields</td>
</tr>
<tr>
<td>astuple()</td>
<td>Converts the outcome to a tuple where the order of items is the same as they are defined as fields</td>
</tr>
<tr>
<td>get(name[, default])</td>
<td>Acts like dict.get</td>
</tr>
<tr>
<td>keys()</td>
<td></td>
</tr>
<tr>
<td>values()</td>
<td></td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- **penalty** = None
  - penalty
- **signing_delay** = -1
  - Delay between agreement conclusion and signing it to be binding

**Methods Documentation**

- **asdict()**
  - Converts the outcome to a dict containing all fields
- **astuple()**
  - Converts the outcome to a tuple where the order of items is the same as they are defined as fields
- **get(name[, default])**
  - Acts like dict.get
- **keys()**
  - Return type `List[str]`
- **values()**
  - Return type `List[str]`

**SCMLAction**

```python
class negmas.apps.scml.SCMLAction(line, profile, action, time=0)
```

**Bases:** `object`
Attributes Summary

time

Time to execute the action at

Attributes Documentation

time = 0
Time to execute the action at

CFP

class negmas.apps.scml.CFP(is_buy, publisher, product, time, unit_price, quantity, penalty=None, signing_delay=None, money_resolution=None, id=<factory>)

Bases: negmas.outcomes.OutcomeType

A Call for proposal upon which a negotiation can start

Attributes Summary

issues

Returns the set of issues associated with this CFP.

max_penalty

max_quantity

max_signing_delay

max_time

max_unit_price

min_penalty

min_quantity

min_signing_delay

min_time

min_unit_price

money_resolution

If not None then it is the minimum unit of money (e.g.

outcomes

penalty

penalty per missing item in case the seller cannot provide the required quantity.

signing_delay

The grace period after which the agents are asked to confirm signing the contract

Methods Summary

asdict()

Converts the outcome to a dict containing all fields

astuple()

Converts the outcome to a tuple where the order of items is the same as they are defined as fields

from_java(idict[, class_name])

rtype CFP

get(name[, default])

Acts like dict.get

keys()

rtype List[str]

satisfies(query)

Tests whether the CFP satisfies the conditions set by the query

Continued on next page
Table 25 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_java()</td>
<td></td>
<td>rtype List[Str]</td>
</tr>
<tr>
<td>values()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- **issues**
  - Returns the set of issues associated with this CFP. Notice that some of the issues may have a single value.

- **max_penalty**
- **max_quantity**
- **max_signing_delay**
- **max_time**
- **max_unit_price**
- **min_penalty**
- **min_quantity**
- **min_signing_delay**
- **min_time**
- **min_unit_price**

- **money_resolution = None**
  - If not None then it is the minimum unit of money (e.g. 1 for dollar, 0.01 for cent, etc)

- **outcomes**
- **penalty = None**
  - penalty per missing item in case the seller cannot provide the required quantity. May be negotiable.

- **signing_delay = None**
  - The grace period after which the agents are asked to confirm signing the contract

**Methods Documentation**

- **asdict()**
  - Converts the outcome to a dict containing all fields

- **astuple()**
  - Converts the outcome to a tuple where the order of items is the same as they are defined as fields

- **classmethod from_java(idict, class_name=None)**
  - Return type CFP

- **get(name, default=None)**
  - Acts like dict.get

- **keys()**
  - Return type List[Str]

- **satisfies(query)**
  - Tests whether the CFP satisfies the conditions set by the query
  
  **Parameters**
  - query (Dict[Str, Any]) – A dictionary given the conditions. See Remarks for details

  **Remarks:**

---

6.1. negmas.apps.scml Package
• The query dictionary can be used to specify any conditions that are required in the CFP. Only CFPs that satisfy ALL the conditions specified in the query are considered satisfying the query. The following keys can be set with corresponding meanings:

- **is_buy**  True or False. If both are OK, just do not add this key
- **publisher**  A string or **SCMLAgent** specifying a specific publisher
- **publishers**  A list of publishers (see publisher key)
- **product**  A string specifying a product name
- **products**  A list of products (see product key)
- **time**  A number, list or 2-items-tuple (range) specifying possible times to consider satisfactory
- **unit_price**  A number, list or 2-items-tuple (range) specifying possible prices to consider satisfactory
- **quantity**  A number, list or 2-items-tuple (range) specifying possible quantities to consider OK
- **penalty**  A number, list or 2-items-tuple (range) specifying possible penalties to consider satisfactory

**Return type**  **bool**

```
to_java()
values()
```
**Return type**  **List[str]**

**Loan**

```python
class negmas.apps.scml.Loan(amount, starts_at, total, interest, installment, n_installments)
    Bases: object
```

**InsurancePolicy**

```python
class negmas.apps.scml.InsurancePolicy(premium, contract, at_time, against)
    Bases: object
```

**Factory**

```python
class negmas.apps.scml.Factory(initial_storage, initial_wallet=0.0, id=<factory>, profiles=<factory>, max_storage=922372036854775807, min_storage=0, min_balance=0, _jobs=<factory>)
    Bases: object
```

Represents a factory within an SCML world. It is only accessed by the World so it need not be made public.

**Attributes Summary**

| **balance** | The total balance of the factory |
| **commands** | **rtype**  <built-in function array> |
| **hidden_money** | **rtype**  **float** |

Continued on next page
### Table 26 – continued from previous page

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden_storage</td>
<td>Dict[int, int]</td>
<td></td>
</tr>
<tr>
<td>initial_balance</td>
<td></td>
<td>Initial balance of the factory</td>
</tr>
<tr>
<td>initial_wallet</td>
<td></td>
<td>Initial Wallet</td>
</tr>
<tr>
<td>jobs</td>
<td>Dict[Tuple[int, int], Job]</td>
<td></td>
</tr>
<tr>
<td>line_schedules</td>
<td></td>
<td>rtype &lt;built-in function array&gt;</td>
</tr>
<tr>
<td>loans</td>
<td>float</td>
<td>rtype float</td>
</tr>
<tr>
<td>max_storage</td>
<td></td>
<td>Maximum storage allowed in this factory</td>
</tr>
<tr>
<td>min_balance</td>
<td></td>
<td>Minimum allowed balance</td>
</tr>
<tr>
<td>min_storage</td>
<td></td>
<td>Minimum allowed storage per product</td>
</tr>
<tr>
<td>n_lines</td>
<td>int</td>
<td>rtype int</td>
</tr>
<tr>
<td>next_step</td>
<td>int</td>
<td>rtype int</td>
</tr>
<tr>
<td>storage</td>
<td>Dict[int, int]</td>
<td></td>
</tr>
<tr>
<td>total_balance</td>
<td></td>
<td>total balance including hidden money</td>
</tr>
<tr>
<td>total_storage</td>
<td>int</td>
<td>rtype int</td>
</tr>
<tr>
<td>wallet</td>
<td>float</td>
<td>rtype float</td>
</tr>
</tbody>
</table>

### Methods Summary

- **add_loan**(total)  
  rtype None

- **attach_to_world**(world)  
  rtype None

- **buy**(product, quantity, price)  
  rtype None

- **hide_funds**(amount)  
  rtype None

- **hide_product**(product, quantity)  
  rtype None

- **pay**(payment)  
  rtype None

Continued on next page
Table 27 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>RETURN TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive(payment)</td>
<td>None</td>
<td>Schedules the given job at its time and line optionally overriding whatever was already scheduled.</td>
</tr>
<tr>
<td>schedule(job[, override])</td>
<td></td>
<td>Schedules the given job at its time and line optionally overriding whatever was already scheduled.</td>
</tr>
<tr>
<td>sell(product, quantity, price)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>step()</td>
<td>List[ProductionReport]</td>
<td></td>
</tr>
<tr>
<td>transport_from(product, quantity)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>transport_to(product, quantity)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>unhide_funds(amount)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>unhide_product(product, quantity)</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

Attributes Documentation

- **balance**
  The total balance of the factory
  - Return type float

- **commands**
  - Return type <built-in function array>

- **hidden_money**
  - Return type float

- **hidden_storage**
  - Return type Dict[int, int]

- **initial_balance** = 0.0
  Initial balance of the factory

- **initial_wallet** = 0.0
  Initial Wallet

- **jobs**
  - Return type Dict[Tuple[int, int], Job]

- **line_schedules**
  - Return type <built-in function array>

- **loans**
  - Return type float

- **max_storage** = 9223372036854775807
  Maximum storage allowed in this factory
**min_balance** = 0
  Minimum allowed balance

**min_storage** = 0
  Minimum allowed storage per product

**n_lines**
  Return type int

**next_step**
  Return type int

**storage**
  Return type Dict[int, int]

**total_balance**
  total balance including hidden money
  Return type float

**total_storage**
  Return type int

**wallet**
  Return type float

### Methods Documentation

**add_loan** (**total**)  
  Return type None

**attach_to_world** (**world**)  

**buy** (**product, quantity, price**)  
  Return type None

**hide_funds** (**amount**)  
  Return type None

**hide_product** (**product, quantity**)  
  Return type None

**pay** (**payment**)  
  Return type None

**receive** (**payment**)  
  Return type None

**schedule** (**job, override=False**)  
  Schedules the given job at its time and line optionally overriding whatever was already scheduled  
  :type job: Job  
  :param job:  
  :param override:
  Returns Success/failure

**sell** (**product, quantity, price**)  
  Return type None

**step**()
Return type  List[ProductionReport]
transport_from (product, quantity)
Return type  None
transport_to (product, quantity)
Return type  None
unhide_funds (amount)
Return type  None
unhide_product (product, quantity)
Return type  None

FactoryState

class  negmas.apps.scml.FactoryState (max_storage, line_schedules, storage, wallet, hidden_money, hidden_storage, loans, n_lines, profiles, next_step, commands, jobs)

Bases:  object
Read Only State of a factory

SCMLAWI

class  negmas.apps.scml.SCMLAWI (world, agent)
Bases:  negmas.situated.AgentWorldInterface
A single contact point between SCML agents and the world simulation.
The agent can access the world simulation in one of two ways:
1. Attributes and methods available in this Agent-World-Interface
2. Attributes and methods in the FactoryManager object itself which provide handy shortcuts to the agent-world interface

Attributes

Simulation settings

• current_step:  Current simulation step
• default_signing_delay:  The grace period allowed between contract conclusion and signature by default (i.e. if not agreed upon during the negotiation)
• n_steps:  Total number of simulation steps.
• relative_time:  The fraction of total simulation time elapsed (it will be a number between 0 and 1)

Production Graph

• products:  A list of Product objects giving all products defined in the world simulation
• processes:  A list of Process objects giving all products defined in the world simulation

Agent Related

• state:  The current private state available to the agent. In SCML it is a FactoryState object.

Methods

Production Control

• schedule_job:  Schedules a Job for production sometime in the future
- **schedule_production**: Schedules production using profile number instead of a Job object
- **cancel_production**: Cancels already scheduled production (if it did not start yet) or stop a running production.
- **execute**: A general function to execute any command on the factory. There is no need to directly call this function as the SCMLAWI provides convenient functions (e.g. schedule_job, hide_funds, etc) to achieve the same goal without having to worry about creating Action objects

**Storage and Wallet Control**

- **hide_funds**: Hides funds from the view of the simulator. Note that when bankruptcy is considered, hidden funds are visible to the simulator.
- **hide_inventory**: Hides inventory from the view of the simulator. Note that when bankruptcy is considered, hidden funds are visible to the simulator.
- **unhide_funds**: Un-hides funds hidden earlier with a call to hide_funds
- **unhide_inventory**: Un-hides inventory hidden earlier with a call to hide_inventory

**Negotiation and CFP Control**

- **register_cfp**: Registers a Call-for-Proposals on the bulletin board.
- **remove_cfp**: Removes a Call-for-Proposals from the bulletin board.
- **request_negotiation**: Requests a negotiation based on the content of a CFP published on the bulletin-board. *It is recommended not to use this method directly and to request negotiations using the request_negotiation method of FactoryManager (i.e. use self.request_negotiation instead of self.awi.request_negotiation). This makes it possible for NegMAS to keep track of existing requested_negotiations and running_negotiations for you.*

**Notification Control**

- **receive_financial_reports**: Register/unregisters interest in receiving financial reports for an agent, a set of agents or all agents.
- **register_interest**: registers interest in receiving CFPs about a set of products. By default all FactoryManager objects are registered to receive all CFPs for any product they can produce or need to consumer according to their line-profiles.
- **unregister_interest**: unregisters interest in receiving CFPs about a set of products.

**Information about Other Agents**

- **is_bankrupt**: Asks about the bankruptcy status of an agent
- **receive_financial_reports**: Register/unregisters interest in receiving financial reports for an agent, a set of agents or all agents.
- **reports_at**: reads all financial reports produced at a given time-step
- **reports_for**: reads all financial reports of a given agent

**Financial Control**

- **evaluate_insurance**: Asks for the premium to be paid for insuring against partner breaches for a given contract
- **buy_insurance**: Buys an insurance against partner breaches for a given contract

**Bulletin-Board**

The bulletin-board is a key-value store. These methods allows the agent to interact with it. The 'SCMLAWI' provides convenient functions for recording to the bulletin-board so you mostly need to use read/query functions.

- **bb_read**: Reads a complete section or a single value from the bulletin-board
- **bb_query**: Returns all records in the given section/sections of the bulletin-board that satisfy a query
• **bb_record**: Registers a record in the bulletin-board.
• **bb_remove**: Removes a record from the bulletin-board.

The following list of sections are available in the SCML Bulletin-Board (Use the exact string for the `section` parameter of any method starting with `bb_`):

- **cfps**: All CFPs currently on the board. The key is the CFP ID
- **products**: A list of all products. The key is the product index/ID
- **processes**: A list of all processes. The key is the product index/ID
- **bankruptcy**: The bankruptcy list giving names of all bankrupt agents.
- **reports_time**: Financial reports indexed by time.
- **reports_agent**: Financial reports indexed by agent
- **breaches**: Breach-list indexed by breach ID giving all breaches committed in the system
- **settings**: Static settings of the simulation.

The following settings are currently available:

- **breach_penalty_society**: Penalty of breaches paid to society (as a fraction of contract value). This is always paid for every breach whether or not there is a negotiated breach.
- **breach_penalty_victim**: Penalty of breaches paid to victim (as a fraction of contract value). This is always paid for every breach whether or not there is a negotiated breach.
- **immediate_negotiations**: Whether negotiations start immediately when registered (the other possibility – which is the default – is for them to start at the next production step).
- **negotiation_speed_multiple**: Number of negotiation steps that finish in a single production step.
- **negotiation_n_steps**: Maximum allowed number of steps (rounds) in any negotiation
- **negotiation_step_time_limit**: The maximum real-time allowed for each negotiation step (round)
- **negotiation_time_limit**: The time limit for a complete negotiation.
- **transportation_delay**: Transportation delay when products are moved between factories. Default is zero.
- **transfer_delay**: The delay in transferring funds between factories when executing a contract. Default is zero.
- **n_steps**: Number of simulation steps
- **time_limit**: Time limit for the complete simulation

- **stats**: Global statistics about the simulation. **Not available for SCML 2019 league.**

**Logging**

- **logerror**: Logs an error in the world simulation log file
- **logwarning**: Logs a warning in the world simulation log file
- **loginfo**: Logs information in the world simulation log file
- **logdebug**: Logs debug information in the world simulation log file

**Attributes Summary**

<table>
<thead>
<tr>
<th>current_step</th>
<th>Current simulation step</th>
<th>Current simulation step</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_signing_delay</td>
<td>type int</td>
<td>Current simulation step</td>
</tr>
</tbody>
</table>
Table 28 – continued from previous page

<table>
<thead>
<tr>
<th>n_steps</th>
<th>Number of steps in a simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>processes</td>
<td>Processes in the world</td>
</tr>
<tr>
<td>products</td>
<td>Products in the world</td>
</tr>
<tr>
<td>relative_time</td>
<td>Relative time of the simulation going from 0 to 1</td>
</tr>
<tr>
<td>state</td>
<td>Returns the private state of the agent in that world.</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>bb_query(section, query[, query_keys])</th>
<th>Returns all records in the given section/sections of the bulletin-board that satisfy the query</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb_read(section, key)</td>
<td>Reads the value associated with given key from the bulletin board</td>
</tr>
<tr>
<td>bb_record(section, value[, key])</td>
<td>Records data in the given section of the bulletin board</td>
</tr>
<tr>
<td>bb_remove(section, *[query, key, . . .])</td>
<td>Removes a value or a set of values from the bulletin Board</td>
</tr>
<tr>
<td>buy_insurance(contract)</td>
<td>Buys insurance for the contract by the premium calculated by the insurance company.</td>
</tr>
<tr>
<td>cancel_production(line, step, contract[, . . .])</td>
<td>Stops/cancels production scheduled at the given line at the given time.</td>
</tr>
<tr>
<td>evaluate_insurance(contract[, t])</td>
<td>Can be called to evaluate the premium for insuring the given contract against breaches committed by others</td>
</tr>
<tr>
<td>execute(action[, callback])</td>
<td>Executes an action in the world simulation</td>
</tr>
<tr>
<td>hide_funds(amount)</td>
<td>Hides the given amount of money so that it is not accessible by the simulator and does not appear in reports etc.</td>
</tr>
<tr>
<td>hide_inventory(product, quantity)</td>
<td>Hides the given quantity of the given product so that it is not accessible by the simulator and does not appear in reports etc.</td>
</tr>
<tr>
<td>is_bankrupt(agent_id)</td>
<td>Checks whether the given agent is bankrupt</td>
</tr>
<tr>
<td>logdebug(msg)</td>
<td>Logs a WARNING message</td>
</tr>
<tr>
<td>logerror(msg)</td>
<td>Logs a WARNING message</td>
</tr>
<tr>
<td>loginfo(msg)</td>
<td>Logs an INFO message</td>
</tr>
<tr>
<td>logwarning(msg)</td>
<td>Logs a WARNING message</td>
</tr>
<tr>
<td>receive_financial_reports((receive, agents))</td>
<td>Registers/unregisters interest in receiving financial reports</td>
</tr>
<tr>
<td>register_cfp(cf)</td>
<td>Registers a CFP</td>
</tr>
<tr>
<td>register_interest(products)</td>
<td>registers interest in receiving callbacks about CFPs related to these products</td>
</tr>
<tr>
<td>remove_cfp(cf)</td>
<td>Removes a CFP</td>
</tr>
<tr>
<td>reports_at([step])</td>
<td>Gets all financial reports of all agents at a given step</td>
</tr>
<tr>
<td>reports_for(agent_id)</td>
<td>Gets all financial reports of an agent (in the order of their publication)</td>
</tr>
<tr>
<td>request_negotiation(cf, req_id[, roles, . . .])</td>
<td>Requests a negotiation with the publisher of a given CFP</td>
</tr>
<tr>
<td>request_negotiation_about(issues, partners, . . .)</td>
<td>Overrides the method of the same name in the base class to disable it in SCM Worlds.</td>
</tr>
<tr>
<td>schedule_job(job, contract)</td>
<td>Schedules production using a Job object.</td>
</tr>
<tr>
<td>schedule_production(profile, step[, . . .])</td>
<td>Schedules production on the agent’s factory</td>
</tr>
<tr>
<td>stop_production(line, step, contract[, override])</td>
<td>Stops/cancels production scheduled at the given line at the given time.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>unhide_funds(amount)</code></td>
<td>Un-hides the given amount of money so that it is not accessible by the simulator and does not appear in reports etc.</td>
</tr>
<tr>
<td><code>unhide_inventory(product, quantity)</code></td>
<td>Un-hides the given quantity of the given product so that it is not accessible by the simulator and does not appear in reports etc.</td>
</tr>
<tr>
<td><code>unregister_interest(products)</code></td>
<td>registers interest in receiving callbacks about CFPs related to these products</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- **current_step**
  - Current simulation step
  - Return type: int
- **default_signing_delay**
  - Return type: int
- **n_steps**
  - Number of steps in a simulation
  - Return type: int
- **processes**
  - Processes in the world
  - Return type: List[Process]
- **products**
  - Products in the world
  - Return type: List[Product]
- **relative_time**
  - Relative time of the simulation going from 0 to 1
  - Return type: float
- **state**
  - Returns the private state of the agent in that world.
  - In the SCML world, that is a reference to its factory. You are allowed to read information from the returned `Factory` but **not to modify it or call ANY methods on it that modify the state**.
  - Return type: FactoryState

**Methods Documentation**

- **bb_query** *(section, query, query_keys=False)*
  - Returns all records in the given section/sections of the bulletin-board that satisfy the query
  - Parameters
    - `section` *(Union[str, List[str], None])* – Either a section name, a list of sections or None specifying ALL public sections (see remarks)
    - `query` *(Any)* – The query which is USUALLY a dict with conditions on it when querying values and a RegExp when
    - `keys` *(querying)* –
    - `query_keys` – Whether the query is to be applied to the keys or values.
  - Returns value pairs giving all records that satisfied the given requirements.
Return type
  • A dictionary with key

Remarks:
  • A public section is a section with a name that does not start with an underscore
  • If a set of sections is given, and two records in different sections had the same key, only one of
    them will be returned
  • Key queries use regular expressions and match from the beginning using the standard re.match
    function

\textbf{bb\_read}(\textit{section, key})

Reads the value associated with given key from the bulletin board

Parameters
  • \textit{section} (str) – section name
  • \textit{key} (str) – key

Return type  Optional[\textit{Any}]

Returns  Content of that key in the bulletin-board

\textbf{bb\_record}(\textit{section, value, key=None})

Records data in the given section of the bulletin board

Parameters
  • \textit{section} (str) – section name (can contain subsections separated by '/')
  • \textit{key} (Optional[str]) – The key
  • \textit{value} (\textit{Any}) – The value

Return type  None

\textbf{bb\_remove}(\textit{section, *, query=None, key=None, query\_keys=False, value=None})

Removes a value or a set of values from the bulletin Board

Parameters
  • \textit{section} (Union[str, List[str], None]) – The section
  • \textit{query} (Optional[\textit{Any}]) – the query to use to select what to remove
  • \textit{key} (Optional[str]) – the key to remove (no need to give a full query)
  • \textit{query\_keys} (bool) – Whether to apply the query (if given) to keys or values
  • \textit{value} (Optional[\textit{Any}]) – Value to be removed

Returns  Success of failure

Return type  bool

\textbf{buy\_insurance}(\textit{contract})

Buys insurance for the contract by the premium calculated by the insurance company.

Remarks: The agent can call \texttt{evaluate\_insurance} to find the premium that will be used.

Return type  bool

\textbf{cancel\_production}(\textit{line, step, contract, override=True})

Stops/cancels production scheduled at the given line at the given time.

Parameters
  • \textit{line} (int) – One of the factory lines (index)
  • \textit{step} (int) – Step to stop/cancel production at
**evaluate_insurance** (*contract, t=None*)
Can be called to evaluate the premium for insuring the given contract against breaches committed by others

**Parameters**
- *contract* (*Contract*) – hypothetical contract
- *t* (*Optional[int]*) – time at which the policy is to be bought. If None, it means current step

**Return type** *Optional[float]*

**execute** (*action, callback=None*)
Executes an action in the world simulation

**Return type** *bool*

**hide_funds** (*amount*)
Hides the given amount of money so that it is not accessible by the simulator and does not appear in reports etc.

**Parameters**
- *amount* (*float*) – The amount of money to hide

**Remarks:**
- if the current cash in the agent’s wallet is less than the amount to be hidden, all the cash is hidden.
- hiding is always immediate

**Return type** *None*

**hide_inventory** (*product, quantity*)
Hides the given quantity of the given product so that it is not accessible by the simulator and does not appear in reports etc.

**Parameters**
- *product* (*int*) – product index
- *quantity* (*int*) – the amount of the product to hide

**Remarks:**
- if the current quantity in storage of the product is less than the amount to be hidden, whatever quantity exists is hidden
- hiding is always immediate

**Return type** *None*

**is_bankrupt** (*agent_id*)
Checks whether the given agent is bankrupt

**Parameters**
- *agent_id* (*str*) – Agent ID

**Return type** *bool*

**Returns** The bankruptcy state of the agent

**logdebug** (*msg*)
Logs a WARNING message

**Parameters**
- *msg* (*str*) – The message to log

**Returns:**

**Return type** *None*

**logerror** (*msg*)
Logs a WARNING message
Parameters `msg (str)` – The message to log

Returns:

Return type None

**loginfo**( `msg` )
Logs an INFO message

Parameters `msg (str)` – The message to log

Returns:

Return type None

**logwarning**( `msg` )
Logs a WARNING message

Parameters `msg (str)` – The message to log

Returns:

Return type None

**receive_financial_reports**( `receive=True, agents=None` )
Registers/unregisters interest in receiving financial reports

Parameters

• `receive (bool)` – True to receive and False to stop receiving

• `agents (Optional[List[str]])` – If given reception is enabled/disabled only for the given set of agents.

Remarks:

• by default financial reports are not sent to any agents. To opt-in to receive financial reports, call this method.

Return type None

**register_cfp**( `cfp` )
Registers a CFP

Return type None

**register_interest**( `products` )
registers interest in receiving callbacks about CFPs related to these products

Return type None

**remove_cfp**( `cfp` )
Removes a CFP

Return type bool

**reports_at**( `step=None` )
Gets all financial reports of all agents at a given step

Parameters `step (Optional[int])` – Step at which the reports are required. If None, the last set of reports is returned

Return type Dict[str, FinancialReport]

Returns A dictionary with agent IDs in keys and their financial reports at the given time as values

**reports_for**( `agent_id` )
Gets all financial reports of an agent (in the order of their publication)

Parameters `agent_id (str)` – Agent ID
Returns:

**Return type** `List[FinancialReport]`

`request_negotiation(cfp, req_id, roles=None, mechanism_name=None, mechanism_params=None)`

Requests a negotiation with the publisher of a given CFP

**Parameters**

- `cfp` (*CFP*) – The CFP to negotiate about
- `req_id` (*str*) – A string that is passed back to the caller in all callbacks related to this negotiation
- `roles` (*Optional[List[str]*) – The roles of the CFP publisher and the agent (in that order). By default no roles are passed (None)
- `mechanism_name` (*Optional[str]*) – The mechanism type to use. If not given the default mechanism from the world will be used
- `mechanism_params` (*Optional[Dict[str, Any]*) – Parameters of the mechanism

**Return type** `bool`

**Returns** Success of failure of the negotiation request

**Remarks:**

- The `SCMLAgent` class implements another `request_negotiation` method that does not receive a `req_id`. This helper method is recommended as it generates the required `req_id` and passes it keeping track of requested negotiations (and later of running negotiations). Call this method directly only if you do not intend to use the `requested_negotiations` and `running_negotiations` properties of the `SCMLAgent` class

`request_negotiation_about(issues, partners, req_id, roles=None, annotation=None, mechanism_name=None, mechanism_params=None)`

Overrides the method of the same name in the base class to disable it in SCM Worlds.

Do not call this method

`schedule_job(job, contract)`

Schedules production using a `Job` object. This can be used to schedule any kind of job

**Parameters**

- `job` (*Job*) – The job description
- `contract` (*Optional[Contract]*) – The contract for which the job is scheduled (optional)

**Remarks:**

- Notice that actions that require the profile member of Job (run) never use the line member and vice versa.

`schedule_production(profile, step, contract=None, override=True)`

Schedules production on the agent’s factory

**Parameters**

- `profile` (*int*) – Index of the profile in the agent’s `compiled_profiles` list
- `step` (*int*) – The step to start production according to the given profile
- `contract` (*Optional[Contract]*) – The contract for which the production is scheduled (optional)
- `override` (*bool*) – Whether to override existing production jobs schedules at the same time.
stop_production (line, step, contract, override=True)

Stops/cancels production scheduled at the given line at the given time.

Parameters

• line (int) – One of the factory lines (index)
• step (int) – Step to stop/cancel production at
• contract (Optional[Contract]) – The contract for which the job is scheduled (optional)
• override (bool) – Whether to override existing production jobs schedules at the same time.

unhide_funds (amount)

Un-hides the given amount of money so that it is not accessible by the simulator and does not appear in reports etc.

Parameters amount (float) – The amount of money to unhide

Remarks:

• if the current cash in the agent’s wallet is less than the amount to be hidden, all the cash is hidden.
• hiding is always immediate

unhide_inventory (product, quantity)

Un-hides the given quantity of the given product so that it is not accessible by the simulator and does not appear in reports etc.

Parameters

• product (int) – product index
• quantity (int) – the amount of the product to hide

Remarks:

• if the current quantity in storage of the product is less than the amount to be hidden, whatever quantity exists is hidden
• hiding is always immediate

unregister_interest (products)

registers interest in receiving callbacks about CFPs related to these products

Return type None

FactoryManager

class negmas.apps.scml.FactoryManager (name=None, simulator_type=<class 'negmas.apps.scml.simulators.FastFactorySimulator'>)

Base factory manager class that will be inherited by participant negmas in ANAC 2019.

The agent can access the world simulation in one of two ways:

1. Attributes and methods available in the Agent-World-Interface (See SCMLAWI documentation for those).
2. Attributes and methods in the `FactoryManager` object itself. All factory managers will have the following attributes and methods that simplify the interaction with the world simulation. Some of these attributes/methods are convenient ways to access functionality already available in the agent’s internal `SCMLAWI`.

**Attributes**

**Agent information**

- `id`: The unique ID assigned to this agent. This is unique system-wide and is what is used in contracts, CFPs, etc.
- `name`: A name of the agent used for display purposes only. The simulator never accesses or uses this name except in printing and logging.
- `uuid`: Another name of the `id`.
- `type_name`: A string giving the type of the agent (as a fully qualified python class name).

**Capabilities/Profiles**

- `line_profiles`: A mapping specifying for each line index, all the profiles that can be run on it
- `process_profiles`: A mapping specifying for each `Process` index, all the profiles used to run it in the factory
- `producing`: Mapping from a product index to all manufacturing processes that can generate it
- `consuming`: Mapping from a product index to all manufacturing processes that can consume it
- `compiled_profiles`: All the profiles to be used by the factory belonging to this agent compiled to use process indices
- `max_storage`: Maximum storage available to the agent. Zero, None or float(‘inf’) all indicate unlimited storage.

**Production Graph** (also accessible through `awi`)

- `products`: List of products in the system
- `processes`: List of processes in the system

**Helper Objects**

- `awi`: The `SCMLAWI` instance assigned to this agent. It can be used to interact with the simulation (See `SCMLAWI` documentation).
- `simulator`: A `FactorySimulator` object that can be used to simulate what happens in the `Factory` assigned to this agent when given operations are conducted (e.g. production, paying money, etc).

**Negotiations/Contracts**

- `requested_negotiations`: A dynamic list of negotiations currently requested by the agent but not started. Correct management of this list is only possible if the agent **always** uses `request_negotiation` method of this class (see methods later) rather than directly calling request_method on the `SCMLAWI(awi)` member.
- `running_negotiations`: A dynamic list of negotiations currently running involving this agent. Correct management of this list is only possible if the agent **always** uses `request_negotiation` method of this class (see methods later) rather than directly calling request_method on the `SCMLAWI(awi)` member.
- `unsigned_contracts`: A dynamic list of negotiations contracts concluded involving this agent but not yet signed. Correct management of this list is only possible if the agent **always** uses `request_negotiation` method of this class (see methods later) rather than directly calling request_method on the `SCMLAWI(awi)` member.

**Simulation attributes** (also accessible through `awi`)

- `transportation_delay`: The transportation delay in the system.
• **current_step**: Current simulation step.
• **immediate_negotiations**: Whether or not negotiations start immediately upon registration (default is to start on the next production step)
• **negotiation_speed_multiple**: The number of negotiation rounds (steps) conducted in a single production step
• **transportation_delay**: Transportation delay in the system. Default is zero

**Methods** (Callable by the agent)

**Actions on the world**

• **request_negotiation**: Called to request a negotiation based on a CFP.

**Scheduling and simulation helpers**

• **can_expect_agreement**: Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable.

**Callbacks** (Callable by the simulation)

**Decision callbacks** (Called to make decisions)

• Negotiation and Contracts
  – **respond_to_negotiation_request**: Decide whether or not to engage in a negotiation on a CFP that was published earlier by this factory manager. If accepted, the agent should return a SAONegotiator object.
  – **sign_contract**: Decide whether or not to sign the contract. If accepted, the agent should return its own ID.
  – **confirm_contract_execution**: Decide whether or not to go on with executing a contract that the agent already signed. If rejected (by returning False), a refusal-to-execute breach will be recorded.

• Breach related
  – **confirm_partial_execution**: Decide whether the agent agrees to partial execution. Called only when the partner of this agent commits a partial breach (of level < 1) and this agent commits no breaches.
  – **set_renegotiation_agenda**: Decide what are the issues and ranges of acceptable values to re-negotiate about. Called only in case of breaches.
  – **respond_to_renegotiation_request**: Decide whether or not to engage in a re-negotiation.

• Financial
  – **confirm_loan**: Decide whether or not to accept an offered loan. *In ANAC 2019 league, loans are not allowed and this callback will never be called by the simulator.

**Time-dependent callbacks** (Information callback called at predefined times)

• **init**: Called once before any production or negotiations to initiate the agent.
• **step**: Called at every production step.

**Information callbacks** (Called to inform the agent about events)

• CFP related
  – **on_new_cfp**: Called whenever a CFP on a Product for which the agent has already registered interest (using register_interest method of its awi) is published. By default all agents register interest in the products they can consume or produce according to their profiles.
  – **on_remove_cfp**: Called whenever a CFP on a Product for which the agent has already registered interest (using register_interest method of its awi) is removed from the bulletin-board.
• Negotiation related
  – on_neg_request_accepted: Called when a negotiation request of the agent is accepted
  – on_neg_request_rejected: Called when a negotiation request of the agent is rejected
  – on_negotiation_success: Called when a negotiation of which the agent is a party succeeds with an agreement.
  – on_negotiation_failure: Called when a negotiation of which the agent is a party ends without agreement.

• Contract related
  – on_contract_cancelled: Called whenever a Contract of which the agent is a party is cancelled because the other party refused to sign it.
  – on_contract_signed: Called whenever a Contract of which the agent is a party is signed by both partners.
  – on_contract_nullified: Called whenever a Contract of which the agent is a party is nullified by the simulator as a part of bankruptcy processing.
  – on_contract_executed: Called when a contract executes completely and successfully.
  – on_contract_breached: Called when a contract is breached after complete contract processing.

• Production and factory related
  – on_production_failure: Called whenever a scheduled production (see SCMLAWI for production commands) fails
  – on_inventory_change: Called whenever there is a change in the inventory (something is moved in or out or out of storage due to an event other than production (e.g. contract execution).
  – on_cash_transfer: Called whenever cash is transferred to or from the factory’s wallet.

• About other agents
  – on_agent_bankrupt: Called whenever another agent goes bankrupt
  – on_new_report: Called whenever a new report of another agent for which this agent has registered interest is published. Interest is registered using the agent’s awi’s receive_financial_reports method.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_expect_agreement(cfp, margin)</td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable.</td>
</tr>
<tr>
<td>confirm_contract_execution(contract)</td>
<td>Called before executing any agreement.</td>
</tr>
<tr>
<td>confirm_loan(loan, bankrupt_if_rejected)</td>
<td>Called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached.</td>
</tr>
<tr>
<td>confirm_partial_execution(contract, breaches)</td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>from_config(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info.</td>
</tr>
<tr>
<td>init()</td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td>init()</td>
<td>The initialization function called by the world directly.</td>
</tr>
<tr>
<td>notify(notifiable, notification)</td>
<td></td>
</tr>
<tr>
<td>on_agent_bankrupt(agent_id)</td>
<td>Will be called whenever any agent goes bankrupt</td>
</tr>
<tr>
<td>on_cash_transfer(amount, cause)</td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td>on_contract_breached(contract, breaches, ...)</td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td>on_contract_cancelled(contract, rejectors)</td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td>on_contract_cancelled_(contract, rejectors)</td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td>on_contract_executed(contract)</td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td>on_contract_nullified(contract, ...)</td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.</td>
</tr>
<tr>
<td>on_contract_signed(contract)</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_contract_signed_(contract)</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_event(event, sender)</td>
<td></td>
</tr>
<tr>
<td>on_inventory_change(product, quantity, cause)</td>
<td>Received whenever something moves in or out of the factory’s storage.</td>
</tr>
<tr>
<td>on_neg_request_accepted(req_id, mechanism)</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>on_neg_request_accepted_(req_id, mechanism)</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>on_neg_request_rejected(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_neg_request_rejected_(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_negotiation_failure(partners, annotation, ...)</td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td>on_negotiation_failure_(partners, ...)</td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td>on_negotiation_success(contract, mechanism)</td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td>on_negotiation_success_(contract, mechanism)</td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td>on_new_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is published.</td>
</tr>
<tr>
<td>on_new_report(report)</td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>on_production_failure</td>
<td>Called with a list of \texttt{ProductionFailure} records on production failure.</td>
</tr>
<tr>
<td>on_production_success</td>
<td>Called with a list of \texttt{ProductionReport} records on production success.</td>
</tr>
<tr>
<td>on_remove_cfp</td>
<td>Called when a new CFP for a product for which the agent registered interest is removed.</td>
</tr>
<tr>
<td>read_config</td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td>request_negotiation</td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests.</td>
</tr>
<tr>
<td>respond_to_negotiation_request</td>
<td>Called when a prospective partner requests a negotiation to start.</td>
</tr>
<tr>
<td>respond_to_negotiation_request</td>
<td>Called when a negotiation request is received.</td>
</tr>
<tr>
<td>set_renegotiation_agenda</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td>sign_contract</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td>step()</td>
<td>Called by the simulator at every simulation step.</td>
</tr>
<tr>
<td>step()</td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- \texttt{awi}:
  Returns the Agent-World-Interface through which the agent does all of its actions in the world.

  A single exception is request\_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

  **Return type** \texttt{SCMLAWI}

- \texttt{id}:
  The unique ID of this entity

- \texttt{name}:
  A convenient name of the entity (intended primarily for printing/logging/debugging).

- \texttt{requested\_negotiations}:
  The negotiations currently requested by the agent.

  **Return type** \texttt{List[NegotiationRequestInfo]}

  **Returns** A list of negotiation request information objects (\texttt{NegotiationRequestInfo})

- \texttt{running\_negotiations}:
  The negotiations currently requested by the agent.

  **Return type** \texttt{List[RunningNegotiationInfo]}

  **Returns** A list of negotiation information objects (\texttt{RunningNegotiationInfo})

- \texttt{short\_type\_name}:
  Returns a short name of the type of this entity

- \texttt{type\_name}:
  Returns the name of the type of this entity

- \texttt{unsigned\_contracts}:
  All contracts that are not yet signed.
uuid
The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters
  • margin (int)
  • cfp (CFP)

Returns:

can_expect_agreement (cfp, margin)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters
  • margin (int)
  • cfp (CFP)

Returns:

confirm_contract_execution (contract)
Called before executing any agreement

Return type bool

confirm_loan (loan, bankrupt_if_rejected)
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

Return type bool

confirm_partial_execution (contract, breaches)
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters
  • contract (Contract) – The contract that was breached
  • breaches (List[Breach]) – A list of all the breaches committed.

Remarks:
  • Will not be called if both partners committed breaches.

Return type bool

classmethod create (*args, **kwargs)
Creates an object and returns a proxy to it.

classmethod from_config (config, section=None, ignore_children=True, try_parsing_children=True, scope=None)
Creates an object of this class given the configuration info

Parameters
  • config (Union[str, dict]) – Either a file name or a dictionary
  • section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params
  • ignore_children (bool) – If true then children will be ignored and there will be a single return
  • try_parsing_children (bool) – If true the children will first be parsed as ConfigReader classes if they are not
  • types (e.g. int, str, float, Iterable[int|str|float] (simple)
  • scope – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are

6.1. negmas.apps.scml Package 369
- any children that are to be parsed. (having) –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a
dictionary with children that were not parsed.

Remarks:

- This function will return an object of its class after passing the key-value pairs found in the config
to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**init()**

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**init_()**

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.
2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
3. registers interest in all products that the agent can produce or consume in its factory.
4. finally it calls any custom initialization logic implemented in ‘init’()

**See also:**

init, step

**notify (notifiable, notification)**

**on_agent_bankrupt (agent_id)**

Will be called whenever any agent goes bankrupt

**Parameters**

agent_id (str) – The ID of the agent that went bankrupt

Remarks:

- Agents can go bankrupt in two cases:
  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).
- All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

**Return type** None

**on_cash_transfer (amount, cause)**

Received whenever money is transferred to the factory or from it.

**Parameters**

- amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
- **cause** *(str)* – The cause of the change. Possibilities include:
  - contract: Contract execution
  - insurance: Received from insurance company
  - bankruptcy: Liquidated due to bankruptcy
  - transfer: Arrival of transferred money (when transfer delay in the system is > 0).

  **Return type** None

  **on_contract_breached** *(contract, breaches, resolution)*
  Called after complete processing of a contract that involved a breach.

  **Parameters**
  - **contract** *(Contract)* – The contract
  - **breaches** *(List[Breach])* – All breaches committed (even if they were resolved)
  - **resolution** *(Optional[Contract])* – The resolution contract if re-negotiation was successful. None if not.

  **Return type** None

  **on_contract_cancelled** *(contract, rejectors)*
  Called whenever at least a partner did not sign the contract

  **Return type** None

  **on_contract_cancelled_** *(contract, rejectors)*
  Called whenever at least a partner did not sign the contract

  **Return type** None

  **on_contract_executed** *(contract)*
  Called after successful contract execution for which the agent is one of the partners.

  **Return type** None

  **on_contract_nullified** *(contract, bankrupt_partner, compensation)*
  Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

  **Return type** None

  **on_contract_signed** *(contract)*
  Called whenever a contract is signed by all partners

  **Return type** None

  **on_contract_signed_** *(contract)*
  Called whenever a contract is signed by all partners

  **Return type** None

  **on_event** *(event, sender)*

  **on_inventory_change** *(product, quantity, cause)*
  Received whenever something moves in or out of the factory’s storage

  **Parameters**
  - **product** *(int)* – Product index.
  - **quantity** *(int)* – Negative value for products moving out and positive value for products moving in
  - **cause** *(str)* – The cause of the change. Possibilities include:
    - contract: Contract execution
    - insurance: Received from insurance company
bankruptcy: Liquidated due to bankruptcy
transport: Arrival of goods (when transportation delay in the system is > 0).

Return type None

on_neg_request_accepted(req_id, mechanism)
Called when a requested negotiation is accepted

Parameters

- req_id (str) – The request ID passed to _request_negotiation
- by (Optional[List[Union[str, None]]]) – A list of agents that refused to participate or None if the failure was for another reason

on_neg_request_rejected(req_id, by)
Called when a requested negotiation is rejected

Parameters

- req_id (str) – The request ID passed to _request_negotiation
- by (Optional[List[Union[str, None]]]) – A list of agents that refused to participate or None if the failure was for another reason

on_negotiation_failure(partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_success(contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

on_new_cfp(cfp)
Called when a new CFP for a product for which the agent registered interest is published

on_new_report(report)
Called whenever a financial report is published.

Parameters report (FinancialReport) – The financial report giving details of the standing of an agent at some time (see FinancialReport)

Remarks:
- Agents must opt-in to receive these calls by calling receive_financial_reports on their AWI

on_production_failure(failures)
Called with a list of ProductionFailure records on production failure.

Return type None
on_production_success (reports)
Called with a list of ProductionReport records on production success
Return type None

on_remove_cfp (cfp)
Called when a new CFP for a product for which the agent registered interest is removed

classmethod read_config (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing
Parameters
- config (Union[str, dict]) – Either a file name or a dictionary for loading params
- section (Optional[str]) – A section in the file or a key in the dictionary to use
Return type Dict[str, Any]
Returns A dict ready to be parsed by from_config
Remarks:

request_negotiation (cfp, negotiator=None, ufun=None)
Requests a negotiation from the AWI while keeping track of available negotiation requests
Parameters
- cfp (CFP)
- negotiator (Optional[Negotiator])
- ufun (Optional[UtilityFunction])
Return type bool
Returns Whether the negotiation request was successful indicating that the partner accepted the negotiation

respond_to_negotiation_request (cfp, partner)
Called when a prospective partner requests a negotiation to start
Return type Optional[Negotiator]

respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received
Return type Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)
Called to respond to a renegotiation request
Parameters
- agenda (RenegotiationRequest)
- contract (Contract)
- breaches (List[Breach])
Returns:
Return type Optional[Negotiator]

set_renegotiation_agenda (contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails
Parameters
- contract (Contract) – The contract being breached
• **breaches** (*List[Breach]*) – All breaches on contract

**Return type** Optional[RenegotiationRequest]

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

**sign_contract** (*contract*)

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** Optional[str]

**step**()

Called by the simulator at every simulation step

**step_**()

Called at every time-step. This function is called directly by the world.

### DoNothingFactoryManager

**class** negmas.apps.scml.DoNothingFactoryManager (*name=None*, *simulator_type=<class 'negmas.apps.scml.simulators.FastFactorySimulator'>*)

**Bases:** negmas.apps.scml.FactoryManager

The default factory manager that will be implemented by the committee of ANAC-SCML 2019

#### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

#### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_expect_agreement(cf, margin)</td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable</td>
</tr>
<tr>
<td>confirm_contract_execution(contract)</td>
<td>Called before executing any agreement</td>
</tr>
<tr>
<td>confirm_loan(loans, bankrupt_if_rejected)</td>
<td>called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached</td>
</tr>
<tr>
<td>confirm_partial_execution(contract, breaches)</td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>from_config(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td>init()</td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
</tbody>
</table>

Continued on next page
- \texttt{init()}\textemdash The initialization function called by the world directly.

- \texttt{notify(notifiable, notification)}\textemdash Will be called whenever any agent goes bankrupt.

- \texttt{on\_agent\_bankrupt(agent\_id)}\textemdash Received whenever money is transferred to the factory or from it.

- \texttt{on\_cash\_transfer(amount, cause)}\textemdash Called after complete processing of a contract that involved a breach.

- \texttt{on\_contract\_breached(contract, breaches, ...)}\textemdash Called whenever at least a partner did not sign the contract.

- \texttt{on\_contract\_cancelled(contract, rejects)}\textemdash Called whenever at least a partner did not sign the contract.

- \texttt{on\_contract\_executed(contract)}\textemdash Called after successful contract execution for which the agent is one of the partners.

- \texttt{on\_contract\_nullified(contract, ...)}\textemdash Called whenever a contract the agent is involved in is nullified because another partner went bankrupt.

- \texttt{on\_contract\_signed(contract)}\textemdash Called whenever a contract is signed by all partners.

- \texttt{on\_event(event, sender)}\textemdash Called whenever a contract is signed by all partners.

- \texttt{on\_inventory\_change(product, quantity, cause)}\textemdash Received whenever something moves in or out of the factory’s storage.

- \texttt{on\_neg\_request\_accepted(req\_id, mechanism)}\textemdash Called when a requested negotiation is accepted.

- \texttt{on\_neg\_request\_accepted_(req\_id, mechanism)}\textemdash Called when a requested negotiation is accepted.

- \texttt{on\_neg\_request\_rejected(req\_id, by)}\textemdash Called when a requested negotiation is rejected.

- \texttt{on\_neg\_request\_rejected_(req\_id, by)}\textemdash Called when a requested negotiation is rejected.

- \texttt{on\_neg\_registration\_failure(partners, annotation, ...)}\textemdash Called whenever a negotiation ends without agreement.

- \texttt{on\_neg\_registration\_failure_(partners, ...)}\textemdash Called whenever a negotiation ends without agreement.

- \texttt{on\_neg\_registration\_success(contract, mechanism)}\textemdash Called whenever a negotiation ends with agreement.

- \texttt{on\_neg\_registration\_success_(contract, mechanism)}\textemdash Called whenever a negotiation ends with agreement.

- \texttt{on\_new\_cfp(cfp)}\textemdash Called when a new CFP for a product for which the agent registered interest is published.

- \texttt{on\_new\_report(report)}\textemdash Called whenever a financial report is published.

- \texttt{on\_production\_failure(failures)}\textemdash Called with a list of \texttt{ProductionFailure} records on production failure.

- \texttt{on\_production\_success(reports)}\textemdash Called with a list of \texttt{ProductionReport} records on production success.

- \texttt{on\_remove\_cfp(cfp)}\textemdash Called when a new CFP for a product for which the agent registered interest is removed.

- \texttt{read\_config(config[, section])}\textemdash Reads the configuration from a file or a dict and prepares it for parsing.

- \texttt{request\_neg\_registration(cfpx[, negotiator, ufuns])}\textemdash Requests a negotiation from the AWI while keeping track of available negotiation requests.

- \texttt{respond\_to\_neg\_registration\_request(cfpx, partner)}\textemdash Called when a prospective partner requests a negotiation to start.

- \texttt{respond\_to\_neg\_registration\_request_(initiator, ...)}\textemdash Called when a negotiation request is received.

\textit{Continued on next page}
respond_to_renegotiation_request (contract, ... )

Called to respond to a renegotiation request

set_renegotiation_agenda (contract, breaches)

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

sign_contract (contract)

Called after the signing delay from contract conclusion to sign the contract.

step ()

Called by the simulator at every simulation step

step_ ()

Called at every time-step.

Attributes Documentation

awi

Returns the Agent-World-Interface through which the agent does all of its actions in the world.

A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

Return type SCMLAWI

id

The unique ID of this entity

name

A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations

The negotiations currently requested by the agent.

Return type List[NegotiationRequestInfo]

Returns A list of negotiation request information objects (NegotiationRequestInfo)

running_negotiations

The negotiations currently requested by the agent.

Return type List[RunningNegotiationInfo]

Returns A list of negotiation information objects (RunningNegotiationInfo)

short_type_name

Returns a short name of the type of this entity

type_name

Returns the name of the type of this entity

unsigned_contracts

All contracts that are not yet signed.

Return type List[Contract]

uuid

The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)

Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

  • margin (int) –
  • cfp (CFP) –

Returns:
**confirm_contract_execution** *(contract)*
Called before executing any agreement

**Return type** bool

**confirm_loan** *(loan, bankrupt_if_rejected)*
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

**Return type** bool

**confirm_partial_execution** *(contract, breaches)*
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

**Parameters**
- **contract** *(Contract)* – The contract that was breached
- **breaches** *(List[Breach])* – A list of all the breaches committed.

**Remarks:**
- Will not be called if both partners committed breaches.

**Return type** bool

**classmethod create** *(\*args, \*\*kwargs)*
Creates an object and returns a proxy to it.

**classmethod from_config** *(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)*
Creates an object of this class given the configuration info

**Parameters**
- **config** *(Union[\str, dict])* – Either a file name or a dictionary
- **section** *(Optional[\str])* – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return
- **try_parsing_children** *(bool)* – If true the children will first be parsed as ConfigReader classes if they are not types (e.g. int, str, float, Iterable[int|str|float]) *(simple)* –
- **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
- **any children that are to be parsed. (having)* –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

**Remarks:**
- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undeclared_name exception, the caller must pass her globals() as the scope.
init()
Called to initialize the agent after the world is initialized. The AWI is accessible at this point.

init_()
The initialization function called by the world directly.
It does the following actions by default:
1. copies some of the static world settings to the agent to make them available without calling the
   AWI.
2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
3. registers interest in all products that the agent can produce or consume in its factory.
4. finally it calls any custom initialization logic implemented in ‘init’()

See also:
init, step
notify (notifiable, notification)
on_agent_bankrupt (agent_id)
Will be called whenever any agent goes bankrupt

Parameters
agent_id (str) – The ID of the agent that went bankrupt

Remarks:
• Agents can go bankrupt in two cases:
  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get
     another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable
     to get a loan to pay for it).
• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not
   to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks
   by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)
Received whenever money is transferred to the factory or from it.

Parameters
• amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breached (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters
• contract (Contract) – The contract
• `breaches (List[Breach])` – All breaches committed (even if they were resolved)

• `resolution (Optional[Contract])` – The resolution contract if re-negotiation was successful. None if not.

Return type: `None`

`on_contract_cancelled(contract, rejectors)`
Called whenever at least a partner did not sign the contract

Return type: `None`

`on_contract_cancelled_(contract, rejectors)`
Called whenever at least a partner did not sign the contract

Return type: `None`

`on_contract_executed(contract)`
Called after successful contract execution for which the agent is one of the partners.

Return type: `None`

`on_contract_nullified(contract, bankrupt_partner, compensation)`
Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

Return type: `None`

`on_contract_signed(contract)`
Called whenever a contract is signed by all partners

Return type: `None`

`on_contract_signed_ (contract)`
Called whenever a contract is signed by all partners

Return type: `None`

`on_event (event, sender)`

`on_inventory_change (product, quantity, cause)`
Received whenever something moves in or out of the factory’s storage

Parameters

• `product (int)` – Product index.

• `quantity (int)` – Negative value for products moving out and positive value for products moving in

• `cause (str)` – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transport: Arrival of goods (when transportation delay in the system is > 0).

Return type: `None`

`on_neg_request_accepted(req_id, mechanism)`
Called when a requested negotiation is accepted

`on_neg_request_accepted_ (req_id, mechanism)`
Called when a requested negotiation is accepted

`on_neg_request_rejected(req_id, by)`
Called when a requested negotiation is rejected

Parameters
• **req_id** *(str)* – The request ID passed to _request_negotiation

• **by** *(Optional[List[str]])* – A list of agents that refused to participate or None if the failure was for another reason

**on_neg_request_rejected**( *req_id, by*)
Called when a requested negotiation is rejected

**Parameters**

• **req_id** *(str)* – The request ID passed to _request_negotiation

• **by** *(Optional[List[str]])* – A list of agents that refused to participate or None if the failure was for another reason

**on_negotiation_failure**( *partners, annotation, mechanism, state*)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_failure**( *partners, annotation, mechanism, state*)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_success**( *contract, mechanism*)
Called whenever a negotiation ends with agreement

**Return type** None

**on_negotiation_success**( *contract, mechanism*)
Called whenever a negotiation ends with agreement

**Return type** None

**on_new_cfp**( *cfp*)
Called when a new CFP for a product for which the agent registered interest is published

**Return type** None

**on_new_report**( *report*)
Called whenever a financial report is published.

**Parameters**


**Remarks:**

• Agents must opt-in to receive these calls by calling *receive_financial_reports* on their AWI

**on_production_failure**( *failures*)
Called with a list of *ProductionFailure* records on production failure.

**Return type** None

**on_production_success**( *reports*)
Called with a list of *ProductionReport* records on production success

**Return type** None

**on_remove_cfp**( *cfp*)
Called when a new CFP for a product for which the agent registered interest is removed

**Return type** None

**classmethod read_config**( *config, section=None*)
Reads the configuration from a file or a dict and prepares it for parsing

**Parameters**

• **config** *(Union[str, dict])* – Either a file name or a dictionary
• `section` (Optional[`str`]) – A section in the file or a key in the dictionary to use for loading params

**Return type** `Dict[`str`,`Any`]`

**Returns** A dict ready to be parsed by `from_config`

Remarks:

`request_negotiation` (`cfp`, `negotiator=None`, `ufun=None`)
Requests a negotiation from the AWI while keeping track of available negotiation requests

**Parameters**

• `cfp` (`CFP`) –
• `negotiator` (Optional[`Negotiator`]) –
• `ufun` (Optional[`UtilityFunction`]) –

**Return type** `bool`

**Returns** Whether the negotiation request was successful indicating that the partner accepted the negotiation

`respond_to_negotiation_request` (`cfp`, `partner`)
Called when a prospective partner requests a negotiation to start

**Return type** `Optional[`Negotiator`]`

`respond_to_negotiation_request_` (`initiator`, `partners`, `issues`, `annotation`, `mechanism`, `role`, `req_id`)
Called when a negotiation request is received

**Return type** `Optional[`Negotiator`]`

`respond_to_renegotiation_request` (`contract`, `breaches`, `agenda`)
Called to respond to a renegotiation request

**Parameters**

• `agenda` (`RenegotiationRequest`) –
• `contract` (`Contract`) –
• `breaches` (`List[Breach]`) –

**Returns:**

**Return type** `Optional[`Negotiator`]`

`set_renegotiation_agenda` (`contract`, `breaches`)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

• `contract` (`Contract`) – The contract being breached
• `breaches` (`List[Breach]`) – All breaches on `contract`

**Return type** `Optional[`RenegotiationRequest`]`

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

`sign_contract` (`contract`)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** `Optional[`str`]`

`step`()
Called by the simulator at every simulation step
step()
   Called at every time-step. This function is called directly by the world.

GreedyFactoryManager

class negmas.apps.scml.GreedyFactoryManager (name=None, simulator_type=<class 'negmas.apps.scml.simulators.FastFactorySimulator'>, scheduler_type=<class 'negmas.apps.scml.schedulers.GreedyScheduler'>, scheduler_params=None, optimism=0.0, negotiator_type='negmas.sao.AspirationNegotiator', negotiator_params=None, n_retrials=5, use_consumer=True, reactive=True, sign_only_guaranteed_contracts=False, riskiness=0.0, max_insurance_premium=0.1, reserved_value=None)

Bases: negmas.apps.scml.DoNothingFactoryManager

The default factory manager that will be implemented by the committee of ANAC-SCML 2019

Attributes Summary

awi
   Returns the Agent-World-Interface through which the agent does all of its actions in the world.

id
   The unique ID of this entity

name
   A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations
   The negotiations currently requested by the agent.

running_negotiations
   The negotiations currently requested by the agent.

short_type_name
   Returns a short name of the type of this entity

type_name
   Returns the name of the type of this entity

unsigned_contracts
   All contracts that are not yet signed.

uuid
   The unique ID of this entity

Methods Summary

can_expect_agreement(cfp, margin)
   Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

can_produce(cfp[,....])
   Whether or not we can produce the required item in time

can_secure_needs(schedule, step)
   Finds if it is possible in principle to arrange these needs at the given time.

confirm_contract_execution(contract)
   Called before executing any agreement

confirm_loan(loan, bankrupt_ifRejected)
   called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

confirm_partial_execution(contract, breaches)
   Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

create(*args, **kwargs)
   Creates an object and returns a proxy to it.
### Table 35 – continued from previous page

<table>
<thead>
<tr>
<th>Method/Publication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_config</code></td>
<td>(config[, section, ...]) Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><code>init_()</code></td>
<td>The initialization function called by the world directly.</td>
</tr>
<tr>
<td><code>notify(notifiable, notification)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_agent_bankrupt(agent_id)</code></td>
<td>Will be called whenever any agent goes bankrupt</td>
</tr>
<tr>
<td><code>on_cash_transfer(amount, cause)</code></td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td><code>on_contract_breached(contract, breaches, ...)</code></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><code>on_contract_cancelled(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><code>on_contract_cancelled_(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><code>on_contract_executed(contract)</code></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><code>on_contract_nullified(contract, ...)</code></td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt</td>
</tr>
<tr>
<td><code>on_contract_signed(contract)</code></td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td><code>on_contract_signed_(contract)</code></td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td><code>on_event(event, sender)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_inventory_change(product, quantity, cause)</code></td>
<td>Received whenever something moves in or out of the factory’s storage</td>
</tr>
<tr>
<td><code>on_neg_request_accepted(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><code>on_neg_request_accepted_(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><code>on_neg_request_rejected(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><code>on_neg_request_rejected_(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><code>on_negotiation_failure(partners, annotation, ...)</code></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><code>on_negotiation_failure_(partners, ...)</code></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><code>on_negotiation_success(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><code>on_negotiation_success_(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><code>on_new_cfp(cfp)</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is published</td>
</tr>
<tr>
<td><code>on_new_report(report)</code></td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td><code>on_production_failure(failures)</code></td>
<td>Called with a list of <code>ProductionFailure</code> records on production failure.</td>
</tr>
<tr>
<td><code>on_production_success(reports)</code></td>
<td>Called with a list of <code>ProductionReport</code> records on production success</td>
</tr>
<tr>
<td><code>on_remove_cfp(cfp)</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td><code>read_config(config[, section])</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td><code>request_negotiation(cfp[, negotiator, ufun])</code></td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests</td>
</tr>
</tbody>
</table>

Continued on next page
Table 35 – continued from previous page

<table>
<thead>
<tr>
<th>Function/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>respond_to_negotiation_request</code> (cfp, partner)</td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request_</code> (initiator, ... )</td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request</code> ... )</td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td><code>set_renegotiation_agenda</code> (contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>sign_contract</code> (contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step.</td>
</tr>
<tr>
<td><code>total_utility</code> ([contracts])</td>
<td>Calculates the total utility for the agent of a collection of contracts</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- **awi**
  
  Returns the Agent-World-Interface through which the agent does all of its actions in the world.
  
  A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

  **Return type** `SCMLAWI`

- **id**
  
  The unique ID of this entity

- **name**
  
  A convenient name of the entity (intended primarily for printing/logging/debugging).

- **requested_negotiations**
  
  The negotiations currently requested by the agent.

  **Return type** `List[NegotiationRequestInfo]`

  **Returns** A list of negotiation request information objects (`NegotiationRequestInfo`)

- **running_negotiations**
  
  The negotiations currently requested by the agent.

  **Return type** `List[RunningNegotiationInfo]`

  **Returns** A list of negotiation information objects (`RunningNegotiationInfo`)

- **short_type_name**
  
  Returns a short name of the type of this entity

- **type_name**
  
  Returns the name of the type of this entity

- **unsigned_contracts**
  
  All contracts that are not yet signed.

  **Return type** `List[Contract]`

- **uuid**
  
  The unique ID of this entity

**Methods Documentation**

- **can_expect_agreement** (`cfp, margin`)
  
  Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable
Parameters

- **margin** (int)
- **cfp** (CFP)

Returns:

**can_produce** *(cfp, assume_no_further_negotiations=False)*

Whether or not we can produce the required item in time

Return type **bool**

**can_secure_needs** *(schedule, step)*

Finds if it is possible in principle to arrange these needs at the given time.

Parameters

- **schedule** *(ScheduleInfo)*
- **step** (int)

Returns:

**confirm_contract_execution** *(contract)*

Called before executing any agreement

Return type **bool**

**confirm_loan** *(loan, bankrupt_if_rejected)*

called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

Return type **bool**

**confirm_partial_execution** *(contract, breaches)*

Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters

- **contract** *(Contract)* – The contract that was breached
- **breaches** *(List[Breach])* – A list of all the breaches committed.

Remarks:

- Will not be called if both partners committed breaches.

Return type **bool**

**classmethod create** *(args, **kwargs)*

Creates an object and returns a proxy to it.

**classmethod from_config** *(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)*

Creates an object of this class given the configuration info

Parameters

- **config** *(Union[str, dict])* – Either a file name or a dictionary
- **section** *(Optional[str])* – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return
- **try_parsing_children** *(bool)* – If true the children will first be parsed as ConfigReader classes if they are not
- **types** *(e.g. int, str, float, Iterable[int|str|float]} (simple)*
• **scope** – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are

• **any children that are to be parsed.** *(having)*—

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing `scope=globals()` to this function is to get around the fact that in python `eval()` will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when `eval()` is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an `undefined_name` exception, the caller must pass her `globals()` as the scope.

**init()**

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**init_()**

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.

2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.

3. registers interest in all products that the agent can produce or consume in its factory.

4. finally it calls any custom initialization logic implemented in `init()`

**See also:**

`init, step`

**notify (notifiable, notification)**

**on_agent_bankrupt (agent_id)**

Will be called whenever any agent goes bankrupt

**Parameters**

agent_id (str) – The ID of the agent that went bankrupt

**Remarks:**

• Agents can go bankrupt in two cases:

  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.

  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).

• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their `on_new_cfp` and `respond_to_negotiation_request` callbacks by pulling the bulletin-board using the helper function `is_bankrupt` of their AWI.

**Return type** None

**on_cash_transfer (amount, cause)**

Received whenever money is transferred to the factory or from it.

**Parameters**
• **amount** (*float*) – Amount of money (negative for transfers out of the factory, positive for transfers to it).

• **cause** (*str*) – The cause of the change. Possibilities include:
  - contract: Contract execution
  - insurance: Received from insurance company
  - bankruptcy: Liquidated due to bankruptcy
  - transfer: Arrival of transferred money (when transfer delay in the system is > 0).

**Return type** None

**on_contract_breached** (*contract, breaches, resolution*)

Called after complete processing of a contract that involved a breach.

**Parameters**

• **contract** (*Contract*) – The contract

• **breaches** (*List[Breach]*) – All breaches committed (even if they were resolved)

• **resolution** (*Optional[Contract]*) – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

**on_contract_cancelled** (*contract, rejectors*)

Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_cancelled_** (*contract, rejectors*)

Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_executed** (*contract*)

Called after successful contract execution for which the agent is one of the partners.

**Return type** None

**on_contract_nullified** (*contract, bankrupt_partner, compensation*)

Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

**Return type** None

**on_contract_signed** (*contract*)

Called whenever a contract is signed by all partners

**on_contract_signed_** (*contract*)

Called whenever a contract is signed by all partners

**Return type** None

**on_event** (*event, sender*)

**on_inventory_change** (*product, quantity, cause*)

Received whenever something moves in or out of the factory’s storage

**Parameters**

• **product** (*int*) – Product index.

• **quantity** (*int*) – Negative value for products moving out and positive value for products moving in

• **cause** (*str*) – The cause of the change. Possibilities include:
  - contract: Contract execution
- insurance: Received from insurance company
- bankruptcy: Liquidated due to bankruptcy
- transport: Arrival of goods (when transportation delay in the system is > 0).

**Return type** None

**on_neg_request_accepted**(req_id, mechanism)
Called when a requested negotiation is accepted

**on_neg_request_accepted_**(req_id, mechanism)
Called when a requested negotiation is accepted

**on_neg_request_rejected**(req_id, by)
Called when a requested negotiation is rejected

**Parameters**

- **req_id**(str) – The request ID passed to _request_negotiation
- **by**(Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

**on_neg_request_rejected_**(req_id, by)
Called when a requested negotiation is rejected

**Parameters**

- **req_id**(str) – The request ID passed to _request_negotiation
- **by**(Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

**on_negotiation_failure**(partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_failure_**(partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_success**(contract, mechanism)
Called whenever a negotiation ends with agreement

**on_negotiation_success_**(contract, mechanism)
Called whenever a negotiation ends with agreement

**Return type** None

**on_new_cfp**(cfp)
Called when a new CFP for a product for which the agent registered interest is published

**Return type** None

**on_new_report**(report)
Called whenever a financial report is published.

**Parameters**


**Remarks:**

- Agents must opt-in to receive these calls by calling receive_financial_reports on their AWI

**on_production_failure**(failures)
Called with a list of ProductionFailure records on production failure.

**Return type** None
on_production_success(reports)
Called with a list of ProductionReport records on production success

Return type None

on_remove_cfp(cfp)
Called when a new CFP for a product for which the agent registered interest is removed

Return type None

classmethod read_config(config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters
- config(Union[str, dict]) – Either a file name or a dictionary
- section(Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns: A dict ready to be parsed by from_config

Remarks:

request_negotiation(cfp, negotiator=None, ufun=None)
Requests a negotiation from the AWI while keeping track of available negotiation requests

Parameters
- cfp(CFP) –
- negotiator(Optional[Negotiator]) –
- ufun(Optional[UtilityFunction]) –

Return type bool

Returns: Whether the negotiation request was successful indicating that the partner accepted the negotiation

respond_to_negotiation_request(cfp, partner)
Called when a prospective partner requests a negotiation to start

Return type Optional[Negotiator]

respond_to_negotiation_request(initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received

Return type Optional[Negotiator]

respond_to_renegotiation_request(contract, breaches, agenda)
Called to respond to a renegotiation request

Parameters
- agenda(RenegotiationRequest) –
- contract(Contract) –
- breaches(List[Breach]) –

Returns:

Return type Optional[Negotiator]

set_renegotiation_agenda(contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters
NegMAS Documentation, Release 0.3.2

- **contract** *(Contract)* – The contract being breached
- **breaches** *(List[Breach])* – All breaches on *contract*

**Return type** Optional*[RenegotiationRequest]*

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

**sign_contract** *(contract)*
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**step()**
Called by the simulator at every simulation step

**step_()**
Called at every time-step. This function is called directly by the world.

**total_utility** *(contracts=())*
Calculates the total utility for the agent of a collection of contracts

**Return type** float

JavaFactoryManager

class negmas.apps.scml.JavaFactoryManager *(java_object=None, java_class_name=None, python_shadow=None, auto_load_java=False, name=None, simulator_type=<class 'negmas.apps.scml.simulators.FastFactorySimulator'>)*

**Bases**: negmas.apps.scml.FactoryManager, negmas.java.JavaCallerMixin

Allows factory managers implemented in Java (using jnegmas) to participate in SCML worlds.

Objects of this class is used to represent a java object to the python environment. This means that they MUST have the same interface as a python class (first class in the inheritance list). The JavaCallerMixin is used to enable it to connect to the java object it is representing.

**Attributes Summary**

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Overrides type name to give the internal java type name</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**
**can_expect_agreement**(cfp, margin) Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

**confirmContractExecution**(contract) Called before executing any agreement

**confirmLoan**(loan, bankrupt_if_rejected) called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

**confirmPartialExecution**(contract, breaches) Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

**create**(*args, **kwargs) Creates an object and returns a proxy to it.

**do_nothing_manager**() Creates an object of this class given the configuration info

**from_config**(config[, section, ...]) Creates a Python object representing the corresponding Java object

**getCompiledProfiles**()... Called to initialize the agent after the world is initialized.

**getConsuming**()... The initialization function called by the world directly.

**getJavaBridge**(java_object, java_class_name) initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**getName**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**getNegotiationRequests**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**getProcesses**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**getProducts**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**getRequestedNegotiations**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**getRunningNegotiations**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**greedy_manager**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**init**(agentId) initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**initPython**()... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**init_java_bridge**(java_object, java_class_name)... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**notify**(notifiable, notification)... initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

**onAgentBankrupt**(agentId) rtype None

**onCashTransfer**(amount, cause) rtype None

**onContractBreached**(contract, breaches, ...) rtype None

**onContractCancelled**(contract, rejectors) rtype None

**onContractExecuted**(contract) rtype None

**onContractNullified**(contract, ...) rtype None

**onContractSigned**(contract) Continued on next page
### Table 37 – continued from previous page

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Parameters</th>
<th>rtype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onInventoryChange</code></td>
<td><code>(product, quantity, cause)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onNegRequestAccepted</code></td>
<td><code>(req_id, mechanism)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onNegRequestRejected</code></td>
<td><code>(req_id, rejectors)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onNegotiationFailure</code></td>
<td><code>(partners, annotation, ...)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onNegotiationSuccess</code></td>
<td><code>(contract, mechanism)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onNewCFP</code></td>
<td><code>(cfp)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onNewReport</code></td>
<td><code>(report)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onProductionFailure</code></td>
<td><code>(failures)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onProductionSuccess</code></td>
<td><code>(reports)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>onRemoveCFP</code></td>
<td><code>(cfp)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_agent_bankrupt</code></td>
<td><code>(agent_id)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_cash_transfer</code></td>
<td><code>(amount, cause)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_breach</code></td>
<td><code>(contract, breaches, ...)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_cancelled</code></td>
<td><code>(contract, rejectors)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_cancelled_</code></td>
<td><code>(contract, rejectors)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_executed</code></td>
<td><code>(contract)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_nullified</code></td>
<td><code>(contract, ...)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_signed</code></td>
<td><code>(contract)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_contract_signed_</code></td>
<td><code>(contract)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_event</code></td>
<td><code>(event, sender)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_inventory_change</code></td>
<td><code>(product, quantity, cause)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_neg_request_accepted</code></td>
<td><code>(req_id, mechanism)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_neg_request_rejected</code></td>
<td><code>(req_id, by)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_neg_request_rejected_</code></td>
<td><code>(req_id, by)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_negotiation_failure</code></td>
<td><code>(partners, annotation, ...)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_negotiation_failure_</code></td>
<td><code>(partners, ...)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_negotiation_success</code></td>
<td><code>(contract, mechanism)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_negotiation_success_</code></td>
<td><code>(contract, mechanism)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_new_cfp</code></td>
<td><code>(cfp)</code></td>
<td>None</td>
</tr>
<tr>
<td><code>on_new_report</code></td>
<td><code>(report)</code></td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
Table 37 – continued from previous page

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_production_failure</code></td>
<td>Called with a list of <code>ProductionFailure</code> records on production failure.</td>
</tr>
<tr>
<td><code>on_production_success</code></td>
<td>Called with a list of <code>ProductionReport</code> records on production success.</td>
</tr>
<tr>
<td><code>on_remove_cfp</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td><code>read_config</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td><code>requestNegotiation</code></td>
<td>Called with a list of <code>ProductionFailure</code> records on production failure.</td>
</tr>
<tr>
<td><code>request_negotiation</code></td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests</td>
</tr>
<tr>
<td><code>respondToNegotiationRequest</code></td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request</code></td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request</code></td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td><code>setID</code></td>
<td></td>
</tr>
<tr>
<td><code>setName</code></td>
<td></td>
</tr>
<tr>
<td><code>setRenegotiationAgenda</code></td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>signContract</code></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td><code>stepPython()</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

**awi**

Returns the Agent-World-Interface through which the agent does all of its actions in the world.

A single exception is `request_negotiation` for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**requested_negotiations**

The negotiations currently requested by the agent.

**Return type** `List[NegotiationRequestInfo]`

**Returns** A list of negotiation request information objects (NegotiationRequestInfo)

**running_negotiations**

The negotiations currently requested by the agent.
NegMAS Documentation, Release 0.3.2

**Return type** List[RunningNegotiationInfo]

**Returns** A list of negotiation information objects (RunningNegotiationInfo)

**short_type_name**

Returns a short name of the type of this entity

**type_name**

Overrides type name to give the internal java type name

**unsigned_contracts**

All contracts that are not yet signed.

**uuid**

The unique ID of this entity

## Methods Documentation

**can_expect_agreement** (cfp, margin)

Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

**Parameters**

- **margin** (int)
- **cfp** (CFP)

**Returns:**

**confirmContractExecution** (contract)

**confirmLoan** (loan, bankruptIfRejected)

**confirmPartialExecution** (contract, breaches)

**confirm_contract_execution** (contract)

Called before executing any agreement

**Return type** bool

**confirm_loan** (loan, bankrupt_if_rejected)

called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

**Return type** bool

**confirm_partial_execution** (contract, breaches)

Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

**Parameters**

- **contract** (Contract) – The contract that was breached
- **breaches** (List[Breach]) – A list of all the breaches committed.

**Remarks:**

- Will not be called if both partners committed breaches.

**Return type** bool

**classmethod create** (*args, **kwargs)

Creates an object and returns a proxy to it.

**classmethod do_nothing_manager** ()
classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)

Creates an object of this class given the configuration info

Parameters

- **config** (*Union[str, dict]*) – Either a file name or a dictionary
- **section** (*Optional[str]*) – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** (*bool*) – If true then children will be ignored and there will be a single return
- **try_parsing_children** (*bool*) – If true the children will first be parsed as ConfigReader classes if they are not
- **types** (*e.g. int, str, float, Iterable[int|str|float]*) (*simple*) –
- **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
- **any children that are to be parsed** (*having*) –

Returns

An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

classmethod from_dict(java_object, *args, **kwargs)

Creates a Python object representing the corresponding Java object
getCompiledProfiles()
getConsuming()
getContracts()
getID()
getLineProfiles()
getName()
getNegotiationRequests()
getProcesses()
getProducing()
getProducts()
getAddressedNegotiations()
getRunningNegotiations()
classmethod greedy_manager()

init()

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.
initPython()

init_()

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.
2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
3. registers interest in all products that the agent can produce or consume in its factory.
4. finally it calls any custom initialization logic implemented in ‘init’()

See also:

init, step

init_java_bridge(java_object, java_class_name, auto_load_java=False, python_shadow_object=None)

initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

Parameters

• java_object – A java object that already exists of the correct type. If given no new objects will be created
• java_class_name (str) – The type of the Java object to be created
• auto_load_java (bool) – When true, a JVM will be automatically created (if one is not available)
• python_shadow_object (Optional[Any]) – A python object to shadow the java object. The object will just call the corresponding

• on this shadow object whenever it needs. (method) –

Remarks:

• sets a member called java_object that can be used to access the corresponding Java object crated
• if python_shadow_object is given, it must be an object of a type that has an internal class called Java which has a single member called ‘implements’ which is a list of one string element representing the Java interface being implemented (it must be either jnegmas.PyCallable or an extension of it).

notify (notifiable, notification)

onAgentBankrupt (agentId)

onCashTransfer (amount, cause)

Return type None

onContractBreached (contract, breaches, resolution)

Return type None

onContractCancelled (contract, rejectors)

onContractExecuted (contract)

Return type None

onContractNullified (contract, bankruptPartner, compensation)

onContractSigned (contract)

onInventoryChange (product, quantity, cause)
Return type None

onNegRequestAccepted (req_id, mechanism)
onNegRequestRejected (req_id, rejectors)
onNegotiationFailure (partners, annotation, mechanism, state)
onNegotiationSuccess (contract, mechanism)
onNewCFP (cfp)
onNewReport (report)
onProductionFailure (failures)
onProductionSuccess (reports)

Return type None

onRemoveCFP (cfp)
on_agent_bankrupt (agent_id)
Will be called whenever any agent goes bankrupt
Parameters agent_id (str) – The ID of the agent that went bankrupt
Remarks:
• Agents can go bankrupt in two cases:
  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).
• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)
Received whenever money is transferred to the factory or from it.
Parameters
• amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breached (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.
Parameters
• contract (Contract) – The contract
• breaches (List[Breach]) – All breaches committed (even if they were resolved)
• **resolution** *(Optional[Contract])* – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

**on_contract_cancelled** *(contract, rejectors)*
Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_cancelled_** *(contract, rejectors)*
Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_executed** *(contract)*
Called after successful contract execution for which the agent is one of the partners.

**Return type** None

**on_contract_nullified** *(contract, bankrupt_partner, compensation)*
Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

**Return type** None

**on_contract_signed** *(contract)*
Called whenever a contract is signed by all partners

**Return type** None

**on_contract_signed_** *(contract)*
Called whenever a contract is signed by all partners

**Return type** None

**on_event** *(event, sender)*

**on_inventory_change** *(product, quantity, cause)*
Received whenever something moves in or out of the factory’s storage

**Parameters**

• **product** *(int)* – Product index.

• **quantity** *(int)* – Negative value for products moving out and positive value for products moving in

• **cause** *(str)* – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transport: Arrival of goods (when transportation delay in the system is > 0).

**Return type** None

**on_neg_request_accepted** *(req_id, mechanism)*
Called when a requested negotiation is accepted

**on_neg_request_accepted_** *(req_id, mechanism)*
Called when a requested negotiation is accepted

**on_neg_request_rejected** *(req_id, by)*
Called when a requested negotiation is rejected

**Parameters**

• **req_id** *(str)* – The request ID passed to _request_negotiation
on_neg_request_rejected_ (req_id, by)
Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation

• by (Optional[List[str]]) – A list of agents that refused to participate or None if
  the failure was for another reason

on_negotiation_failure (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_failure_ (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_success (contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

on_negotiation_success_ (contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

on_new_cfp (cfp)
Called when a new CFP for a product for which the agent registered interest is published

on_new_report (report)
Called whenever a financial report is published.

Parameters report (FinancialReport) – The financial report giving details of the
standing of an agent at some time (see FinancialReport)

Remarks:

• Agents must opt-in to receive these calls by calling receive_financial_reports on their
  AWI

on_production_failure (failures)
Called with a list of ProductionFailure records on production failure.

Return type None

on_production_success (reports)
Called with a list of ProductionReport records on production success

Return type None

on_remove_cfp (cfp)
Called when a new CFP for a product for which the agent registered interest is removed

classmethod read_config (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• config (Union[str, dict]) – Either a file name or a dictionary

• section (Optional[str]) – A section in the file or a key in the dictionary to use
  for loading params

Return type Dict[str, Any]
Returns A dict ready to be parsed by from_config

Remarks:

**requestNegotiation** *(cfp, negotiator=None, ufun=None)*

**Return type** bool

**request_negotiation** *(cfp, negotiator=None, ufun=None)*

Requests a negotiation from the AWI while keeping track of available negotiation requests

**Parameters**

- **cfp** *(CFP)*
- **negotiator** *(Optional[Negotiator])*  
- **ufun** *(Optional[UtilityFunction])*  

**Return type** bool

**Returns** Whether the negotiation request was successful indicating that the partner accepted the negotiation

**respondToNegotiationRequest** *(cfp, partner)*

**respondToRenegotiationRequest** *(contract, breaches, agenda)*

**respond_to_negotiation_request** *(cfp, partner)*

Called when a prospective partner requests a negotiation to start

**Return type** Optional[Negotiator]

**respond_to_negotiation_request_(initiator, partners, issues, annotation, mechanism, role, req_id)*

Called when a negotiation request is received

**Return type** Optional[Negotiator]

**respond_to_renegotiation_request** *(contract, breaches, agenda)*

Called to respond to a renegotiation request

**Parameters**

- **agenda** *(RenegotiationRequest)*
- **contract** *(Contract)*
- **breaches** *(List[Breach])*  

**Returns**:

**Return type** Optional[Negotiator]

**setID** *(value)*

**setName** *(value)*

**setRenegotiationAgenda** *(contract, breaches)*

**set_renegotiation_agenda** *(contract, breaches)*

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- **contract** *(Contract)* – The contract being breached
- **breaches** *(List[Breach])* – All breaches on contract

**Return type** Optional[RenegotiationRequest]

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

**signContract** *(contract)*
**sign_contract** (*contract*)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type  Optional[str]

**step()**
Called by the simulator at every simulation step

**stepPython()**

**step()**
Called at every time-step. This function is called directly by the world.

### JavaDoNothingFactoryManager

class negmas.apps.scml.JavaDoNothingFactoryManager (auto_load_java=False,  
name=None,  
simulator_type=<class 'negmas.apps.scml.simulators.FastFactorySimulator'>)

Bases: negmas.apps.scml.JavaFactoryManager

### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Overrides type name to give the internal java type name</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_expect_agreement(cfp, margin)</td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable</td>
</tr>
<tr>
<td>confirmContractExecution(contract)</td>
<td>Called before executing any agreement</td>
</tr>
<tr>
<td>confirmLoan(loan, bankrupt_if_rejected)</td>
<td>called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached</td>
</tr>
<tr>
<td>confirmPartialExecution(contract, breaches)</td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
<tr>
<td>confirm_contract_execution(contract)</td>
<td>Called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached</td>
</tr>
<tr>
<td>confirm_loan(loan, bankrupt_if_rejected)</td>
<td>Created an object and returns a proxy to it.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td>from_config(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
</tbody>
</table>

Continued on next page
Table 39 – continued from previous page

<table>
<thead>
<tr>
<th>Method/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_dict(java_object, *args, **kwargs)</code></td>
<td>Creates a Python object representing the corresponding Java object</td>
</tr>
<tr>
<td><code>getCompiledProfiles()</code></td>
<td></td>
</tr>
<tr>
<td><code>getConsuming()</code></td>
<td></td>
</tr>
<tr>
<td><code>getContracts()</code></td>
<td></td>
</tr>
<tr>
<td><code>getID()</code></td>
<td></td>
</tr>
<tr>
<td><code>getLineProfiles()</code></td>
<td></td>
</tr>
<tr>
<td><code>getName()</code></td>
<td></td>
</tr>
<tr>
<td><code>getNegotiationRequests()</code></td>
<td></td>
</tr>
<tr>
<td><code>getProcesses()</code></td>
<td></td>
</tr>
<tr>
<td><code>getProducing()</code></td>
<td></td>
</tr>
<tr>
<td><code>getProducts()</code></td>
<td></td>
</tr>
<tr>
<td><code>getRequestedNegotiations()</code></td>
<td></td>
</tr>
<tr>
<td><code>getRunningNegotiations()</code></td>
<td></td>
</tr>
<tr>
<td><code>greedy_manager()</code></td>
<td></td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><code>initPython()</code></td>
<td></td>
</tr>
<tr>
<td><code>init_()</code></td>
<td>The initialization function called by the world directly.</td>
</tr>
<tr>
<td><code>init_java_bridge(java_object, java_class_name)</code></td>
<td>initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java</td>
</tr>
<tr>
<td><code>notify(notifiable, notification)</code></td>
<td></td>
</tr>
<tr>
<td><code>onAgentBankrupt(agentId)</code></td>
<td></td>
</tr>
<tr>
<td><code>onCashTransfer(amount, cause)</code></td>
<td><code>rtype None</code></td>
</tr>
<tr>
<td><code>onContractBreached(contract, breaches, ...)</code></td>
<td><code>rtype None</code></td>
</tr>
<tr>
<td><code>onContractCancelled(contract, rejectors)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractExecuted(contract)</code></td>
<td><code>rtype None</code></td>
</tr>
<tr>
<td><code>onContractNullified(contract,...)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractSigned(contract)</code></td>
<td></td>
</tr>
<tr>
<td><code>onInventoryChange(product, quantity, cause)</code></td>
<td><code>rtype None</code></td>
</tr>
<tr>
<td><code>onNegRequestAccepted(req_id, mechanism)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegRequestRejected(req_id, rejectors)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegotiationFailure(partners, annotation,...)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegotiationSuccess(contract, mechanism)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNewCFP(cfp)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNewReport(report)</code></td>
<td></td>
</tr>
<tr>
<td><code>onProductionFailure(failures)</code></td>
<td></td>
</tr>
<tr>
<td><code>onProductionSuccess(reports)</code></td>
<td><code>rtype None</code></td>
</tr>
<tr>
<td><code>onRemoveCFP(cfp)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_agent_bankrupt(agent_id)</code></td>
<td>Will be called whenever any agent goes bankrupt</td>
</tr>
</tbody>
</table>
Table 39 – continued from previous page

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_cash_transfer(amount, cause)</td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td>on_contract_breached(contract, breaches, ...)</td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td>on_contract_cancelled(contract, rejectors)</td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td>on_contract_cancelled_(contract, rejectors)</td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td>on_contract_executed(contract)</td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td>on_contract_nullified(contract, ...)</td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt</td>
</tr>
<tr>
<td>on_contract_signed(contract)</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_contract_signed_(contract)</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_event(event, sender)</td>
<td></td>
</tr>
<tr>
<td>on_inventory_change(product, quantity, cause)</td>
<td>Received whenever something moves in or out of the factory’s storage</td>
</tr>
<tr>
<td>on_neg_request_accepted(req_id, mechan-</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>ism)</td>
<td></td>
</tr>
<tr>
<td>on_neg_request_accepted_(req_id, mech-</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>anism)</td>
<td></td>
</tr>
<tr>
<td>on_neg_request_rejected(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_neg_request_rejected_(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_negotiation_failure(partners, anno-</td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td>tation, ...)</td>
<td></td>
</tr>
<tr>
<td>on_negotiation_failure_(partners, ...)</td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td>on_negotiation_success(contract, mech-</td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td>anism)</td>
<td></td>
</tr>
<tr>
<td>on_negotiation_success_(contract, mech-</td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td>anism)</td>
<td></td>
</tr>
<tr>
<td>on_new_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is published</td>
</tr>
<tr>
<td>on_new_report(report)</td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td>on_production_failure(failures)</td>
<td>Called with a list of ProductionFailure records on production failure.</td>
</tr>
<tr>
<td>on_production_success(reports)</td>
<td>Called with a list of ProductionReport records on production success</td>
</tr>
<tr>
<td>on_remove_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td>read_config(config[, section])</td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td>requestNegotiation(cfp[, negotiator,</td>
<td>rtype bool</td>
</tr>
<tr>
<td>ufun])</td>
<td></td>
</tr>
<tr>
<td>request_negotiation(cfp[, negotiator,</td>
<td>Requests a negotiation from the AWI while keeping track of available negotia-</td>
</tr>
<tr>
<td>ufun])</td>
<td>tion requests</td>
</tr>
<tr>
<td>respondToNegotiationRequest(cfp, part-</td>
<td></td>
</tr>
<tr>
<td>ner)</td>
<td></td>
</tr>
<tr>
<td>respondToRenegotiationRequest(contract,</td>
<td></td>
</tr>
<tr>
<td>...)</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
### Table 39 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respond_to_negotiation_request(cfp, partner)</td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td>respond_to_negotiation_request_(initiator, ...)</td>
<td>(initially called when a negotiation request is received)</td>
</tr>
<tr>
<td>respond_to_renegotiation_request(...)</td>
<td>(called to respond to a renegotiation request)</td>
</tr>
<tr>
<td>setID(value)</td>
<td></td>
</tr>
<tr>
<td>setName(value)</td>
<td></td>
</tr>
<tr>
<td>setRenegotiationAgenda(contract, breaches)</td>
<td></td>
</tr>
<tr>
<td>set_renegotiation_agenda(contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td>signContract(contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td>sign_contract(contract)</td>
<td></td>
</tr>
<tr>
<td>step()</td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td>stepPython()</td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

#### Attributes Documentation

**awi**

Returns the Agent-World-Interface through which the agent does all of its actions in the world. A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**requested_negotiations**

The negotiations currently requested by the agent.

*Return type* List[NegotiationRequestInfo]

*Returns* A list of negotiation request information objects (NegotiationRequestInfo)

**running_negotiations**

The negotiations currently requested by the agent.

*Return type* List[RunningNegotiationInfo]

*Returns* A list of negotiation information objects (RunningNegotiationInfo)

**short_type_name**

Returns a short name of the type of this entity

**type_name**

Overrides type name to give the internal java type name

**unsigned_contracts**

All contracts that are not yet signed.

*Return type* List[Contract]

**uuid**

The unique ID of this entity
Methods Documentation

**can_expect_agreement(cfp, margin)**
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

• **margin** (int) –
• **cfp** (CFP) –

Returns:

**confirmContractExecution(contract)**
**confirmLoan(loan, bankruptIfRejected)**
**confirmPartialExecution(contract, breaches)**

**confirm_contract_execution(contract)**
Called before executing any agreement

Return type bool

**confirm_loan(loan, bankrupt_if_rejected)**
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

Return type bool

**confirm_partial_execution(contract, breaches)**
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters

• **contract** (Contract) – The contract that was breached
• **breaches** (List[Breach]) – A list of all the breaches committed.

Remarks:

• Will not be called if both partners committed breaches.

Return type bool

**classmethod create(*args, **kwargs)**
Creates an object and returns a proxy to it.

**classmethod do_nothing_manager()**

**classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)**
Creates an object of this class given the configuration info

Parameters

• **config** (Union[str, dict]) – Either a file name or a dictionary
• **section** (Optional[str]) – A section in the file or a key in the dictionary to use for loading params
• **ignore_children** (bool) – If true then children will be ignored and there will be a single return
• **try_parsing_children** (bool) – If true the children will first be parsed as ConfigReader classes if they are not
• **types** (e.g. int, str, float, Iterable[int,str,float], simple) –
**scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are any children that are to be parsed. (having) –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

**Remarks:**

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.

- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**classmethod from_dict (java_object, *args, **kwargs)**

Creates a Python object representing the corresponding Java object

**getCompiledProfiles ()**

**getConsuming ()**

**getContracts ()**

**getID ()**

**getLineProfiles ()**

**getName ()**

**getNegotiationRequests ()**

**getProcesses ()**

**getProducing ()**

**getProducts ()**

**getRequestedNegotiations ()**

**getRunningNegotiations ()**

**classmethod greedy_manager ()**

**init ()**

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**initPython ()**

**init_ ()**

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.

2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.

3. registers interest in all products that the agent can produce or consume in its factory.

4. finally it calls any custom initialization logic implemented in ‘init’()

**See also:**

*init, step*
Initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java.

**Parameters**

- **java_object** – A java object that already exists of the correct type. If given no new objects will be created.
- **java_class_name** (str) – The type of the Java object to be created.
- **auto_load_java** (bool) – When true, a JVM will be automatically created (if one is not available).
- **python_shadow_object** (Optional[python_shadow_object]) – A python object to shadow the java object. The object will just call the corresponding.

**Remarks:**

- sets a member called java_object that can be used to access the corresponding Java object crated
- if python_shadow_object is given, it must be an object of a type that has an internal class called Java which has a single member called ‘implements’ which is a list of one string element representing the Java interface being implemented (it must be either jnegmas.PyCallable or an extension of it).

**Methods**

- **notify** (notifiable, notification)
- **onAgentBankrupt** (agentId)
- **onCashTransfer** (amount, cause)
  
  **Return type** None

- **onContractBreached** (contract, breaches, resolution)
  
  **Return type** None

- **onContractCancelled** (contract, rejectors)

- **onContractExecuted** (contract)
  
  **Return type** None

- **onContractNullified** (contract, bankruptPartner, compensation)

- **onContractSigned** (contract)

- **onInventoryChange** (product, quantity, cause)
  
  **Return type** None

- **onNegRequestAccepted** (req_id, mechanism)

- **onNegRequestRejected** (req_id, rejectors)

- **onNegotiationFailure** (partners, annotation, mechanism, state)

- **onNegotiationSuccess** (contract, mechanism)

- **onNewCFP** (cfp)

- **onNewReport** (report)

- **onProductionFailure** (failures)

- **onProductionSuccess** (reports)

  **Return type** None

- **onRemoveCFP** (cfp)
on_agent_bankrupt (agent_id)
Will be called whenever any agent goes bankrupt

Parameters agent_id (str) – The ID of the agent that went bankrupt

Remarks:
• Agents can go bankrupt in two cases:
  1. Failing to pay one installment of a loan they bought and refusing (or being unable to) get another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).
• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)
Received whenever money is transferred to the factory or from it.

Parameters
• amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breached (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters
• contract (Contract) – The contract
• breaches (List[Breach]) – All breaches committed (even if they were resolved)
• resolution (Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

on_contract_cancelled (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

on_contract_cancelled_ (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

on_contract_executed (contract)
Called after successful contract execution for which the agent is one of the partners.

Return type None
on_contract_nullified (contract, bankrupt_partner, compensation)

Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

Return type None

on_contract_signed (contract)

Called whenever a contract is signed by all partners

Return type None

on_contract_signed_ (contract)

Called whenever a contract is signed by all partners

Return type None

on_event (event, sender)

on_inventory_change (product, quantity, cause)

Received whenever something moves in or out of the factory's storage

Parameters

• product (int) – Product index.
• quantity (int) – Negative value for products moving out and positive value for products moving in
• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transport: Arrival of goods (when transportation delay in the system is > 0).

Return type None

on_neg_request_accepted (req_id, mechanism)

Called when a requested negotiation is accepted

on_neg_request_accepted_ (req_id, mechanism)

Called when a requested negotiation is accepted

on_neg_request_rejected (req_id, by)

Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_neg_request_rejected_ (req_id, by)

Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_negotiation_failure (partners, annotation, mechanism, state)

Called whenever a negotiation ends without agreement

Return type None

on_negotiation_failure_ (partners, annotation, mechanism, state)

Called whenever a negotiation ends without agreement
Return type None

on_negotiation_success (contract, mechanism)
    Called whenever a negotiation ends with agreement

Return type None

on_negotiation_success_(contract, mechanism)
    Called whenever a negotiation ends with agreement

Return type None

on_new_cfp (cfp)
    Called when a new CFP for a product for which the agent registered interest is published

on_new_report (report)
    Called whenever a financial report is published.

Parameters report (FinancialReport) – The financial report giving details of the standing of an agent at some time (see FinancialReport)

Remarks:

• Agents must opt-in to receive these calls by calling receive_financial_reports on their AWI

on_production_failure (failures)
    Called with a list of ProductionFailure records on production failure.

Return type None

on_production_success (reports)
    Called with a list of ProductionReport records on production success

Return type None

on_remove_cfp (cfp)
    Called when a new CFP for a product for which the agent registered interest is removed

classmethod read_config (config, section=None)
    Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:

requestNegotiation (cfp, negotiator=None, ufun=None)

Return type bool

request_negotiation (cfp, negotiator=None, ufun=None)
    Requests a negotiation from the AWI while keeping track of available negotiation requests

Parameters

• cfp (CFP) –
• negotiator (Optional[Negotiator]) –
• ufun (Optional[UtilityFunction]) –

Return type bool
**Returns** Whether the negotiation request was successful indicating that the partner accepted the negotiation

```python
respondToNegotiationRequest (cfp, partner)
```

Called when a prospective partner requests a negotiation to start

```python
respondToRenegotiationRequest (contract, breaches, agenda)
```

Called to respond to a renegotiation request

**Parameters**

- **agenda** (`RenegotiationRequest`) –
- **contract** (`Contract`) –
- **breaches** (`List[Breach]`) –

Returns:

```python
respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)
```

Called when a negotiation request is received

```python
respond_to_renegotiation_request (contract, breaches, agenda)
```

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- **contract** (`Contract`) – The contract being breached
- **breaches** (`List[Breach]`) – All breaches on `contract`

Returns: `Optional[RenegotiationRequest]`

```python
setID (value)
```

```python
setName (value)
```

```python
setRenegotiationAgenda (contract, breaches)
```

```python
set_renegotiation_agenda (contract, breaches)
```

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

```python
signContract (contract)
```

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

```python
step ()
```

Called by the simulator at every simulation step

```python
stepPython ()
```

Called at every time-step. This function is called directly by the world.
JavaGreedyFactoryManager

```python
class negmas.apps.scml.JavaGreedyFactoryManager (auto_load_java=False, name=None, simulator_type=<class 'negmas.apps.scml.simulators.FastFactorySimulator'>)

Bases: negmas.apps.scml.JavaFactoryManager
```

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Overrides type name to give the internal java type name</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_expect_agreement(cfp, margin)</td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable.</td>
</tr>
<tr>
<td>confirmContractExecution(contract)</td>
<td>Called before executing any agreement</td>
</tr>
<tr>
<td>confirmLoan(loan, bankruptIfRejected)</td>
<td>Called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached.</td>
</tr>
<tr>
<td>confirmPartialExecution(contract, breaches)</td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>do_nothing_manager()</td>
<td></td>
</tr>
<tr>
<td>from_config(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td>from_dict(java_object, *args, **kwargs)</td>
<td>Creates a Python object representing the corresponding Java object</td>
</tr>
<tr>
<td>getCompiledProfiles()</td>
<td></td>
</tr>
<tr>
<td>getConsuming()</td>
<td></td>
</tr>
<tr>
<td>getContracts()</td>
<td></td>
</tr>
<tr>
<td>getID()</td>
<td></td>
</tr>
<tr>
<td>getLineProfiles()</td>
<td></td>
</tr>
<tr>
<td>getName()</td>
<td></td>
</tr>
<tr>
<td>getNegotiationRequests()</td>
<td></td>
</tr>
<tr>
<td>getProcesses()</td>
<td></td>
</tr>
<tr>
<td>getProducing()</td>
<td></td>
</tr>
<tr>
<td>getProducts()</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getRequestedNegotiations()</code></td>
<td>Called to initialize the agent <strong>after</strong> the world is initialized.</td>
</tr>
<tr>
<td><code>getRunningNegotiations()</code></td>
<td></td>
</tr>
<tr>
<td><code>greedy_manager()</code></td>
<td></td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Called to initialize the agent <strong>after</strong> the world is initialized.</td>
</tr>
<tr>
<td><code>initPython()</code></td>
<td>The initialization function called by the world directly.</td>
</tr>
<tr>
<td><code>init_java_bridge(java_object, java_class_name)</code></td>
<td>initializes a connection to the java bridge creating a member called <code>java_object</code> that can be used to access the counterpart object in Java</td>
</tr>
<tr>
<td><code>notify(notifiable, notification)</code></td>
<td></td>
</tr>
<tr>
<td><code>onAgentBankrupt(agentId)</code></td>
<td></td>
</tr>
<tr>
<td><code>onCashTransfer(amount, cause)</code></td>
<td><strong>rtype</strong>: <code>None</code></td>
</tr>
<tr>
<td><code>onContractBreached(contract, breaches, ...)</code></td>
<td><strong>rtype</strong>: <code>None</code></td>
</tr>
<tr>
<td><code>onContractCancelled(contract, rejectors)</code></td>
<td><strong>rtype</strong>: <code>None</code></td>
</tr>
<tr>
<td><code>onContractExecuted(contract)</code></td>
<td><strong>rtype</strong>: <code>None</code></td>
</tr>
<tr>
<td><code>onContractNullified(contract, ...)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractSigned(contract)</code></td>
<td></td>
</tr>
<tr>
<td><code>onInventoryChange(product, quantity, cause)</code></td>
<td><strong>rtype</strong>: <code>None</code></td>
</tr>
<tr>
<td><code>onNegRequestAccepted(req_id, mechanism)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegRequestRejected(req_id, rejectors)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegotiationFailure(partners, annotation, ...)</code></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><code>onNegotiationSuccess(contract, mechanism)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNewCFP(cfp)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNewReport(report)</code></td>
<td></td>
</tr>
<tr>
<td><code>onProductionFailure(failures)</code></td>
<td></td>
</tr>
<tr>
<td><code>onProductionSuccess(reports)</code></td>
<td></td>
</tr>
<tr>
<td><code>onRemoveCFP(cfp)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_agent_bankrupt(agent_id)</code></td>
<td>Will be called whenever any agent goes bankrupt</td>
</tr>
<tr>
<td><code>on_cash_transfer(amount, cause)</code></td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td><code>on_contract_breached(contract, breaches, ...)</code></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><code>on_contract_cancelled(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><code>on_contract_cancelled_(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><code>on_contract_executed(contract)</code></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_contract_nullified( contract, ... )</td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt</td>
</tr>
<tr>
<td>on_contract_signed( contract )</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_contract_signed_( contract )</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_event(event, sender)</td>
<td></td>
</tr>
<tr>
<td>on_inventory_change( product, quantity, cause )</td>
<td>Received whenever something moves in or out of the factory’s storage</td>
</tr>
<tr>
<td>on_neg_request_accepted(req_id, mechanism)</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>on_neg_request_accepted_( req_id, mechanism )</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>on_neg_request_rejected(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_neg_request_rejected_( req_id, by )</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_negotiation_failure(partners, annotation,...)</td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td>on_negotiation_failure_(partners, ...)</td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td>on_negotiation_success(contract, mechanism)</td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td>on_negotiation_success_(contract, mechanism)</td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td>on_new_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is published</td>
</tr>
<tr>
<td>on_new_report(report)</td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td>on_production_failure(failures)</td>
<td>Called with a list of ProductionFailure records on production failure.</td>
</tr>
<tr>
<td>on_production_success(reports)</td>
<td>Called with a list of ProductionReport records on production success.</td>
</tr>
<tr>
<td>on_remove_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td>read_config(config[, section])</td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td>requestNegotiation(cfp[, negotiator, ufun])</td>
<td>category: bool</td>
</tr>
<tr>
<td>request_negotiation(cfp[, negotiator, ufun])</td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests</td>
</tr>
<tr>
<td>respondToNegotiationRequest(cfp, partner)</td>
<td></td>
</tr>
<tr>
<td>respondToRenegotiationRequest(contract, ...)</td>
<td></td>
</tr>
<tr>
<td>respond_to_negotiation_request(cfp, partner)</td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td>respond_to_negotiation_request_(initiator)</td>
<td>(initially called when a negotiation request is received)</td>
</tr>
<tr>
<td>respond_to_renegotiation_request(... )</td>
<td>(continuously called to respond to a renegotiation request)</td>
</tr>
<tr>
<td>setID(value)</td>
<td></td>
</tr>
<tr>
<td>setName(value)</td>
<td></td>
</tr>
<tr>
<td>setRenegotiationAgenda(contract, breaches)</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
set_renegotiation_agenda(contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

signContract(contract)

sign_contract(contract)
Called after the signing delay from contract conclusion to sign the contract.

step() Called by the simulator at every simulation step

stepPython() Called at every time-step.

Attributes Documentation

awi
Returns the Agent-World-Interface through which the agent does all of its actions in the world.
A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations
The negotiations currently requested by the agent.

RunningNegotiationInfo

running_negotiations
The negotiations currently requested by the agent.

RunningNegotiationInfo

short_type_name
Returns a short name of the type of this entity

type_name
Overrides type name to give the internal java type name

unsigned_contracts
All contracts that are not yet signed.

Contract

uuid
The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

• margin (int)
• cfp (CFP)

Returns:
confirmContractExecution\(\{contract\}\)

confirmLoan\(\{loan, bankruptIfRejected\}\)

confirmPartialExecution\(\{contract, breaches\}\)

**confirm_contract_execution\(\{contract\}\)**

Called before executing any agreement

**Return type** bool

**confirm_loan\(\{loan, bankrupt_if_rejected\}\)**

called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

**Return type** bool

**confirm_partial_execution\(\{contract, breaches\}\)**

Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

**Parameters**

- **contract** \((Contract)\) – The contract that was breached
- **breaches** \((List[Breach])\) – A list of all the breaches committed.

**Remarks:**

- Will not be called if both partners committed breaches.

**Return type** bool

classmethod create\(*args, **kwargs\)**

Creates an object and returns a proxy to it.

classmethod do_nothing_manager\()\)

classmethod from_config\(config, section=None, ignore_children=True, try_parsing_children=True, scope=None\)

Creates an object of this class given the configuration info

**Parameters**

- **config** \((Union[str, dict])\) – Either a file name or a dictionary
- **section** \((Optional[str])\) – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** \((bool)\) – If true then children will be ignored and there will be a single return
- **try_parsing_children** \((bool)\) – If true the children will first be parsed as ConfigReader classes if they are not types\(\{e.g. int, str, float, Iterable[int|str|float]\}\)\(\(simple\)\)-
- **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
- **any children that are to be parsed.\(\(having\)\)\)-

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

**Remarks:**

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
• Requiring passing `scope=globals()` to this function is to get around the fact that in python `eval()` will be called with a `globals` dictionary based on the module in which the function is defined not called. This means that in general when `eval()` is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an `undefined_name` exception, the caller must pass her `globals()` as the scope.

```python
classmethod from_dict(java_object, *args, **kwargs)
```

Creates a Python object representing the corresponding Java object

```python
classmethod greedy_manager()
```

6.1. negmas.apps.scml Package
• **auto_load_java** *(bool)* – When true, a JVM will be automatically created (if one is not available)

• **python_shadow_object** *(Optional[Any])* – A python object to shadow the java object. The object will just call the corresponding

• **on this shadow object whenever it needs** *(method)* –

Remarks:

• sets a member called `java_object` that can be used to access the corresponding Java object created

• if `python_shadow_object` is given, it must be an object of a type that has an internal class `Java` which has a single member called `implements` which is a list of one string element representing the Java interface being implemented (it must be either `jnepmas.PyCallable` or an extension of it).

```python
notify (notifiable, notification)
onAgentBankrupt (agentId)
CashTransfer (amount, cause)

Return type None
onContractBreached (contract, breaches, resolution)

Return type None
onContractCancelled (contract, rejectors)
onContractExecuted (contract)

Return type None
onContractNullified (contract, bankruptPartner, compensation)
onContractSigned (contract)
onInventoryChange (product, quantity, cause)

Return type None
onNegRequestAccepted (req_id, mechanism)
onNegRequestRejected (req_id, rejectors)
onNegotiationFailure (partners, annotation, mechanism, state)
onNegotiationSuccess (contract, mechanism)
onNewCFP (cfp)
onNewReport (report)
onProductionFailure (failures)
onProductionSuccess (reports)

Return type None
onRemoveCFP (cfp)
on_agent_bankrupt (agent_id)

    Will be called whenever any agent goes bankrupt

Parameters agent_id *(str)* – The ID of the agent that went bankrupt

Remarks:

• Agents can go bankrupt in two cases:

  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.

Chapter 6. Application Modules
2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).

- All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their `on_new_cfp` and `respond_to_negotiation_request` callbacks by pulling the bulletin-board using the helper function `is_bankrupt` of their AWI.

Return type None

`on_cash_transfer` *(amount, cause)*

Received whenever money is transferred to the factory or from it.

Parameters

- `amount` *(float)* – Amount of money (negative for transfers out of the factory, positive for transfers to it).
- `cause` *(str)* – The cause of the change. Possibilities include:
  - `contract`: Contract execution
  - `insurance`: Received from insurance company
  - `bankruptcy`: Liquidated due to bankruptcy
  - `transfer`: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

`on_contract_breached` *(contract, breaches, resolution)*

Called after complete processing of a contract that involved a breach.

Parameters

- `contract` *(Contract)* – The contract
- `breaches` *(List[Breach]*) – All breaches committed (even if they were resolved)
- `resolution` *(Optional[Contract]*) – The resolution contract if re-negotiation was successful. None if not.

Return type None

`on_contract_cancelled` *(contract, rejectors)*

Called whenever at least a partner did not sign the contract

Return type None

`on_contract_cancelled_` *(contract, rejectors)*

Called whenever at least a partner did not sign the contract

Return type None

`on_contract_executed` *(contract)*

Called after successful contract execution for which the agent is one of the partners.

Return type None

`on_contract_nullified` *(contract, bankrupt_partner, compensation)*

Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

Return type None

`on_contract_signed` *(contract)*

Called whenever a contract is signed by all partners

Return type None

`on_contract_signed_` *(contract)*

Called whenever a contract is signed by all partners
Return type None

**on_event** *(event, sender)*

**on_inventory_change** *(product, quantity, cause)*

Received whenever something moves in or out of the factory’s storage

**Parameters**

- **product** *(int)* – Product index.
- **quantity** *(int)* – Negative value for products moving out and positive value for products moving in.
- **cause** *(str)* – The cause of the change. Possibilities include:
  - contract: Contract execution
  - insurance: Received from insurance company
  - bankruptcy: Liquidated due to bankruptcy
  - transport: Arrival of goods (when transportation delay in the system is > 0).

Return type None

**on_neg_request_accepted** *(req_id, mechanism)*

Called when a requested negotiation is accepted

**on_neg_request_accepted_** *(req_id, mechanism)*

Called when a requested negotiation is accepted

**on_neg_request_rejected** *(req_id, by)*

Called when a requested negotiation is rejected

**Parameters**

- **req_id** *(str)* – The request ID passed to _request_negotiation
- **by** *(Optional[List[str]])* – A list of agents that refused to participate or None if the failure was for another reason

**on_neg_request_rejected_** *(req_id, by)*

Called when a requested negotiation is rejected

**Parameters**

- **req_id** *(str)* – The request ID passed to _request_negotiation
- **by** *(Optional[List[str]])* – A list of agents that refused to participate or None if the failure was for another reason

**on_negotiation_failure** *(partners, annotation, mechanism, state)*

Called whenever a negotiation ends without agreement

Return type None

**on_negotiation_failure_** *(partners, annotation, mechanism, state)*

Called whenever a negotiation ends without agreement

Return type None

**on_negotiation_success** *(contract, mechanism)*

Called whenever a negotiation ends with agreement

Return type None

**on_negotiation_success_** *(contract, mechanism)*

Called whenever a negotiation ends with agreement

Return type None
on_new_cfp(cfp)
Called when a new CFP for a product for which the agent registered interest is published

on_new_report(report)
Called whenever a financial report is published.

Parameters report (FinancialReport) – The financial report giving details of the standing of an agent at some time (see FinancialReport)

Remarks:
• Agents must opt-in to receive these calls by calling receive_financial_reports on their AWI

on_production_failure(failures)
Called with a list of ProductionFailure records on production failure.

Return type None

on_production_success(reports)
Called with a list of ProductionReport records on production success

Return type None

on_remove_cfp(cfp)
Called when a new CFP for a product for which the agent registered interest is removed

classmethod read_config(config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters
• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:
requestNegotiation(cfp, negotiator=None, ufun=None)

Return type bool

request_negotiation(cfp, negotiator=None, ufun=None)
Requests a negotiation from the AWI while keeping track of available negotiation requests

Parameters
• cfp (CFP) –
• negotiator (Optional[Negotiator]) –
• ufun (Optional[UtilityFunction]) –

Return type bool

Returns Whether the negotiation request was successful indicating that the partner accepted the negotiation

respondToNegotiationRequest(cfp, partner)
respondToRenegotiationRequest(contract, breaches, agenda)
respond_to_negotiation_request(cfp, partner)
Called when a prospective partner requests a negotiation to start

Return type Optional[Negotiator]
respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received
Return type Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)
Called to respond to a renegotiation request
Parameters
  • agenda (RenegotiationRequest) –
  • contract (Contract) –
  • breaches (List[Breach]) –
Returns:
  Return type Optional[Negotiator]

setID (value)
setName (value)

setRenegotiationAgenda (contract, breaches)
set_renegotiation_agenda (contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails
Parameters
  • contract (Contract) – The contract being breached
  • breaches (List[Breach]) – All breaches on contract
Return type Optional[RenegotiationRequest]
Returns Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

signContract (contract)
sign_contract (contract)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.
Return type Optional[str]

step ()
Called by the simulator at every simulation step

stepPython ()

step_ ()
Called at every time-step. This function is called directly by the world.

JavaDummyMiddleMan

class negmas.apps.scml.JavaDummyMiddleMan (auto_load_java=False, name= None, simulator_type= <class 'negmas.apps.scml.simulators.FastFactorySimulator'>)
Bases: negmas.apps.scml.JavaFactoryManager

Attributes Summary
awi

Returns the Agent-World-Interface through which the agent does all of its actions in the world.

id

The unique ID of this entity

can_expect_agreement(cfp, margin)

Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable.

confirmContractExecution(contract)

called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached.

confirmLoan(loan, bankruptIfRejected)

called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached.

confirmPartialExecution(contract, breaches)

Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

creator(*args, **kwargs)

Creates an object and returns a proxy to it.

do_nothing_manager()


from_config(config[, section, ...])

Creates an object of this class given the configuration info.

from_dict(java_object, *args, **kwargs)

Creates a Python object representing the corresponding Java object.

greedy_manager()


init()

Called to initialize the agent after the world is initialized.

initPython()

The initialization function called by the world directly.

Continued on next page
Table 43 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init_java_bridge</code></td>
<td><code>(java_object, java_class_name)</code></td>
<td>initializes a connection to the java bridge creating a member called <code>java_object</code> that can be used to access the counterpart object in Java.</td>
</tr>
<tr>
<td><code>notify</code></td>
<td><code>(notifiable, notification)</code></td>
<td></td>
</tr>
<tr>
<td><code>onAgentBankrupt</code></td>
<td><code>(agentId)</code></td>
<td></td>
</tr>
<tr>
<td><code>onCashTransfer</code></td>
<td><code>(amount, cause)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractBreached</code></td>
<td><code>(contract, breaches, ...)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractCancelled</code></td>
<td><code>(contract, rejectors)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractExecuted</code></td>
<td><code>(contract)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractNullified</code></td>
<td><code>(contract, ...)</code></td>
<td></td>
</tr>
<tr>
<td><code>onContractSigned</code></td>
<td><code>(contract)</code></td>
<td></td>
</tr>
<tr>
<td><code>onInventoryChange</code></td>
<td><code>(product, quantity, cause)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegRequestAccepted</code></td>
<td><code>(req_id, mechanism)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegRequestRejected</code></td>
<td><code>(req_id, rejectors)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegotiationFailure</code></td>
<td><code>(partners, annotation, ...)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNegotiationSuccess</code></td>
<td><code>(contract, mechanism)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNewCFP</code></td>
<td><code>(cfp)</code></td>
<td></td>
</tr>
<tr>
<td><code>onNewReport</code></td>
<td><code>(report)</code></td>
<td></td>
</tr>
<tr>
<td><code>onProductionFailure</code></td>
<td><code>(failures)</code></td>
<td></td>
</tr>
<tr>
<td><code>onProductionSuccess</code></td>
<td><code>(reports)</code></td>
<td></td>
</tr>
<tr>
<td><code>onRemoveCFP</code></td>
<td><code>(cfp)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_agent_bankrupt</code></td>
<td><code>(agent_id)</code></td>
<td>Will be called whenever any agent goes bankrupt.</td>
</tr>
<tr>
<td><code>on_cash_transfer</code></td>
<td><code>(amount, cause)</code></td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td><code>on_contract_breached</code></td>
<td><code>(contract, breaches, ...)</code></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><code>on_contract_cancelled</code></td>
<td><code>(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
<tr>
<td><code>on_contract_cancelled_2</code></td>
<td><code>(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
<tr>
<td><code>on_contract_executed</code></td>
<td><code>(contract)</code></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><code>on_contract_nullified</code></td>
<td><code>(contract, ...)</code></td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.</td>
</tr>
<tr>
<td><code>on_contract_signed</code></td>
<td><code>(contract)</code></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><code>on_contract_signed_2</code></td>
<td><code>(contract)</code></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><code>on_event</code></td>
<td><code>(event, sender)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_inventory_change</code></td>
<td><code>(product, quantity, cause)</code></td>
<td>Received whenever something moves in or out of the factory’s storage.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 43 – continued from previous page

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_neg_request_accepted</code></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><code>on_neg_request_rejected</code></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><code>on_negotiation_failure</code></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><code>on_negotiation_success</code></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><code>on_new_cfp</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is published</td>
</tr>
<tr>
<td><code>on_new_report</code></td>
<td>Called whenever a financial report is published</td>
</tr>
<tr>
<td><code>on_production_failure</code></td>
<td>Called with a list of ProductionFailure records on production failure.</td>
</tr>
<tr>
<td><code>on_production_success</code></td>
<td>Called with a list of ProductionReport records on production success</td>
</tr>
<tr>
<td><code>on_remove_cfp</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td><code>read_config</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td><code>requestNegotiation</code></td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests</td>
</tr>
<tr>
<td><code>respondToNegotiationRequest</code></td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request</code></td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request</code></td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td><code>setID</code></td>
<td></td>
</tr>
<tr>
<td><code>setName</code></td>
<td></td>
</tr>
<tr>
<td><code>setRenegotiationAgenda</code></td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>signContract</code></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step</code></td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td><code>stepPython</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>
Attributes Documentation

awi
Returns the Agent-World-Interface through which the agent does all of its actions in the world.
A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations
The negotiations currently requested by the agent.

Return type List[NegotiationRequestInfo]

Returns A list of negotiation request information objects (NegotiationRequestInfo)

running_negotiations
The negotiations currently requested by the agent.

Return type List[RunningNegotiationInfo]

Returns A list of negotiation information objects (RunningNegotiationInfo)

short_type_name
Returns a short name of the type of this entity

type_name
Overrides type name to give the internal java type name

unsigned_contracts
All contracts that are not yet signed.

Return type List[Contract]

uuid
The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

• margin (int) –
• cfp (CFP) –

Returns:

confirmContractExecution (contract)
confirmLoan (loan, bankruptIfRejected)
confirmPartialExecution (contract, breaches)
confirm_contract_execution (contract)
Called before executing any agreement

Return type bool

confirm_loan (loan, bankrupt_if_rejected)
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached
Return type `bool`

`confirm_partial_execution(contract, breaches)`
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters

- `contract (Contract)` – The contract that was breached
- `breaches (List[Breach])` – A list of all the breaches committed.

Remarks:

- Will not be called if both partners committed breaches.

Return type `bool`

`classmethod create(*args, **kwargs)`
Creates an object and returns a proxy to it.

`classmethod do_nothing_manager()`

`classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)`
Creates an object of this class given the configuration info

Parameters

- `config (Union[str, dict])` – Either a file name or a dictionary
- `section (Optional[str])` – A section in the file or a key in the dictionary to use for loading params
- `ignore_children (bool)` – If true then children will be ignored and there will be a single return
- `try_parsing_children (bool)` – If true the children will first be parsed as `ConfigReader` classes if they are not `types (e.g. int, str, float, Iterable[int,str,float] (simple)–
- `scope` – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are
- `any children that are to be parsed. (having)–

Returns

An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

`classmethod from_dict(java_object, *args, **kwargs)`
Creates a Python object representing the corresponding Java object

`getCompiledProfiles()`

`getConsuming()`

`getContracts()`

6.1. negmas.apps.scml Package
getID()
getLineProfiles()
getName()
getNegotiationRequests()
getProcesses()
getProducing()
getProducts()
getRequestedNegotiations()
getRunningNegotiations()
classmethod greedy_manager()
inited
    Called to initialize the agent after the world is initialized. the AWI is accessible at this point.
initedPython()
inited()
The initialization function called by the world directly.
    It does the following actions by default:
    1. copies some of the static world settings to the agent to make them available without calling the AWI.
    2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
    3. registers interest in all products that the agent can produce or consume in its factory.
    4. finally it calls any custom initialization logic implemented in ‘init’()

See also:
inited, step
init_java_bridge(java_object, java_class_name, auto_load_java=False,
python_shadow_object=None)
initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java

Parameters

• java_object – A java object that already exists of the correct type. If given no new objects will be created
• java_class_name (str) – The type of the Java object to be created
• auto_load_java (bool) – When true, a JVM will be automatically created (if one is not available)
• python_shadow_object (Optional[Any]) – A python object to shadow the java object. The object will just call the corresponding

• on this shadow object whenever it needs. (method) –

Remarks:

• sets a member called java_object that can be used to access the corresponding Java object crated
• if python_shadow_object is given, it must be an object of a type that has an internal class called Java which has a single member called ‘implements’ which is a list of one string element representing the Java interface being implemented (it must be either jnegmas.PyCallable or an extension of it).
notify (notifiable, notification)
onAgentBankrupt (agentId)
onCashTransfer (amount, cause)
    Return type None
onContractBreached (contract, breaches, resolution)
    Return type None
onContractCancelled (contract, rejectors)
onContractExecuted (contract)
    Return type None
onContractNullified (contract, bankruptPartner, compensation)
onContractSigned (contract)
onInventoryChange (product, quantity, cause)
    Return type None
onNegRequestAccepted (req_id, mechanism)
onNegRequestRejected (req_id, rejectors)
onNegotiationFailure (partners, annotation, mechanism, state)
onNegotiationSuccess (contract, mechanism)
onNewCFP (cfp)
onNewReport (report)
onProductionFailure (failures)
onProductionSuccess (reports)
    Return type None
onRemoveCFP (cfp)
on_agent_bankrupt (agent_id)
    Will be called whenever any agent goes bankrupt

Parameters agent_id (str) – The ID of the agent that went bankrupt

Remarks:
    • Agents can go bankrupt in two cases:
        1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
        2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).
    • All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)
    Received whenever money is transferred to the factory or from it.

Parameters
    • amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
• **cause** *(str)* – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

**Return type** None

**on_contract_breached** *(contract, breaches, resolution)*
Called after complete processing of a contract that involved a breach.

**Parameters**
• **contract** *(Contract)* – The contract
• **breaches** *(List[Breach])* – All breaches committed (even if they were resolved)
• **resolution** *(Optional[Contract])* – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

**on_contract_cancelled** *(contract, rejectors)*
Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_cancelled_** *(contract, rejectors)*
Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_executed** *(contract)*
Called after successful contract execution for which the agent is one of the partners.

**Return type** None

**on_contract_nullified** *(contract, bankrupt_partner, compensation)*
Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

**Return type** None

**on_contract_signed** *(contract)*
Called whenever a contract is signed by all partners

**Return type** None

**on_contract_signed_** *(contract)*
Called whenever a contract is signed by all partners

**Return type** None

**on_event** *(event, sender)*

**on_inventory_change** *(product, quantity, cause)*
Received whenever something moves in or out of the factory’s storage

**Parameters**
• **product** *(int)* – Product index.
• **quantity** *(int)* – Negative value for products moving out and positive value for products moving in
• **cause** *(str)* – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
– bankruptcy: Liquidated due to bankruptcy
– transport: Arrival of goods (when transportation delay in the system is $> 0$).

**Return type** None

**on_neg_request_accepted** (*req_id, mechanism*)
Called when a requested negotiation is accepted

**on_neg_request_accepted_** (*req_id, mechanism*)
Called when a requested negotiation is accepted

**on_neg_request_rejected** (*req_id, by*)
Called when a requested negotiation is rejected

**Parameters**
- *req_id* (*str*) – The request ID passed to _request_negotiation
- *by* (*Optional[List[str]]*) – A list of agents that refused to participate or None if the failure was for another reason

**on_neg_request_rejected_** (*req_id, by*)
Called when a requested negotiation is rejected

**Parameters**
- *req_id* (*str*) – The request ID passed to _request_negotiation
- *by* (*Optional[List[str]]*) – A list of agents that refused to participate or None if the failure was for another reason

**on_negotiation_failure** (*partners, annotation, mechanism, state*)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_failure_** (*partners, annotation, mechanism, state*)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_success** (*contract, mechanism*)
Called whenever a negotiation ends with agreement

**Return type** None

**on_negotiation_success_** (*contract, mechanism*)
Called whenever a negotiation ends with agreement

**Return type** None

**on_new_cfp** (*cfp*)
Called when a new CFP for a product for which the agent registered interest is published

**on_new_report** (*report*)
Called whenever a financial report is published.

**Parameters** *report* (*FinancialReport*) – The financial report giving details of the standing of an agent at some time (see *FinancialReport*)

**Remarks:**
- Agents must opt-in to receive these calls by calling *receive_financial_reports* on their AWI

**on_production_failure** (*failures*)
Called with a list of *ProductionFailure* records on production failure.

**Return type** None

---

6.1. negmas.apps.scml Package
on_production_success (reports)
Called with a list of ProductionReport records on production success
Return type None

on_remove_cfp (cfp)
Called when a new CFP for a product for which the agent registered interest is removed

classmethod read_config (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters
• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:

requestNegotiation (cfp, negotiator=None, ufun=None)

Return type bool

request_negotiation (cfp, negotiator=None, ufun=None)
Requests a negotiation from the AWI while keeping track of available negotiation requests

Parameters
• cfp (CFP) –
• negotiator (Optional[Negotiator]) –
• ufun (Optional[UtilityFunction]) –

Return type bool

Returns Whether the negotiation request was successful indicating that the partner accepted the negotiation

respondToNegotiationRequest (cfp, partner)

respondToRenegotiationRequest (contract, breaches, agenda)

respond_to_negotiation_request (cfp, partner)
Called when a prospective partner requests a negotiation to start

Return type Optional[Negotiator]

respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received

Return type Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)
Called to respond to a renegotiation request

Parameters
• agenda (RenegotiationRequest) –
• contract (Contract) –
• breaches (List[Breach]) –

Returns:

Return type Optional[Negotiator]
**setID** *(value)*

**setName** *(value)*

**setRenegotiationAgenda** *(contract, breaches)*

**set_renegotiation_agenda** *(contract, breaches)*

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- **contract** *(Contract)* – The contract being breached
- **breaches** *(List[Breach])* – All breaches on **contract**

**Return type** *Optional[RenegotiationRequest]*

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

**signContract** *(contract)*

**sign_contract** *(contract)*

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** *Optional[str]*

**step** *

Called by the simulator at every simulation step

**stepPython** *

**step_** *

Called at every time-step. This function is called directly by the world.

---

**DefaultBank**

**class** *negmas.apps.scml.DefaultBank* *(minimum_balance, interest_rate, interest_max, balance_at_max_interest, installment_interest, time_increment, a2f, disabled=False, name=None)*

**Bases** *negmas.apps.scml.Bank*

Represents a bank in the world

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>awi</strong></td>
<td>Gets the Agent-world interface.</td>
</tr>
<tr>
<td><strong>id</strong></td>
<td>The unique ID of this entity.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>A convenient name of the entity (intended primar-</td>
</tr>
<tr>
<td></td>
<td>ily for printing/logging/debugging).</td>
</tr>
<tr>
<td><strong>requested_negotiations</strong></td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td><strong>running_negotiations</strong></td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td><strong>short_type_name</strong></td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td><strong>type_name</strong></td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td><strong>unsigned_contracts</strong></td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td><strong>uuid</strong></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**
buy_loan(agent, amount, n_installments, . . . )
   Gives a loan of amount to agent at the interest calculated using evaluate_loan.

creator(*args, **kwargs)
   Creates an object and returns a proxy to it.

credit_rating(agent_id)
   rtype float
   evaluate_loan(agent, amount, start_at, . . . )
   Evaluates the interest that will be imposed on the agent to buy_loan that amount.

from_config(config[, section, . . . ])
   Creates an object of this class given the configuration info.

init()
   Called to initialize the agent after the world is initialized.

init_()
   Called to initialize the agent after the world is initialized.

notify(notifiable, notification)

on_contract_breached(contract, breaches, . . . )
   Called after complete processing of a contract that involved a breach.

on_contract_cancelled(contract, rejectors)
   Called whenever at least a partner did not sign the contract.

on_contract_cancelled_(contract, rejectors)
   Called whenever at least a partner did not sign the contract.

on_contract_executed(contract)
   Called after successful contract execution for which the agent is one of the partners.

on_contract_signed(contract)
   Called whenever a contract is signed by all partners.

on_contract_signed_(contract)
   Called whenever a contract is signed by all partners.

on_event(event, sender)

on_neg_request_accepted(req_id, mechanism)
   Called when a requested negotiation is accepted.

on_neg_request_accepted_(req_id, mechanism)
   Called when a requested negotiation is accepted.

on_neg_request_rejected(req_id, by)
   Called when a requested negotiation is rejected.

on_neg_request_rejected_(req_id, by)
   Called when a requested negotiation is rejected.

on_negotiation_failure(partners, annotation, . . . )
   Called whenever a negotiation ends without agreement.

on_negotiation_failure_(partners, . . . )
   Called whenever a negotiation ends without agreement.

on_negotiation_success(contract, mechanism)
   Called whenever a negotiation ends with agreement.

on_negotiation_success_(contract, mechanism)
   Called whenever a negotiation ends with agreement.

read_config(config[, section])
   Reads the configuration from a file or a dict and prepares it for parsing.

respond_to_negotiation_request(initiator, . . . )
   rtype Optional[Negotiator]

respond_to_negotiation_request_(initiator, . . . )
   Called when a negotiation request is received.

respond_to_renegotiation_request(contract, . . . )
   Called to respond to a renegotiation request.

set_renegotiation_agenda(contract, breaches)
   Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails.
Table 45 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sign_contract(contract)</code></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Takes payments from agents</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- `awi`  
  Gets the Agent-world interface.  
  
  **Return type** `AgentWorldInterface`

- `id`  
  The unique ID of this entity

- `name`  
  A convenient name of the entity (intended primarily for printing/logging/debugging).

- `requested_negotiations`  
  The negotiations currently requested by the agent.  
  
  **Return type** `List[NegotiationRequestInfo]`  
  **Returns** A list of negotiation request information objects `NegotiationRequestInfo`

- `running_negotiations`  
  The negotiations currently requested by the agent.  
  
  **Return type** `List[RunningNegotiationInfo]`  
  **Returns** A list of negotiation information objects `RunningNegotiationInfo`

- `short_type_name`  
  Returns a short name of the type of this entity

- `type_name`  
  Returns the name of the type of this entity

- `unsigned_contracts`  
  All contracts that are not yet signed.  
  
  **Return type** `List[Contract]`

- `uuid`  
  The unique ID of this entity

**Methods Documentation**

- `buy_loan(agent, amount, n_installments, beneficiary, contract, force=False)`  
  Gives a loan of amount to agent at the interest calculated using `evaluate_loan`  
  
  **Return type** `Optional[Loan]`

- `classmethod create(*args, **kwargs)`  
  Creates an object and returns a proxy to it.

- `credit_rating(agent_id)`  
  
  **Return type** `float`

- `evaluate_loan(agent, amount, start_at, n_installments)`  
  Evaluates the interest that will be imposed on the agent to `buy_loan` that amount  
  
  **Return type** `Optional[Loan]`
classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)

Creates an object of this class given the configuration info

Parameters

- `config`(Union[str, dict]) – Either a file name or a dictionary
- `section`(Optional[str]) – A section in the file or a key in the dictionary to use for loading params
- `ignore_children`(bool) – If true then children will be ignored and there will be a single return
- `try_parsing_children`(bool) – If true the children will first be parsed as ConfigReader classes if they are not
- `types`(e.g. int, str, float, Iterable[int|str|float])
- `scope` – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
- `any children that are to be parsed.`(having)

Returns An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

`init()`
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

`init_()`
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

`notify(notifiable, notification)`

`on_contract_breached(contract, breaches, resolution)`
Called after complete processing of a contract that involved a breach.

Parameters

- `contract`(Contract) – The contract
- `breaches`(List[Breach]) – All breaches committed (even if they were resolved)
- `resolution`(Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

`on_contract_cancelled(contract, rejectors)`
Called whenever at least a partner did not sign the contract

Return type None

`on_contract_cancelled_(contract, rejectors)`
Called whenever at least a partner did not sign the contract

Return type None
on_contract_executed \( (contract) \)
Called after successful contract execution for which the agent is one of the partners.

Return type None

on_contract_signed \( (contract) \)
Called whenever a contract is signed by all partners

Return type None

on_contract_signed\_ \( (contract) \)
Called whenever a contract is signed by all partners

Return type None

on_event \( (event, sender) \)

on_neg_request_accepted \( (req_id, mechanism) \)
Called when a requested negotiation is accepted

on_neg_request_accepted\_ \( (req_id, mechanism) \)
Called when a requested negotiation is accepted

on_neg_request_rejected \( (req_id, by) \)
Called when a requested negotiation is rejected

Parameters
  • \textbf{req_id} (str) – The request ID passed to \_request_negotiation
  • \textbf{by} (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_neg_request_rejected\_ \( (req_id, by) \)
Called when a requested negotiation is rejected

Parameters
  • \textbf{req_id} (str) – The request ID passed to \_request_negotiation
  • \textbf{by} (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_negotiation_failure \( (partners, annotation, mechanism, state) \)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_failure\_ \( (partners, annotation, mechanism, state) \)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_success \( (contract, mechanism) \)
Called whenever a negotiation ends with agreement

Return type None

on_negotiation_success\_ \( (contract, mechanism) \)
Called whenever a negotiation ends with agreement

Return type None

\textbf{classmethod read_config} \( (config, section=None) \)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters
  • \textbf{config} (Union[str, dict]) – Either a file name or a dictionary
  • \textbf{section} (Optional[str]) – A section in the file or a key in the dictionary to use for loading params
Return type  Dict[str, Any]

Returns  A dict ready to be parsed by from_config

Remarks:

respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)

Return type  Optional[Negotiator]

respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism, role, req_id)

Called when a negotiation request is received

Return type  Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)

Called to respond to a renegotiation request

Parameters

• agenda (RenegotiationRequest) –
• contract (Contract) –
• breaches (List[Breach]) –

Returns:

Return type  Optional[Negotiator]

set_renegotiation_agenda (contract, breaches)

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters

• contract (Contract) – The contract being breached
• breaches (List[Breach]) – All breaches on contract

Return type  Optional[RenegotiationRequest]

Returns  Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

sign_contract (contract)

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type  Optional[str]

step ()

Takes payments from agents

step_ ()

Called at every time-step. This function is called directly by the world.

Bank
class negmas.apps.scml.Bank (*args, **kwargs)

Bases: negmas.situated.Agent, abc.ABC

Base class for all banks

Attributes Summary
awi
id
name
requested_negotiations
running_negotiations
short_type_name
type_name
unsigned_contracts
uuid

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**create(**args, <strong>kwargs)</strong></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><strong>from_config</strong>(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td><strong>init()</strong></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><strong>init_()</strong></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><strong>notify</strong>(notifiable, notification)</td>
<td>Called to notify the agent of an event.</td>
</tr>
<tr>
<td><strong>on_contract_breached</strong></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><strong>on_contract_cancelled</strong></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><strong>on_contract_cancelled_</strong></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><strong>on_contract_executed</strong></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><strong>on_contract_signed</strong></td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td><strong>on_contract_signed_</strong></td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td><strong>on_event</strong>(event, sender)</td>
<td>Called when a contract event occurs</td>
</tr>
<tr>
<td><strong>on_neg_request_accepted</strong></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><strong>on_neg_request_accepted_</strong></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><strong>on_neg_request_rejected</strong></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><strong>on_neg_request_rejected_</strong></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><strong>on_negotiation_failure</strong></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><strong>on_negotiation_failure_</strong></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><strong>on_negotiation_success</strong></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><strong>on_negotiation_success_</strong></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><strong>read_config</strong>(config[, section])</td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td><strong>respond_to_negotiation_request</strong></td>
<td>Called when a negotiation request is processed.</td>
</tr>
<tr>
<td></td>
<td><em>rtype</em> Optional[Negotiator]</td>
</tr>
</tbody>
</table>
Table 47 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>respond_to_negotiation_request_(initiator,</td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>respond_to_renegotiation_request</td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td>set_renegotiation_agenda(contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order</td>
</tr>
<tr>
<td></td>
<td>to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td>sign_contract(contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td>step()</td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td>step_()</td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

Attributes Documentation

awi
- Gets the Agent-world interface.

  **Return type** AgentWorldInterface

id
- The unique ID of this entity

name
- A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations
- The negotiations currently requested by the agent.

  **Return type** List[NegotiationRequestInfo]

  **Returns** A list of negotiation request information objects (NegotiationRequestInfo)

running_negotiations
- The negotiations currently requested by the agent.

  **Return type** List[RunningNegotiationInfo]

  **Returns** A list of negotiation information objects (RunningNegotiationInfo)

short_type_name
- Returns a short name of the type of this entity

type_name
- Returns the name of the type of this entity

unsigned_contracts
- All contracts that are not yet signed.

  **Return type** List[Contract]

uuid
- The unique ID of this entity

Methods Documentation

classmethod create(*args, **kwargs)
- Creates an object and returns a proxy to it.

classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)
- Creates an object of this class given the configuration info

  **Parameters**

  - config (Union[str, dict]) – Either a file name or a dictionary
• **section** *(Optional)* – A section in the file or a key in the dictionary to use for loading params

• **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return

• **try_parsing_children** *(bool)* – If true the children will first be parsed as `ConfigReader` classes if they are not

• **types** *(e.g., int, str, float, Iterable[int|str|float]) (simple)* –

• **scope** – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are

• **any children that are to be parsed.** *(having)* –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing `scope=globals()` to this function is to get around the fact that in python `eval()` will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when `eval()` is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her `globals()` as the scope.

`init()`
Called to initialize the agent **after** the world is initialized. The AWI is accessible at this point.

`init_()`
Called to initialize the agent **after** the world is initialized. The AWI is accessible at this point.

`notify` *(notifiable, notification)*

`on_contract_breached` *(contract, breaches, resolution)*
Called after complete processing of a contract that involved a breach.

**Parameters**

• **contract** *(Contract)* – The contract

• **breaches** *(List[Breach])* – All breaches committed (even if they were resolved)

• **resolution** *(Optional)* *(Contract)* – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

`on_contract_cancelled` *(contract, rejectors)*
Called whenever at least a partner did not sign the contract

**Return type** None

`on_contract_cancelled_` *(contract, rejectors)*
Called whenever at least a partner did not sign the contract

**Return type** None

`on_contract_executed` *(contract)*
Called after successful contract execution for which the agent is one of the partners.

**Return type** None

`on_contract_signed` *(contract)*
Called whenever a contract is signed by all partners
Return type None

on_contract_signed_(contract)
Called whenever a contract is signed by all partners

Return type None

on_event (event, sender)

on_neg_request_accepted (req_id, mechanism)
Called when a requested negotiation is accepted

on_neg_request_accepted_ (req_id, mechanism)
Called when a requested negotiation is accepted

on_neg_request_rejected (req_id, by)
Called when a requested negotiation is rejected

Parameters

- req_id (str) – The request ID passed to _request_negotiation
- by (Optional[List[str]]) – A list of agents that refused to participate or None if
  the failure was for another reason

on_neg_request_rejected_ (req_id, by)
Called when a requested negotiation is rejected

Parameters

- req_id (str) – The request ID passed to _request_negotiation
- by (Optional[List[str]]) – A list of agents that refused to participate or None if
  the failure was for another reason

on_negotiation_failure (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_failure_ (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_success (contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

on_negotiation_success_ (contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

classmethod read_config (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

- config (Union[str, dict]) – Either a file name or a dictionary
- section (Optional[str]) – A section in the file or a key in the dictionary to use
  for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:

respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)
**Return type** Optional[\texttt{Negotiator}]

\texttt{respond_to_negotiation_request}\_ (\texttt{initiator}, \texttt{partners}, \texttt{issues}, \texttt{annotation}, \texttt{mechanism}, \texttt{role}, \texttt{req\_id})

Called when a negotiation request is received

**Return type** Optional[\texttt{Negotiator}]

\texttt{respond_to_renegotiation_request} (\texttt{contract}, \texttt{breaches}, \texttt{agenda})

Called to respond to a renegotiation request

**Parameters**

- \texttt{agenda} (\texttt{RenegotiationRequest}) –
- \texttt{contract} (\texttt{Contract}) –
- \texttt{breaches} (\texttt{List[Breach]}) –

**Returns:**

**Return type** Optional[\texttt{Negotiator}]

\texttt{set_renegotiation_agenda} (\texttt{contract}, \texttt{breaches})

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- \texttt{contract} (\texttt{Contract}) – The contract being breached
- \texttt{breaches} (\texttt{List[Breach]}) – All breaches on \texttt{contract}

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

\texttt{sign_contract} (\texttt{contract})

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** Optional[\texttt{str}]

\texttt{step} ()

Called by the simulator at every simulation step

\texttt{step\_} ()

Called at every time-step. This function is called directly by the world.

### DefaultInsuranceCompany

**class** \texttt{negmas.apps.scml.DefaultInsuranceCompany} (\texttt{premium}, \texttt{premium\_breach\_increment}, \texttt{premium\_time\_increment}, \texttt{a2f}, \texttt{disabled}=\texttt{False}, \texttt{name}=\texttt{None})

\texttt{Bases: negmas.apps.scml.InsuranceCompany}

Represents an insurance company in the world

#### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{awi}</td>
<td>Gets the Agent-world interface.</td>
</tr>
<tr>
<td>\texttt{id}</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>\texttt{name}</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 48 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

- **buy_insurance**(contract, insured, against) Buys insurance for the contract at the premium calculated by the insurance company.
- **create**(\*args, \*\*kwargs) Creates an object and returns a proxy to it.
- **evaluate_insurance**(contract, insured, against) Can be called to evaluate the premium for insuring the given contract against breaches committed by others.
- **from_config**(config[, section, . . .]) Creates an object of this class given the configuration info.
- **init()** Called to initialize the agent after the world is initialized.
- **init()** Called to initialize the agent after the world is initialized.
- **is_insured**(contract, perpetrator) type contract  Contract
- **notify**(notifiable, notification) Called after complete processing of a contract that involved a breach.
- **on_contract_breached**(contract, breaches, ...) Called whenever at least a partner did not sign the contract.
- **on_contract_cancelled**(contract, rejectors) Called whenever at least a partner did not sign the contract.
- **on_contract_cancelled**(contract, rejectors) Called whenever at least a partner did not sign the contract.
- **on_contract_executed**(contract) Called after successful contract execution for which the agent is one of the partners.
- **on_contract_signed**(contract) Called whenever a contract is signed by all partners.
- **on_contract_signed**(contract) Called whenever a contract is signed by all partners.
- **on_event**(event, sender)
- **on_neg_request_accepted**(req_id, mechanism) Called when a requested negotiation is accepted.
- **on_neg_request_accepted**(req_id, mechanism) Called when a requested negotiation is accepted.
- **on_neg_request_rejected**(req_id, by) Called when a requested negotiation is rejected.
- **on_neg_request_rejected**(req_id, by) Called when a requested negotiation is rejected.
- **on_negotiation_failure**(partners, annotation, . . .) Called whenever a negotiation ends without agreement.
- **on_negotiation_failure**(partners, . . .) Called whenever a negotiation ends without agreement.
- **on_negotiation_success**(contract, mechanism) Called whenever a negotiation ends with agreement.
- **on_negotiation_success**(contract, mechanism) Called whenever a negotiation ends with agreement.
- **read_config**(config[, section]) Reads the configuration from a file or a dict and prepares it for parsing.

Continued on next page
Table 49 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>respond_to_negotiation_request</code> (initiator, ...)</td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request</code> (contract, ...)</td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td><code>set_renegotiation_agenda</code> (contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>sign_contract</code> (contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>does nothing</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

- **awi**
  Gets the Agent-world interface.
  
  **Return type** `AgentWorldInterface`

- **id**
  The unique ID of this entity

- **name**
  A convenient name of the entity (intended primarily for printing/logging/debugging).

- **requested_negotiations**
  The negotiations currently requested by the agent.
  
  **Return type** `List[NegotiationRequestInfo]`
  
  **Returns** A list of negotiation request information objects (`NegotiationRequestInfo`)

- **running_negotiations**
  The negotiations currently requested by the agent.
  
  **Return type** `List[RunningNegotiationInfo]`
  
  **Returns** A list of negotiation information objects (`RunningNegotiationInfo`)

- **short_type_name**
  Returns a short name of the type of this entity

- **type_name**
  Returns the name of the type of this entity

- **unsigned_contracts**
  All contracts that are not yet signed.
  
  **Return type** `List[Contract]`

- **uuid**
  The unique ID of this entity

**Methods Documentation**

- **buy_insurance** (`contract`, insured, against)
  Buys insurance for the contract at the premium calculated by the insurance company.
  
  **Remarks:** The agent can call `evaluate_insurance` to find the premium that will be used.
See also:
evaluate_premium

Return type Optional[InsurancePolicy]

classmethod create(*args, **kwargs)
Creates an object and returns a proxy to it.

evaluate_insurance(contract, insured, against, t=None)
Can be called to evaluate the premium for insuring the given contract against breaches committed by others

Parameters
- against (SCMLAgent) – The SCMLAgent to insure against
- contract (Contract) – hypothetical contract
- insured (SCMLAgent) – The SCMLAgent to buy the insurance
- t (Optional[int]) – time at which the policy is to be bought. If None, it means current step

Remarks:
- The premium returned is relative to the contract price. To actually calculate the cost of buying this insurance, you need to multiply this by the contract value (quantity * unit_price).

Return type Optional[float]

classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)
Creates an object of this class given the configuration info

Parameters
- config (Union[str, dict]) – Either a file name or a dictionary
- section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params
- ignore_children (bool) – If true then children will be ignored and there will be a single return
- try_parsing_children (bool) – If true the children will first be parsed as ConfigReader classes if they are not
- types (e.g. int, str, float, Iterable[int|str|float]) (simple)–
  - scope – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
  - any children that are to be parsed. (having)–

Returns An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:
- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file).
To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**init()**
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**init**()
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**is_insured**(contract, perpetrator)

Parameters

- **contract** ([Contract])
- **perpetrator** ([SCMLAgent])

Returns:

- **Return type** bool

**notify**(notifiable, notification)

**on_contract_breached**(contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters

- **contract** ([Contract]) – The contract
- **breaches** ([List[Breach]]) – All breaches committed (even if they were resolved)
- **resolution** ([Optional[Contract]]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

**on_contract_cancelled**(contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

**on_contract_cancelled_**(contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

**on_contract_executed**(contract)
Called after successful contract execution for which the agent is one of the partners.

Return type None

**on_contract_signed**(contract)
Called whenever a contract is signed by all partners

Return type None

**on_contract_signed_**(contract)
Called whenever a contract is signed by all partners

Return type None

**on_event**(event, sender)

**on_neg_request_accepted**(req_id, mechanism)
Called when a requested negotiation is accepted

**on_neg_request_accepted_**(req_id, mechanism)
Called when a requested negotiation is accepted

**on_neg_request_rejected**(req_id, by)
Called when a requested negotiation is rejected
Parameters

• `req_id (str)` – The request ID passed to `_request_negotiation`

• `by (Optional[List[str]])` – A list of agents that refused to participate or None if the failure was for another reason

`on_neg_request_rejected_ (req_id, by)`
Called when a requested negotiation is rejected

Parameters

• `req_id (str)` – The request ID passed to `_request_negotiation`

• `by (Optional[List[str]])` – A list of agents that refused to participate or None if the failure was for another reason

`on_negotiation_failure (partners, annotation, mechanism, state)`
Called whenever a negotiation ends without agreement

Return type None

`on_negotiation_failure_ (partners, annotation, mechanism, state)`
Called whenever a negotiation ends without agreement

Return type None

`on_negotiation_success (contract, mechanism)`
Called whenever a negotiation ends with agreement

Return type None

`on_negotiation_success_ (contract, mechanism)`
Called whenever a negotiation ends with agreement

Return type None

`classmethod read_config (config, section=None)`
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• `config (Union[str, dict])` – Either a file name or a dictionary

• `section (Optional[str])` – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:

`respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)`

Return type Optional[Negotiator]

`respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism, role, req_id)`
Called when a negotiation request is received

Return type Optional[Negotiator]

`respond_to_renegotiation_request (contract, breaches, agenda)`
Called to respond to a renegotiation request

Parameters

• `agenda (RenegotiationRequest)` –

• `contract (Contract)` –
• breaches (List[Breach]) –

Returns:

Return type Optional[Negotiator]

set_renegotiation_agenda (contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters

• contract (Contract) – The contract being breached
• breaches (List[Breach]) – All breaches on contract

Return type Optional[RenegotiationRequest]

Returns Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

sign_contract (contract)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type Optional[str]

step ()
does nothing

step ()
Called at every time-step. This function is called directly by the world.

InsuranceCompany

class negmas.apps.scml.InsuranceCompany (*args, **kwargs)
Bases: negmas.situated.Agent, abc.ABC

Base class for all insurance companies

Attributes Summary

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Gets the Agent-world interface.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create(*args, **kwargs)</td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td>from_config(config[, section, ...])</td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td>init()</td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><code>notify(notifiable, notification)</code></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><code>on_contract_breach(contract, breaches, ...)</code></td>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
<tr>
<td><code>on_contract_cancelled(contract, rejectors)</code></td>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
<tr>
<td><code>on_contract_executed(contract)</code></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><code>on_contract_signed(contract)</code></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><code>on_event(event, sender)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_neg_request_accepted(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><code>on_neg_request_accepted_(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><code>on_neg_request_rejected(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected.</td>
</tr>
<tr>
<td><code>on_neg_request_rejected_(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected.</td>
</tr>
<tr>
<td><code>on_negotiation_failure(partners, annotation, ...)</code></td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td><code>on_negotiation_success(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td><code>on_negotiation_success_(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td><code>read_config(config[, section])</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request(initiator, ...)</code></td>
<td>Called when a negotiation request is received.</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request_(initiator, ...)</code></td>
<td>Called when a negotiation request is received.</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request(contract, ...)</code></td>
<td>Called to respond to a renegotiation request.</td>
</tr>
<tr>
<td><code>set_renegotiation_agenda(contract, breaches)</code></td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails.</td>
</tr>
<tr>
<td><code>sign_contract(contract)</code></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step(contract)</code></td>
<td>Called by the simulator at every simulation step.</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

- **awi**
  - Gets the Agent-world interface.
  
  **Return type** `AgentWorldInterface`

- **id**
  - The unique ID of this entity
name
A convenient name of the entity (intended primarily for printing/logging/debugging).

**requested_negotiations**
The negotiations currently requested by the agent.

**Return type** List[NegotiationRequestInfo]

**Returns** A list of negotiation request information objects (NegotiationRequestInfo)

**running_negotiations**
The negotiations currently requested by the agent.

**Return type** List[RunningNegotiationInfo]

**Returns** A list of negotiation information objects (RunningNegotiationInfo)

**short_type_name**
Returns a short name of the type of this entity

**type_name**
Returns the name of the type of this entity

**unsigned_contracts**
All contracts that are not yet signed.

**Return type** List[Contract]

**uuid**
The unique ID of this entity

**Methods Documentation**

**classmethod create(**args, **kwargs)**
Creates an object and returns a proxy to it.

**classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)**
Creates an object of this class given the configuration info

**Parameters**

- **config**(Union[str, dict]) – Either a file name or a dictionary
- **section**(Optional[str]) – A section in the file or a key in the dictionary to use for loading params
- **ignore_children**(bool) – If true then children will be ignored and there will be a single return
- **try_parsing_children**(bool) – If true the children will first be parsed as ConfigReader classes if they are not
- **types** *(e.g. int, str, float, Iterable[int|str|float]) (simple)* –
- **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
- **any children that are to be parsed. (having)** –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

**Remarks:**

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

```
init()
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.
```

```
init()
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.
```

```
notify (notifiable, notification)
on_contract_breached (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters

• contract (Contract) – The contract
• breaches (List[Breach]) – All breaches committed (even if they were resolved)
• resolution (Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None
```

```
on_contract_cancelled (contract, rejectors)
Called whenever at least a partner did not sign the contract
```

```
on_contract_cancelled_ (contract, rejectors)
Called whenever at least a partner did not sign the contract
```

```
on_contract_executed (contract)
Called after successful contract execution for which the agent is one of the partners.
```

```
on_contract_signed (contract)
Called whenever a contract is signed by all partners
```

```
on_contract_signed_ (contract)
Called whenever a contract is signed by all partners
```

```
on_event (event, sender)
on_neg_request_accepted (req_id, mechanism)
Called when a requested negotiation is accepted
```

```
on_neg_request_accepted_ (req_id, mechanism)
Called when a requested negotiation is accepted
```

```
on_neg_request_rejected (req_id, by)
Called when a requested negotiation is rejected
```

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason
on_neg_request_rejected_ (req_id, by)
Called when a requested negotiation is rejected

Parameters
• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_negotiation_failure (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_failure_ (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

Return type None

on_negotiation_success (contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

on_negotiation_success_ (contract, mechanism)
Called whenever a negotiation ends with agreement

Return type None

classmethod read_config (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters
• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:
respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)

Return type Optional[Negotiator]

respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received

Return type Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)
Called to respond to a renegotiation request

Parameters
• agenda (RenegotiationRequest) –
• contract (Contract) –
• breaches (List[Breach]) –

Returns:

Return type Optional[Negotiator]
set_renegotiation_agenda \((contract, breaches)\)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- **contract** \((\text{Contract})\) – The contract being breached
- **breaches** \((\text{List[Breach]}\)) – All breaches on \(contract\)

**Return type** Optional[RenegotiationRequest]

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

sign_contract \((contract)\)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** Optional[str]

step()
Called by the simulator at every simulation step

step()
Called at every time-step. This function is called directly by the world.

SCMLAgent

class negmas.apps.scml.SCMLAgent \((name=None)\)

**Bases:** negmas.situated.Agent

The base for all SCM Agents

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_expect_agreement ((\text{cfp, margin}))</td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable</td>
</tr>
<tr>
<td>confirm_contract_execution ((\text{contract}))</td>
<td>Called before executing any agreement called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached</td>
</tr>
<tr>
<td>confirm_loan ((\text{loan, bankrupt_if_rejected}))</td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
<tr>
<td>confirm_partial_execution ((\text{contract, breaches}))</td>
<td></td>
</tr>
</tbody>
</table>
create(*args, **kwargs) Creates an object and returns a proxy to it.

from_config(config[, section, ...]) Creates an object of this class given the configuration info.

init() Called to initialize the agent after the world is initialized.

init_() The initialization function called by the world directly.

notify(notifiable, notification) Will be called whenever any agent goes bankrupt

on_agent_bankrupt(agent_id) Received whenever money is transferred to the factory or from it.

on_cash_transfer(amount, cause) Called after complete processing of a contract that involved a breach.

on_contract_breach(contract, breaches, ...) Called whenever at least a partner did not sign the contract

on_contract_cancelled(contract, rejectors) Called whenever at least a partner did not sign the contract

on_contract_cancelled_(contract, rejectors) Called whenever at least a partner did not sign the contract

on_contract_executed(contract) Called after successful contract execution for which the agent is one of the partners.

on_contract_nullified(contract, ...) Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

on_contract_signed(contract) Called whenever a contract is signed by all partners

on_contract_signed_(contract) Called whenever a contract is signed by all partners

on_event(event, sender) Received whenever something moves in or out of the factory’s storage

on_inventory_change(product, quantity, cause) Called when a requested negotiation is accepted

on_neg_request_accepted(req_id, mechanism) Called when a requested negotiation is accepted

on_neg_request_accepted_(req_id, mechanism) Called when a requested negotiation is accepted

on_neg_request_rejected(req_id, by) Called when a requested negotiation is rejected

on_neg_request_rejected_(req_id, by) Called when a requested negotiation is rejected

on_negotiation_failure(partners, annotation, ...) Called whenever a negotiation ends without agreement

on_negotiation_failure_(partners, ...) Called whenever a negotiation ends without agreement

on_negotiation_success(contract, mechanism) Called whenever a negotiation ends with agreement

on_negotiation_success_(contract, mechanism) Called whenever a negotiation ends with agreement

on_new_cfp(cfp) Called when a new CFP for a product for which the agent registered interest is published

on_new_report(report) Called whenever a financial report is published.

on_remove_cfp(cfp) Called when a new CFP for a product for which the agent registered interest is removed

read_config(config[, section]) Reads the configuration from a file or a dict and prepares it for parsing

request_negotiation(cfp[, negotiator, ufun]) Requests a negotiation from the AWI while keeping track of available negotiation requests

respond_to_negotiation_request(cfp, partner) Called when a prospective partner requests a negotiation to start

Continued on next page
Table 53 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>respond_to_negotiation_request_</code> (initiator, ... )</td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request</code></td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td><code>set_renegotiation_agenda</code> (contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>sign_contract</code> (contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

**awi**

Returns the Agent-World-Interface through which the agent does all of its actions in the world.

A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

*Return type* SCMLAWI

**id**

The unique ID of this entity

**name**

A convenient name of the entity (intended primarily for printing/logging/debugging).

**requested_negotiations**

The negotiations currently requested by the agent.

*Return type* List[NegotiationRequestInfo]

*Returns* A list of negotiation request information objects (NegotiationRequestInfo)

**running_negotiations**

The negotiations currently requested by the agent.

*Return type* List[RunningNegotiationInfo]

*Returns* A list of negotiation information objects (RunningNegotiationInfo)

**short_type_name**

Returns a short name of the type of this entity

**type_name**

Returns the name of the type of this entity

**unsigned_contracts**

All contracts that are not yet signed.

*Return type* List[Contract]

**uuid**

The unique ID of this entity

**Methods Documentation**

**can_expect_agreement** (cfp, margin)

Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

*Parameters*

- *margin* (int) –
• `cfp` (`CFP`) –

Returns:

`confirm_contract_execution` (`contract`)
Called before executing any agreement

Return type `bool`

`confirm_loan` (`loan`, `bankrupt_if_rejected`)
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

Return type `bool`

`confirm_partial_execution` (`contract`, `breaches`)
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters

• `contract` (`Contract`) – The contract that was breached

• `breaches` (`List[Breach]`) – A list of all the breaches committed.

Remarks:

• Will not be called if both partners committed breaches.

Return type `bool`

`classmethod create` (`*args`, `**kwargs`)
Creates an object and returns a proxy to it.

`classmethod from_config` (`config`, `section=None`, `ignore_children=True`, `try_parsing_children=True`, `scope=None`)
Creates an object of this class given the configuration info

Parameters

• `config` (`Union[str, dict]`) – Either a file name or a dictionary

• `section` (`Optional[str]`) – A section in the file or a key in the dictionary to use for loading params

• `ignore_children` (`bool`) – If true then children will be ignored and there will be a single return

• `try_parsing_children` (`bool`) – If true the children will first be parsed as `ConfigReader` classes if they are not `types` (`e.g. int, str, float, Iterable[int|str|float] (simple)–

• `scope` – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are

• `any children that are to be parsed. (having)–`

Returns An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file).
To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

\texttt{init()} 
Called to initialize the agent \texttt{after} the world is initialized. the AWI is accessible at this point.

\texttt{init}() 
The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.
2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
3. registers interest in all products that the agent can produce or consume in its factory.
4. finally it calls any custom initialization logic implemented in ‘init()’

\textbf{See also:} 
\texttt{init, step}

\texttt{notify}(\texttt{notifiable, notification})

\textbf{on\_agent\_bankrupt}(\texttt{agent\_id}) 
Will be called whenever any agent goes bankrupt

\textbf{Parameters} \texttt{agent\_id}(\texttt{str}) – The ID of the agent that went bankrupt

\textbf{Remarks:}

\begin{itemize}
\item Agents can go bankrupt in two cases:
  \begin{itemize}
  \item Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
  \item Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).
  \end{itemize}
\item All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their \texttt{on\_new\_cfp} and \texttt{respond\_to\_negotiation\_request} callbacks by pulling the bulletin-board using the helper function \texttt{is\_bankrupt} of their AWI.
\end{itemize}

\textbf{Return type} None

\texttt{on\_cash\_transfer}(\texttt{amount, cause}) 
Received whenever money is transferred to the factory or from it.

\textbf{Parameters}

\begin{itemize}
\item \texttt{amount}(\texttt{float}) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
\item \texttt{cause}(\texttt{str}) – The cause of the change. Possibilities include:
  \begin{itemize}
  \item contract: Contract execution
  \item insurance: Received from insurance company
  \item bankruptcy: Liquidated due to bankruptcy
  \item transfer: Arrival of transferred money (when transfer delay in the system is > 0).
  \end{itemize}
\end{itemize}

\textbf{Return type} None

\texttt{on\_contract\_breached}(\texttt{contract, breaches, resolution})
Called after complete processing of a contract that involved a breach.

\textbf{Parameters}
• **contract** (*Contract*) – The contract

• **breaches** (*List[Breach]*) – All breaches committed (even if they were resolved)

• **resolution** (*Optional[Contract]*) – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

**on_contract_cancelled** (*contract, rejectors*)

Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_cancelled_** (*contract, rejectors*)

Called whenever at least a partner did not sign the contract

**Return type** None

**on_contract_executed** (*contract*)

Called after successful contract execution for which the agent is one of the partners.

**Return type** None

**on_contract_nullified** (*contract, bankrupt_partner, compensation*)

Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

**Return type** None

**on_contract_signed** (*contract*)

Called whenever a contract is signed by all partners

**Return type** None

**on_contract_signed_** (*contract*)

Called whenever a contract is signed by all partners

**Return type** None

**on_event** (*event, sender*)

**on_inventory_change** (*product, quantity, cause*)

Received whenever something moves in or out of the factory’s storage

**Parameters**

• **product** (*int*) – Product index.

• **quantity** (*int*) – Negative value for products moving out and positive value for products moving in

• **cause** (*str*) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transport: Arrival of goods (when transportation delay in the system is > 0).

**Return type** None

**on_neg_request_accepted** (*req_id, mechanism*)

Called when a requested negotiation is accepted

**Return type** None

**on_neg_request_accepted_** (*req_id, mechanism*)

Called when a requested negotiation is accepted

**Return type** None

**on_neg_request_rejected** (*req_id, by*)

Called when a requested negotiation is rejected
Parameters

• `req_id (str)` – The request ID passed to `_request_negotiation`

• `by (Optional[List[str]])` – A list of agents that refused to participate or None if the failure was for another reason

`on_neg_request_rejected_(req_id, by)`
Called when a requested negotiation is rejected

Parameters

• `req_id (str)` – The request ID passed to `_request_negotiation`

• `by (Optional[List[str]])` – A list of agents that refused to participate or None if the failure was for another reason

`on_negotiation_failure_(partners, annotation, mechanism, state)`
Called whenever a negotiation ends without agreement

Return type None

`on_negotiation_failure_(partners, annotation, mechanism, state)`
Called whenever a negotiation ends without agreement

Return type None

`on_negotiation_success_(contract, mechanism)`
Called whenever a negotiation ends with agreement

Return type None

`on_negotiation_success_(contract, mechanism)`
Called whenever a negotiation ends with agreement

Return type None

`on_new_cfp(cfp)`
Called when a new CFP for a product for which the agent registered interest is published

`on_new_report(report)`
Called whenever a financial report is published.

Parameters `report (FinancialReport)` – The financial report giving details of the standing of an agent at some time (see `FinancialReport`)

Remarks:

• Agents must opt-in to receive these calls by calling `receive_financial_reports` on their AWI

`on_remove_cfp(cfp)`
Called when a new CFP for a product for which the agent registered interest is removed

`classmethod read_config(config, section=None)`
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• `config (Union[str, dict])` – Either a file name or a dictionary

• `section (Optional[str])` – A section in the file or a key in the dictionary to use for loading params

Return type Dict[str, Any]

Returns A dict ready to be parsed by from_config

Remarks:

`request_negotiation(cfp, negotiator=None, ufun=None)`
Requests a negotiation from the AWI while keeping track of available negotiation requests
Parameters

- **cfp** (*CFP*)
- **negotiator** (*Optional[Negotiator]*)
- **ufun** (*Optional[UtilityFunction]*)

**Return type** *bool*

**Returns** Whether the negotiation request was successful indicating that the partner accepted the negotiation

**respond_to_negotiation_request** (*cfp, partner*)
Called when a prospective partner requests a negotiation to start

**Return type** *Optional[Negotiator]*

**respond_to_negotiation_request** (*initiator, partners, issues, annotation, mechanism, role, req_id*)
Called when a negotiation request is received

**Return type** *Optional[Negotiator]*

**respond_to_renegotiation_request** (*contract, breaches, agenda*)
Called to respond to a renegotiation request

**Parameters**

- **agenda** (*RenegotiationRequest*)
- **contract** (*Contract*)
- **breaches** (*List[Breach]*)

**Returns**:

**Return type** *Optional[Negotiator]*

**set_renegotiation_agenda** (*contract, breaches*)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- **contract** (*Contract*) – The contract being breached
- **breaches** (*List[Breach]*) – All breaches on *contract*

**Return type** *Optional[RenegotiationRequest]*

**Returns** Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

**sign_contract** (*contract*)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** *Optional[str]*

**step()**
Called by the simulator at every simulation step

**step()**
Called at every time-step. This function is called directly by the world.

**FactorySimulator**

```python
class negmas.apps.scml.FactorySimulator(initial_wallet, initial_storage, n_steps, n_products, profiles, max_storage=None)
Bases: abc.ABC
```

6.1. negmas.apps.scml Package
Simulates a factory allowing for prediction of storage/balance in the future.

**Parameters**

- `initial_wallet (float)` – The initial amount of cash in the wallet
- `initial_storage (Dict[int, int])` – initial inventory
- `n_steps (int)` – number of simulation steps
- `n_products (int)` – number of products in the world
- `profiles (List[ManufacturingProfile])` – all profiles that the factory being simulated can run
- `max_storage (Optional[int])` – maximum available storage space.

**Attributes Summary**

- `final_balance` Final balance given everything scheduled so-far
- `fixed_before` Gives the time before which the schedule is fixed.
- `initial_storage` Initial inventory
- `initial_wallet` Initial cash in wallet
- `max_storage` Maximum storage available
- `n_lines` Number of lines
- `n_steps` Number of steps to predict ahead.

**Methods Summary**

- `add_loan(total, t)` Adds a loan at the given time
- `available_storage_at(t)` Returns the available storage of all products at time t.
- `available_storage_to(t)` Returns the available storage of all products up to time t.
- `balance_at(t)` Returns the balance of the factory at time t.
- `balance_to(t)` Returns the balance of the factory up to time t.
- `bookmark()` Sets a bookmark to the current location
- `buy(product, quantity, price, t[, . . . ])` Buy a given quantity of a product for a given price at some time t.
- `delete_bookmark(bookmark_id)` Commits everything since the bookmark so it cannot be rolled back
- `fix_before(t)` Fix the history before this point
- `line_schedules_at(t)` Returns the schedule of each line at a given timestep
- `line_schedules_to(t)` Returns the schedule of each line up to a given timestep
- `loans_at(t)` Returns loans at time t.
- `loans_to(t)` Returns loans up to time t.
- `pay(payment, t[, ignore_money_shortage])` Simulate payment at time t.
- `receive(payment, t)` Simulates receiving payment at time t.
- `reserve(product, quantity, t)` Simulates reserving the given quantity of the given product at times >= t.
- `reserved_storage_at(t)` Returns the reserved storage of all products at time t.
- `reserved_storage_to(t)` Returns the reserved storage of all products up to time t.
- `rollback(bookmark_id)` Rolls back to the given bookmark ID

Continued on next page
### Table 55 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>schedule(job[, ignore_inventory_shortage, ...])</code></td>
<td>Simulates scheduling the given job at its time and line optionally overriding whatever was already scheduled</td>
</tr>
<tr>
<td><code>sell(product, quantity, price, t[, ...])</code></td>
<td>Sell a given quantity of a product for a given price at some time t</td>
</tr>
<tr>
<td><code>set_state(t, storage, wallet, loans, ...)</code></td>
<td>Sets the current state at the given time-step.</td>
</tr>
<tr>
<td><code>storage_at(t)</code></td>
<td>Returns the storage of all products at time t</td>
</tr>
<tr>
<td><code>storage_to(t)</code></td>
<td>Returns the storage of all products up to time t</td>
</tr>
<tr>
<td><code>total_storage_at(t)</code></td>
<td>The total storage at a given time</td>
</tr>
<tr>
<td><code>total_storage_to(t)</code></td>
<td>The total storage up to a given time</td>
</tr>
<tr>
<td><code>transport_to(product, quantity, t[, ...])</code></td>
<td>Simulates transporting products to/from storage at time t</td>
</tr>
<tr>
<td><code>wallet_at(t)</code></td>
<td>Returns the cash in wallet at a given timestep (given all simulated actions)</td>
</tr>
<tr>
<td><code>wallet_to(t)</code></td>
<td>Returns the cash in wallet up to and including time t</td>
</tr>
</tbody>
</table>

### Attributes Documentation

**final_balance**

Final balance given everything scheduled so-far

**Return type** float

**fixed_before**

Gives the time before which the schedule is fixed.

**See also:**

`fix_before`

**initial_storage**

Initial inventory

**Return type** <built-in function array>

**initial_wallet**

Initial cash in wallet

**Return type** float

**max_storage**

Maximum storage available

**Return type** Optional[int]

**n_lines**

Number of lines

**n_steps**

Number of steps to predict ahead.

**Return type** int

### Methods Documentation

**add_loan(total, t)**

Adds a loan at the given time

**Parameters**

- `total (float)` – Total amount of the loan
- `t (int)` – time step to take the loan
Return type \( \text{bool} \)

Returns Success or failure

Remarks:

• Taking a loan is simulated as reception of money. Payment back of the loan is not simulated in this call. To simulate paying back the loan, use \text{pay} at the times of installment payments.

\text{available_storage_at}(t)

Returns the available storage of all products at time \( t \)

Parameters \( t (\text{int}) \) – Time

Return type \( \text{<built-in function array>} \)

Returns An array of size \( n_{\text{products}} \) giving the quantity of each product available at time-step \( t \).

Remarks:

• Available storage is defined as the difference between storage and reserved storage.

• Reserved storage is counted in calls to \text{storage_at}, \text{total_storage_at}, \text{storage_to}, \text{total_storage_to}

• Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

\text{total_storage_to} \text{storage_to} \text{reserved_storage_at}

\text{available_storage_to}(t)

Returns the available storage of all products up to time \( t \).

Parameters \( t (\text{int}) \) – Time

Return type \( \text{<built-in function array>} \)

Returns An array of size \( n_{\text{products}} \times t \) giving the quantity of each product available at every step up to \( t \).

Remarks:

• Available storage is defined as the difference between storage and reserved storage.

• Reserved storage is counted in calls to \text{storage_at}, \text{total_storage_at}, \text{storage_to}, \text{total_storage_to}

• Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

\text{total_storage_to} \text{storage_to} \text{reserved_storage_to}

\text{balance_at}(t)

Returns the balance fo the factory at time \( t \).

Parameters \( t (\text{int}) \) – time

Remarks:

• The balance is defined as the cash in wallet minus loans

See also:

\text{loans_at} \text{wallet_at}

Return type \( \text{float} \)
balance_to(t)
Returns the balance fo the factory up to time t.

Parameters t (int) – time

Remarks:
• The balance is defined as the cash in wallet minus loans

See also:
loans_to wallet_to

Return type <built-in function array>

bookmark()
Sets a bookmark to the current location

Return type int

Returns bookmark ID

Remarks:
• Bookmarks can be used to implement transactions.

See also:
delete_bookmark rollback transaction temporary_transaction

buy (product, quantity, price, t, ignore_money_shortage=True, ignore_space_shortage=True)
Buy a given quantity of a product for a given price at some time t

Parameters
• product (int) – Product to buy (ID/index)
• quantity (int) – quantity to buy
• price (int) – unit price
• t (int) – time
• ignore_money_shortage (bool) – If True, shortage in money will be ignored and the wallet can go negative
• ignore_space_shortage (bool) – Ignore the limit on total storage which may lead to total_storage > max_storage

Return type bool

Returns Success or failure

Remarks:
• buy cannot ever have inventory shortage

See also:
sell

delete_bookmark(bookmark_id)
Commits everything since the bookmark so it cannot be rolled back

Parameters The bookmark ID returned from bookmark (bookmark_id) –

Return type bool

Returns Success/failure

Remarks:
• You can delete bookmarks in the reverse order of their creation only. If the bookmark ID given here is not the one at the top of the bookmarks stack, the deletion will fail (return False).
See also:

delete_bookmark rollback transaction temporary_transaction

**fix_before(t)**
Fix the history before this point

Parameters  
\[ t \]  (int) – time

Return type  
bool

Returns  
Success/failure

Remarks:
- After this function is called at any time-step \( t \), there is no way to change any component of the factory state at any timestep before \( t \).
- This function is useful for fixing any difference between the simulator and the real state (in conjunction with set_state).

See also:

set_state fixed_before

**line_schedules_at(t)**
Returns the schedule of each line at a given timestep

Parameters  
\[ t \]  (int) – time

Return type  
<built-in function array>

Returns  
An array of \( n\_lines \) values giving the schedule up at \( t \).

Remarks:
- A \( NO\_PRODUCTION \) value means no production, otherwise the index of the process being run

**line_schedules_to(t)**
Returns the schedule of each line up to a given timestep

Parameters  
\[ t \]  (int) – time

Return type  
<built-in function array>

Returns  
An array of \( n\_lines \times t \) values giving the schedule up to \( t \).

Remarks:
- A \( NO\_PRODUCTION \) value means no production, otherwise the index of the process being run

**loans_at(t)**
Returns loans at time \( t \)

Parameters  
\[ t \]  (int) – time

Return type  
float

**loans_to(t)**
Returns loans up to time \( t \)

Parameters  
\[ t \]  (int) – time

Return type  
<built-in function array>

Returns  
An array of \( t \) real numbers giving the loans registered at time-steps up to \( t \)

**pay(payment, t, ignore_money_shortage=True)**
Simulate payment at time \( t \)

Parameters:
- \[ payment \]  (float) – Amount payed
- \[ t \]  (int) – time
• **ignore_money_shortage** *(bool)* – If True, shortage in money will be ignored and the wallet can go negative

  **Return type** bool

  **Returns** Success or failure

**receive** *(payment, t)*

Simulates receiving payment at time t

  **Parameters**

  • **payment** *(float)* – Amount received

  • **t** *(int)* – time

  **Return type** bool

  **Returns** Success or failure

**reserve** *(product, quantity, t)*

Simulates reserving the given quantity of the given product at times \(\geq t\).

  **Parameters**

  • **product** *(int)* – Index/ID of the product being reserved

  • **quantity** *(int)* – quantity being reserved

  • **t** *(int)* – time

  **Return type** bool

  **Returns** Success/failure

  **Remarks:**

  • Reserved products show in calls to `storage_at`, `total_storage_at` etc.

  • Reserving a product does nothing more than mark some quantity as reserved for calls to `reserved_storage_at` and `available_storage_at`.

  • This feature can be used to simulate inventory hiding commands in the real factory and to avoid double counting of inventory when calculating needs for future contracts.

**reserved_storage_at** *(t)*

Returns the reserved storage of all products at time \(t\)

  **Parameters** \(t\) *(int)* – Time

  **Return type** <built-in function array>

  **Returns** An array of size \(n\_products\) giving the quantity of each product reserved at time-step \(t\).

  **Remarks:**

  • Reserved storage is counted in calls to `storage_at`, `total_storage_at`, `storage_to`, `total_storage_to`

  • Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

  **See also:**

  `total_storage_to` `storage_to` `reserved_storage_at`

**reserved_storage_to** *(t)*

Returns the reserved storage of all products up to time \(t\)

  **Parameters** \(t\) *(int)* – Time

  **Return type** <built-in function array>
Returns An array of size \( n_{\text{products}} \times t \) giving the quantity of each product reserved at every step up to \( t \).

Remarks:
- Reserved storage is counted in calls to \texttt{storage\_at}, \texttt{total\_storage\_at}, \texttt{storage\_to}, \texttt{total\_storage\_to}
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:
\texttt{total\_storage\_at}, \texttt{storage\_at}, \texttt{reserved\_storage\_at}

\texttt{rollback}(\texttt{bookmark\_id})
Rolls back to the given bookmark ID

Parameters The bookmark ID returned from \texttt{bookmark}(\texttt{bookmark\_id}) –

Remarks:
- You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the rollback will fail (return False)

See also:
\texttt{delete\_bookmark}, \texttt{rollback}, \texttt{transaction}, \texttt{temporary\_transaction}

Return type \texttt{bool}

\texttt{schedule}(\texttt{job}, \texttt{ignore\_inventory\_shortage=True}, \texttt{ignore\_money\_shortage=True}, \texttt{ignore\_space\_shortage=True}, \texttt{override=True})
Simulates scheduling the given job at its time and line optionally overriding whatever was already scheduled

Parameters
- \texttt{job} (\texttt{Job}) – Production job
- \texttt{ignore\_inventory\_shortage} – If true shortages in inputs will be ignored
- \texttt{ignore\_money\_shortage} – If true, shortage in money will be ignored
- \texttt{ignore\_space\_shortage} – If true, shortage in space will be ignored
- \texttt{override} – Whether the job should override any already registered job at its time-step

Return type \texttt{bool}

Returns Success/failure

\texttt{sell}(\texttt{product}, \texttt{quantity}, \texttt{price}, \texttt{t}, \texttt{ignore\_money\_shortage=True}, \texttt{ignore\_inventory\_shortage=True})
sell a given quantity of a product for a given price at some time \( t \)

Parameters
- \texttt{product} (\texttt{int}) – Index/ID of the product to be sold
- \texttt{quantity} (\texttt{int}) – quantity to be sold
- \texttt{price} (\texttt{int}) – unit price
- \texttt{t} (\texttt{int}) – time
- \texttt{ignore\_money\_shortage} (\texttt{bool}) – If True, shortage in money will be ignored and the wallet can go negative
- \texttt{ignore\_inventory\_shortage} (\texttt{bool}) – Ignore shortage in the \texttt{product} which may lead to negative storage[product]
Return type \texttt{bool}

Returns Success or failure

Remarks:
• sell cannot ever have space shortage

See also:
\texttt{buy}

\texttt{set\_state}(t, storage, wallet, loans, line\_schedules)
Sets the current state at the given time-step. It implicitly causes a \texttt{fix\_before}(t + 1)

Parameters
\begin{itemize}
  \item \texttt{t (int)} – Time step to set the state at
  \item \texttt{storage (\textless built-in function array\textgreater)} – quantity of every product (array of integers of size \texttt{n\_products})
  \item \texttt{wallet (float)} – Cash in wallet
  \item \texttt{loans (float)} – Loans
  \item \texttt{line\_schedules (\textless built-in function array\textgreater)} – Line schedules (array of process numbers/\texttt{NO\_PRODUCTION} of size \texttt{n\_lines})
\end{itemize}

Return type None

\texttt{storage\_at}(t)
Returns the storage of all products \textit{at} time \texttt{t}

Parameters \texttt{t (int)} – Time

Return type \texttt{<built-in function array>}

Returns An array of size \texttt{n\_products} giving the quantity of each product in storage at time-step \texttt{t}.

See also:
\texttt{storage\_to wallet\_at}

\texttt{storage\_to}(t)
Returns the storage of all products \textit{up to} time \texttt{t}

Parameters \texttt{t (int)} – Time

Return type \texttt{<built-in function array>}

Returns An array of size \texttt{n\_products} * \texttt{t} giving the quantity of each product in storage at every step up to \texttt{t}.

\texttt{total\_storage\_at}(t)
The total storage \textit{at} a given time

Parameters \texttt{t (int)} – time

Return type \texttt{int}

Returns an integer giving the total quantity of stored products in the inventory at timestep \texttt{t}

See also:
\texttt{total\_storage\_to storage\_at}

\texttt{total\_storage\_to}(t)
The total storage \textit{up to} a given time

Parameters \texttt{t (int)} – time

Return type \texttt{<built-in function array>}

6.1. \texttt{negmas.apps.scml Package}
Returns an array of size $t$ giving the total quantity of stored products in the inventory up to timestep $t$.

See also:

$\text{total\_storage\_at\ storage\_to}$

$\text{transport\_to}(product,\ quantity,\ t,\ \text{ignore\_inventory\_shortage=}True,\ \text{ignore\_space\_shortage=}True)$

Simulates transporting products to/from storage at time $t$.

Parameters

- $\text{product}$ ($\text{int}$) – product ID (index)
- $\text{quantity}$ ($\text{int}$) – quantity to transport
- $t$ ($\text{int}$) – time
- $\text{ignore\_inventory\_shortage}$ ($\text{bool}$) – Ignore shortage in the product which may lead to negative storage[$\text{product}$]
- $\text{ignore\_space\_shortage}$ ($\text{bool}$) – Ignore the limit on total storage which may lead to $\text{total\_storage} > \text{max\_storage}$

Return type $\text{bool}$

Returns Success or failure

$\text{wallet\_at}(t)$

Returns the cash in wallet at a given timestep (given all simulated actions)

Parameters $t$ ($\text{int}$)

Returns:

Return type $\text{float}$

$\text{wallet\_to}(t)$

Returns the cash in wallet up to and including time $t$.

Parameters $t$ ($\text{int}$) – Time

Returns:

Return type $<$built-in function array$>$

SlowFactorySimulator

class $\text{negmas.apps.scml.SlowFactorySimulator}$ ($\text{initial\_wallet, initial\_storage, n\_steps, n\_products, profiles, max\_storage}$)

Bases: $\text{negmas.apps.scml.FactorySimulator}$

A slow factory simulator that runs an internal factory to find-out what will happen in the future.

Remarks:

- It is much faster to always access the properties/methods of this class in ascending time. If that is not the case, each time reversal will cause a complete reset.
- It is recommended to call $\text{fix\_before()}$ to fix the past once a production step is completed. That will speed up operations.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>final_balance</td>
<td>Final balance given everything scheduled so-far</td>
</tr>
<tr>
<td>fixed_before</td>
<td>Gives the time before which the schedule is fixed.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 56 – continued from previous page

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_storage</td>
<td>Initial inventory</td>
</tr>
<tr>
<td>initial_wallet</td>
<td>Initial cash in wallet</td>
</tr>
<tr>
<td>max_storage</td>
<td>Maximum storage available</td>
</tr>
<tr>
<td>n_lines</td>
<td>Number of lines</td>
</tr>
<tr>
<td>n_steps</td>
<td>Number of steps to predict ahead.</td>
</tr>
</tbody>
</table>

#### Methods Summary

- **add_loan**(total, t) Adds a loan at the given time
- **available_storage_at(t)** Returns the available storage of all products at time t
- **available_storage_to(t)** Returns the available storage of all products up to time t.
- **balance_at(t)** Returns the balance to the factory at time t.
- **balance_to(t)** Returns the balance to the factory up to time t.
- **bookmark**() Sets a bookmark to the current location
- **buy**(product, quantity, price, t[, . . .]) Buy a given quantity of a product for a given price at some time t
- **delete_bookmark**(bookmark_id) Commits everything since the bookmark so it cannot be rolled back
- **fix_before**(t) Fix the history before this point
- **goto**(t) Steps the factory to the end of step t :type t: int :param t: time
- **line_schedules_at(t)** Returns the schedule of each line at a given timestep
- **line_schedules_to(t)** Returns the schedule of each line up to a given timestep
- **loans_at(t)** Returns loans at time t
- **loans_to(t)** Returns loans up to time t
- **pay**(payment, t[, ignore_money_shortage]) Simulate payment at time t
- **pay**(payment, t) Simulates receiving payment at time t
- **reserve**(product, quantity, t) Simulates reserving the given quantity of the given product at times >= t.
- **reserved_storage_at(t)** Returns the reserved storage of all products at time t
- **reserved_storage_to(t)** Returns the reserved storage of all products up to time t
- **reset_to(t)**
- **rollback**(bookmark_id) Rolls back to the given bookmark ID
- **schedule**(job[, ignore_inventory_shortage, ...]) Simulates scheduling the given job at its time and line optionally overriding whatever was already scheduled
- **sell**(product, quantity, price, t[, . . .]) sell a given quantity of a product for a given price at some time t
- **set_state**(t, storage, wallet, loans, ...) Sets the current state at the given time-step.
- **storage_at(t)** Returns the storage of all products at time t
- **storage_to(t)** Returns the storage of all products up to time t
- **total_storage_at(t)** The total storage at a given time
- **total_storage_to(t)** The total storage up to a given time
- **transport_to**(product, quantity, t[, . . .]) Simulates transporting products to/from storage at time t

**rtype** None

Continued on next page
<table>
<thead>
<tr>
<th>Attribute / Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallet_at(t)</td>
<td>Returns the cash in wallet at a given timestep (given all simulated actions)</td>
</tr>
<tr>
<td>wallet_to(t)</td>
<td>Returns the cash in wallet up to and including time t.</td>
</tr>
</tbody>
</table>

**Attributes Documentation**

**final_balance**
- Final balance given everything scheduled so-far
  
  **Return type** float

**fixed_before**
- Gives the time before which the schedule is fixed.

  **See also:**
  - fix_before

**initial_storage**
- Initial inventory

  **Return type** <built-in function array>

**initial_wallet**
- Initial cash in wallet

  **Return type** float

**max_storage**
- Maximum storage available

  **Return type** Optional[int]

**n_lines**
- Number of lines

**n_steps**
- Number of steps to predict ahead.

  **Return type** int

**Methods Documentation**

**add_loan** (total, t)
- Adds a loan at the given time

  **Parameters**
  - total (float) – Total amount of the loan
  - t (int) – time step to take the loan

  **Return type** bool

  **Returns** Success or failure

  **Remarks:**
  - Taking a loan is simulated as reception of money. Payment back of the loan is not simulated in this call. To simulate paying back the loan, use pay at the times of installment payments.

**available_storage_at** (t)
- Returns the available storage of all products at time t

  **Parameters** t (int) – Time

  **Return type** <built-in function array>
Returns An array of size \( n_{\text{products}} \) giving the quantity of each product available at time-step \( t \).

Remarks:

- Available storage is defined as the difference between storage and reserved storage.
- Reserved storage is counted in calls to \texttt{storage\_at}, \texttt{total\_storage\_at}, \texttt{storage\_to}, \texttt{total\_storage\_to}
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

\texttt{total\_storage\_to} \texttt{storage\_to} \texttt{reserved\_storage\_at}

\texttt{available\_storage\_to}(t)

Returns the available storage of all products \textit{up to} time \( t \).

Parameters \( t \) (\texttt{int}) – Time

Return type \texttt{<built-in function array>}

Returns An array of size \( n_{\text{products}} \times t \) giving the quantity of each product available at every step up to \( t \).

Remarks:

- Available storage is defined as the difference between storage and reserved storage.
- Reserved storage is counted in calls to \texttt{storage\_at}, \texttt{total\_storage\_at}, \texttt{storage\_to}, \texttt{total\_storage\_to}
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

\texttt{total\_storage\_to} \texttt{storage\_to} \texttt{reserved\_storage\_to}

\texttt{balance\_at}(t)

Returns the balance for the factory at time \( t \).

Parameters \( t \) (\texttt{int}) – time

Remarks:

- The balance is defined as the cash in wallet minus loans

See also:

\texttt{loans\_at} \texttt{wallet\_at}

Return type \texttt{float}

\texttt{balance\_to}(t)

Returns the balance for the factory \textit{up to} time \( t \).

Parameters \( t \) (\texttt{int}) – time

Remarks:

- The balance is defined as the cash in wallet minus loans

See also:

\texttt{loans\_to} \texttt{wallet\_to}

Return type \texttt{<built-in function array>}
bookmark()
Sets a bookmark to the current location

Return type int
Returns bookmark ID

Remarks:
• Bookmarks can be used to implement transactions.

See also:
dele_bookmark rollback transaction temporary_transaction

buy (product, quantity, price, t, ignore_money_shortage=True, ignore_space_shortage=True)
Buy a given quantity of a product for a given price at some time t

Parameters
• product (int) – Product to buy (ID/index)
• quantity (int) – quantity to buy
• price (int) – unit price
• t (int) – time
• ignore_money_shortage (bool) – If True, shortage in money will be ignored and the wallet can go negative
• ignore_space_shortage (bool) – Ignore the limit on total storage which may lead to total_storage > max_storage

Return type bool
Returns Success or failure

Remarks:
• buy cannot ever have inventory shortage

See also:
sell
delete_bookmark (bookmark_id)
Commits everything since the bookmark so it cannot be rolled back

Parameters The bookmark ID returned from bookmark (bookmark_id)

Return type bool
Returns Success/failure

Remarks:
• You can delete bookmarks in the reverse order of their creation only. If the bookmark ID given here is not the one at the top of the bookmarks stack, the deletion will fail (return False).

See also:
dele_bookmark rollback transaction temporary_transaction

fix_before(t)
Fix the history before this point

Parameters t (int) – time

Return type bool
Returns Success/failure

Remarks:
• After this function is called at any time-step \( t \), there is no way to change any component of the factory state at any timestep before \( t \).

• This function is useful for fixing any difference between the simulator and the real state (in conjunction with \texttt{set_state}).

See also:

\texttt{set_state fixed_before}

\textbf{goto}(t)

Steps the factory to the end of step \( t \):

\textbf{type} \( int \) :param \( t \): time

\textbf{Returns}:

\textbf{Return type} None

\textbf{line_schedules_at}(\( t \))

Returns the schedule of each line at a given timestep

\textbf{Parameters} \( t \) (int) – time

\textbf{Return type} \texttt{<built-in function array>}

\textbf{Returns} An array of \( n_{lines} \) values giving the schedule up at \( t \).

\textbf{Remarks}:

• A \texttt{NO_PRODUCTION} value means no production, otherwise the index of the process being run

\textbf{line_schedules_to}(\( t \))

Returns the schedule of each line up to a given timestep

\textbf{Parameters} \( t \) (int) – time

\textbf{Return type} \texttt{<built-in function array>}

\textbf{Returns} An array of \( n_{lines} * t \) values giving the schedule up to \( t \).

\textbf{Remarks}:

• A \texttt{NO_PRODUCTION} value means no production, otherwise the index of the process being run

\textbf{loans_at}(\( t \))

Returns loans at time \( t \)

\textbf{Parameters} \( t \) (int) – time

\textbf{Return type} float

\textbf{loans_to}(\( t \))

Returns loans up to time \( t \)

\textbf{Parameters} \( t \) (int) – time

\textbf{Return type} float

\textbf{Returns} An array of \( t \) real numbers giving the loans registered at time-steps up to \( t \).

\textbf{pay}(pay, t, ignore_money_shortage=True)

Simulate payment at time \( t \)

\textbf{Parameters}

• \texttt{payment} (float) – Amount payed

• \( t \) (int) – time

• ignore_money_shortage (bool) – If True, shortage in money will be ignored and the wallet can go negative

\textbf{Return type} bool

\textbf{Returns} Success or failure
**receive** *(payment, t)*
Simulates receiving payment at time t

Parameters
- **payment** *(float)* – Amount received
- **t** *(int)* – time

Return type **bool**

Returns Success or failure

**reserve** *(product, quantity, t)*
Simulates reserving the given quantity of the given product at times >= t.

Parameters
- **product** *(int)* – Index/ID of the product being reserved
- **quantity** *(int)* – quantity being reserved
- **t** *(int)* – time

Return type **bool**

Returns Success/failure

Remarks:
- Reserved products show in calls to `storage_at`, `total_storage_at` etc.
- Reserving a product does nothing more than mark some quantity as reserved for calls to `reserved_storage_at` and `available_storage_at`.
- This feature can be used to simulate inventory hiding commands in the real factory and to avoid double counting of inventory when calculating needs for future contracts.

**reserved_storage_at** *(t)*
Returns the reserved storage of all products at time t

Parameters **t** *(int)* – Time

Return type *<built-in function array>*

Returns An array of size `n_products` giving the quantity of each product reserved at time-step t.

Remarks:
- Reserved storage **is counted** in calls to `storage_at`, `total_storage_at`, `storage_to`, `total_storage_to`
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

`total_storage_to`, `storage_to`, `reserved_storage_at`

**reserved_storage_to** *(t)*
Returns the reserved storage of all products up to time t

Parameters **t** *(int)* – Time

Return type *<built-in function array>*

Returns An array of size `n_products * t` giving the quantity of each product reserved at every step up to t.

Remarks:
- Reserved storage **is counted** in calls to `storage_at`, `total_storage_at`, `storage_to`, `total_storage_to`
• Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

\[ \text{total\_storage\_at} \quad \text{storage\_at} \quad \text{reserved\_storage\_at} \]

\text{reset\_to}(t)

Return type None

\text{rollback}(\text{bookmark\_id})

Rolls back to the given bookmark ID

Parameters The \text{bookmark\_id} \text{ returned from bookmark}(\text{bookmark\_id}) –

Remarks:

• You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the rollback will fail (return False)

See also:

\text{delete\_bookmark} \quad \text{rollback} \quad \text{transaction} \quad \text{temporary\_transaction}

Return type \text{bool}

\text{schedule}(\text{job}, \text{ignore\_inventory\_shortage}=\text{True}, \text{ignore\_money\_shortage}=\text{True}, \text{ignore\_space\_shortage}=\text{True}, \text{override}=\text{True})

Simulates scheduling the given job at its time and line optionally overriding whatever was already scheduled

Parameters

• \text{job} (\text{Job}) – Production job

• \text{ignore\_inventory\_shortage} – If true shortages in inputs will be ignored

• \text{ignore\_money\_shortage} – If true, shortage in money will be ignored

• \text{ignore\_space\_shortage} – If true, shortage in space will be ignored

• \text{override} – Whether the job should override any already registered job at its time-step

Return type \text{bool}

Returns Success/failure

\text{sell}(\text{product}, \text{quantity}, \text{price}, \text{t}, \text{ignore\_money\_shortage}=\text{True}, \text{ignore\_inventory\_shortage}=\text{True})

sell a given quantity of a product for a given price at some time \text{t}

Parameters

• \text{product} (\text{int}) – Index/ID of the product to be sold

• \text{quantity} (\text{int}) – quantity to be sold

• \text{price} (\text{int}) – unit price

• \text{t} (\text{int}) – time

• \text{ignore\_money\_shortage} (\text{bool}) – If True, shortage in money will be ignored and the wallet can go negative

• \text{ignore\_inventory\_shortage} (\text{bool}) – Ignore shortage in the \text{product} which may lead to negative storage[product]

Return type \text{bool}

Returns Success or failure

Remarks:
• sell cannot ever have space shortage

See also:

buy

set_state \((t, \text{storage, wallet, loans, line_schedules})\)
Sets the current state at the given time-step. It implicitly causes a fix_before\((t + 1)\)

Parameters

- \(t\) \((\text{int})\) – Time step to set the state at
- storage \((<\text{built-in function array}>\)) – quantity of every product (array of integers of size \(n\_products\))
- wallet \((\text{float})\) – Cash in wallet
- loans \((\text{float})\) – Loans
- line_schedules \((<\text{built-in function array}>\)) – Line schedules (array of process numbers/NO_PRODUCTION of size \(n\_lines\))

Return type None

storage_at \((t)\)
Returns the storage of all products at time \(t\)

Parameters \(t\) \((\text{int})\) – Time

Return type <built-in function array>

Returns An array of size \(n\_products\) giving the quantity of each product in storage at time-step \(t\).

See also:

storage_to wallet_at

storage_to \((t)\)
Returns the storage of all products up to time \(t\)

Parameters \(t\) \((\text{int})\) – Time

Return type <built-in function array>

Returns An array of size \(n\_products \times t\) giving the quantity of each product in storage at every step up to \(t\).

total_storage_at \((t)\)
The total storage at a given time

Parameters \(t\) \((\text{int})\) – time

Return type int

Returns an integer giving the total quantity of stored products in the inventory at timestep \(t\).

See also:

total_storage_to storage_at

total_storage_to \((t)\)
The total storage up to a given time

Parameters \(t\) \((\text{int})\) – time

Return type <built-in function array>

Returns an array of size \(t\) giving the total quantity of stored products in the inventory up to timestep \(t\).
See also:

\texttt{total\_storage\_at\ storage\_to}

\texttt{transport\_to (product, quantity, t, ignore\_inventory\_shortage=True, ignore\_space\_shortage=True)}

Simulates transporting products to/from storage at time t

Parameters

- \texttt{product (int)} – product ID (index)
- \texttt{quantity (int)} – quantity to transport
- \texttt{t (int)} – time
- \texttt{ignore\_inventory\_shortage (bool)} – Ignore shortage in the \texttt{product} which may lead to negative storage[product]
- \texttt{ignore\_space\_shortage (bool)} – Ignore the limit on total storage which may lead to total\_storage > max\_storage

Return type: \texttt{bool}

Returns: Success or failure

\texttt{wallet\_at (t)}

Returns the cash in wallet at a given timestep (given all simulated actions)

Parameters \texttt{t (int)} –

Returns:

Return type: \texttt{float}

\texttt{wallet\_to (t)}

Returns the cash in wallet up to and including time t.

Parameters \texttt{t (int)} – Time

Returns:

Return type: \texttt{<built-in function array>}

\textbf{FastFactorySimulator}

class \texttt{negmas.apps.scml.FastFactorySimulator (initial\_wallet, initial\_storage, n\_steps, n\_products, profiles, max\_storage)}

Bases: \texttt{negmas.apps.scml.FactorySimulator}

A faster implementation of the \texttt{FactorySimulator} interface (compared with \texttt{SlowFactorySimulator}).

\textbf{Attributes Summary}

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{final_balance}</td>
<td>Final balance given everything scheduled so-far</td>
</tr>
<tr>
<td>\texttt{fixed_before}</td>
<td>Gives the time before which the schedule is fixed.</td>
</tr>
<tr>
<td>\texttt{initial_storage}</td>
<td>Initial inventory</td>
</tr>
<tr>
<td>\texttt{initial_wallet}</td>
<td>Initial cash in wallet</td>
</tr>
<tr>
<td>\texttt{max_storage}</td>
<td>Maximum storage available</td>
</tr>
<tr>
<td>\texttt{n_lines}</td>
<td>Number of lines</td>
</tr>
<tr>
<td>\texttt{n_steps}</td>
<td>Number of steps to predict ahead.</td>
</tr>
</tbody>
</table>
### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_loan(t)</code></td>
<td>Adds a loan at the given time.</td>
</tr>
<tr>
<td><code>available_storage_at(t)</code></td>
<td>Returns the available storage of all products at time t.</td>
</tr>
<tr>
<td><code>available_storage_to(t)</code></td>
<td>Returns the available storage of all products up to time t.</td>
</tr>
<tr>
<td><code>balance_at(t)</code></td>
<td>Returns the balance for the factory at time t.</td>
</tr>
<tr>
<td><code>balance_to(t)</code></td>
<td>Returns the balance for the factory up to time t.</td>
</tr>
<tr>
<td><code>bookmark()</code></td>
<td>Sets a bookmark to the current location.</td>
</tr>
<tr>
<td><code>buy(product, quantity, price, t[, ...])</code></td>
<td>Buy a given quantity of a product for a given price at some time t.</td>
</tr>
<tr>
<td><code>delete_bookmark(bookmark_id)</code></td>
<td>Commits everything since the bookmark so it cannot be rolled back.</td>
</tr>
<tr>
<td><code>fix_before(t)</code></td>
<td>Fix the history before this point.</td>
</tr>
<tr>
<td><code>init(*args, **kwargs)</code></td>
<td></td>
</tr>
<tr>
<td><code>line_schedules_at(t)</code></td>
<td>Returns the schedule of each line at a given timestep.</td>
</tr>
<tr>
<td><code>line_schedules_to(t)</code></td>
<td>Returns the schedule of each line up to a given timestep.</td>
</tr>
<tr>
<td><code>loans_at(t)</code></td>
<td>Returns loans at time t.</td>
</tr>
<tr>
<td><code>loans_to(t)</code></td>
<td>Returns loans up to time t.</td>
</tr>
<tr>
<td><code>pay(payment, t[, ignore_money_shortage])</code></td>
<td>Simulate payment at time t.</td>
</tr>
<tr>
<td><code>receive(payment, t)</code></td>
<td>Simulates receiving payment at time t.</td>
</tr>
<tr>
<td><code>reserve(product, quantity, t)</code></td>
<td>Simulates reserving the given quantity of the given product at times &gt;= t.</td>
</tr>
<tr>
<td><code>reserved_storage_at(t)</code></td>
<td>Returns the reserved storage of all products at time t.</td>
</tr>
<tr>
<td><code>reserved_storage_to(t)</code></td>
<td>Returns the reserved storage of all products up to time t.</td>
</tr>
<tr>
<td><code>rollback(bookmark_id)</code></td>
<td>Rolls back to the given bookmark ID.</td>
</tr>
<tr>
<td><code>schedule(job[, ignore_inventory_shortage, ...])</code></td>
<td>Simulates scheduling the given job at its time and line optionally overriding whatever was already scheduled</td>
</tr>
<tr>
<td><code>sell(product, quantity, price, t[, ...])</code></td>
<td>sell a given quantity of a product for a given price at some time t</td>
</tr>
<tr>
<td><code>set_state(t, storage, wallet, loans, ...)</code></td>
<td>Sets the current state at the given time-step.</td>
</tr>
<tr>
<td><code>storage_at(t)</code></td>
<td>Returns the storage of all products at time t.</td>
</tr>
<tr>
<td><code>storage_to(t)</code></td>
<td>Returns the storage of all products up to time t.</td>
</tr>
<tr>
<td><code>total_storage_at(t)</code></td>
<td>The total storage at a given time.</td>
</tr>
<tr>
<td><code>total_storage_to(t)</code></td>
<td>The total storage up to a given time.</td>
</tr>
<tr>
<td><code>transport_to(product, quantity, t[, ...])</code></td>
<td>Simulates transporting products to/from storage at time t.</td>
</tr>
<tr>
<td><code>wallet_at(t)</code></td>
<td>Returns the cash in wallet at a given timestep (given all simulated actions)</td>
</tr>
<tr>
<td><code>wallet_to(t)</code></td>
<td>Returns the cash in wallet up to and including time t.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

- **final_balance**
  Final balance given everything scheduled so-far
  
  **Return type**: `float`

- **fixed_before**
  Gives the time before which the schedule is fixed.
See also:

**fix.before**

**initial_storage**
Initial inventory

*Return type* `<built-in function array>`

**initial_wallet**
Initial cash in wallet

*Return type* `float`

**max_storage**
Maximum storage available

*Return type* `Optional[int]`

**n_lines**
Number of lines

**n_steps**
Number of steps to predict ahead.

*Return type* `int`

### Methods Documentation

**add_loan** *(total, t)*
Adds a loan at the given time

**Parameters**

- `total` *(float)* – Total amount of the loan
- `t` *(int)* – time step to take the loan

*Return type* `bool`

*Returns* Success or failure

Remarks:

- Taking a loan is simulated as reception of money. Payment back of the loan is not simulated in this call. To simulate paying back the loan, use `pay` at the times of installment payments.

**available_storage_at** *(t)*
Returns the **available** storage of all products at time t

**Parameters** `t` *(int)* – Time

*Return type* `<built-in function array>`

*Returns* An array of size `n_products` giving the quantity of each product available at time-step t.

Remarks:

- Available storage is defined as the difference between storage and reserved storage.
- Reserved storage is counted in calls to `storage_at`, `total_storage_at`, `storage_to`, and `total_storage_to`
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

`total_storage_to`, `storage_to`, `reserved_storage_at`
**available_storage_to**(*t*)

Returns the available storage of all products up to time t.

**Parameters**

- **t** *(int) – Time*

**Return type** <built-in function array>

**Returns**

An array of size \(n_{products} \times t\) giving the quantity of each product available at every step up to t.

**Remarks:**

- Available storage is defined as the difference between storage and reserved storage.
- Reserved storage is counted in calls to `storage_at`, `total_storage_at`, `storage_to`, `total_storage_to`.
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

**See also:**

- `total_storage_to`
- `storage_to`
- `reserved_storage_to`

**balance_at**(*t*)

Returns the balance for the factory at time t.

**Parameters**

- **t** *(int) – time*

**Remarks:**

- The balance is defined as the cash in wallet minus loans

**See also:**

- `loans_at`
- `wallet_at`

**Return type** float

**balance_to**(*t*)

Returns the balance for the factory up to time t.

**Parameters**

- **t** *(int) – time*

**Remarks:**

- The balance is defined as the cash in wallet minus loans

**See also:**

- `loans_to`
- `wallet_to`

**Return type** <built-in function array>

**bookmark**()

Sets a bookmark to the current location

**Return type** int

**Returns** bookmark ID

**Remarks:**

- Bookmarks can be used to implement transactions.

**See also:**

- `delete_bookmark`
- `rollback`
- `transaction`
- `temporary_transaction`

**buy**(product, quantity, price, t, ignore_money_shortage=True, ignore_space_shortage=True)

Buy a given quantity of a product for a given price at some time t
Parameters

- **product** *(int)* – Product to buy (ID/index)
- **quantity** *(int)* – quantity to buy
- **price** *(int)* – unit price
- **t** *(int)* – time
- **ignore_money_shortage** *(bool)* – If True, shortage in money will be ignored and the wallet can go negative
- **ignore_space_shortage** *(bool)* – Ignore the limit on total storage which may lead to total_storage > max_storage

**Return type** `bool`

**Returns** Success or failure

**Remarks:**
- buy cannot ever have inventory shortage

**See also:**

`sell`

**delete_bookmark** *(bookmark_id)*

Commits everything since the bookmark so it cannot be rolled back

**Parameters**
The bookmark ID returned from bookmark *(bookmark_id)* –

**Return type** `bool`

**Returns** Success/failure

**Remarks:**
- You can delete bookmarks in the reverse order of their creation only. If the bookmark ID given here is not the one at the top of the bookmarks stack, the deletion will fail (return False).

**See also:**

`delete_bookmark rollback transaction temporary_transaction`

**fix_before** *(t)*

Fix the history before this point

**Parameters**
`t` *(int)* – time

**Return type** `bool`

**Returns** Success/failure

**Remarks:**
- After this function is called at any time-step t, there is no way to change any component of the factory state at any timestep before t.
- This function is useful for fixing any difference between the simulator and the real state (in conjunction with `set_state`).

**See also:**

`set_state fixed_before`

**init** *(*args, **kwargs)*

**line_schedules_at** *(t)*

Returns the schedule of each line at a given timestep

**Parameters**
`t` *(int)* – time

**Return type** `<built-in function array>`
Returns An array of $n\_lines$ values giving the schedule up at $t$.

Remarks:
• A `NO\_PRODUCTION` value means no production, otherwise the index of the process being run

`line\_schedules\_to(t)`
Returns the schedule of each line up to a given timestep

Parameters $t$ (`int`) – time  
Return type <built-in function array>  
Returns An array of $n\_lines * t$ values giving the schedule up to $t$.

Remarks:
• A `NO\_PRODUCTION` value means no production, otherwise the index of the process being run

`loans\_at(t)`
Returns loans at time $t$

Parameters $t$ (`int`) – time  
Return type `float`

`loans\_to(t)`
Returns loans up to time $t$

Parameters $t$ (`int`) – time  
Return type <built-in function array>  
Returns An array of $t$ real numbers giving the loans registered at time-steps up to $t$

`pay(payment, t, ignore\_money\_shortage=True)`
Simulate payment at time $t$

Parameters
• `payment` (`float`) – Amount paid  
• $t$ (`int`) – time  
• `ignore\_money\_shortage` (`bool`) – If True, shortage in money will be ignored and the wallet can go negative

Return type `bool`

Returns Success or failure

`receive(payment, t)`
Simulates receiving payment at time $t$

Parameters
• `payment` (`float`) – Amount received  
• $t$ (`int`) – time

Return type `bool`

Returns Success or failure

`reserve(product, quantity, t)`
Simulates reserving the given quantity of the given product at times $\geq t$

Parameters
• `product` (`int`) – Index/ID of the product being reserved  
• `quantity` (`int`) – quantity being reserved  
• $t$ (`int`) – time
Return type **bool**

**Returns** Success/failure

Remarks:

- Reserved products show in calls to `storage_at`, `total_storage_at` etc.
- Reserving a product does nothing more than mark some quantity as reserved for calls to `reserved_storage_at` and `available_storage_at`.
- This feature can be used to simulate inventory hiding commands in the real factory and to avoid double counting of inventory when calculating needs for future contracts.

### `reserved_storage_at(t)`

Returns the *reserved* storage of all products at time `t`

**Parameters** `t (int)` – Time

**Return type** `<built-in function array>`

**Returns** An array of size `n_products` giving the quantity of each product reserved at time-step `t`.

Remarks:

- Reserved storage is counted in calls to `storage_at`, `total_storage_at`, `storage_to`, `total_storage_to`
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

`total_storage_to` `storage_to` `reserved_storage_at`

### `reserved_storage_to(t)`

Returns the *reserved* storage of all products up to time `t`

**Parameters** `t (int)` – Time

**Return type** `<built-in function array>`

**Returns** An array of size `n_products`* `t` giving the quantity of each product reserved at every step up to `t`.

Remarks:

- Reserved storage is counted in calls to `storage_at`, `total_storage_at`, `storage_to`, `total_storage_to`
- Reserving quantities of products is a tool that can be used to avoid double counting availability of given products in the inventory for multiple contracts.

See also:

`total_storage_at` `storage_at` `reserved_storage_at`

### `rollback(bookmark_id)`

Rolls back to the given bookmark ID

**Parameters** The bookmark ID returned from `bookmark(bookmark_id)`

**Remarks**

- You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the rollback will fail (return `False`)

See also:

`delete_bookmark` `rollback` `transaction` `temporary_transaction`

**Return type** **bool**
schedule (job, ignore_inventory_shortage=True, ignore_money_shortage=True, ignore_space_shortage=True, override=True)
Simulates scheduling the given job at its time and line optionally overriding whatever was already scheduled

Parameters
• job (Job) – Production job
• ignore_inventory_shortage – If true shortages in inputs will be ignored
• ignore_money_shortage – If true, shortage in money will be ignored
• ignore_space_shortage – If true, shortage in space will be ignored
• override – Whether the job should override any already registered job at its time-step

Return type bool

Returns Success/failure

sell (product, quantity, price, t, ignore_inventory_shortage=True, ignore_money_shortage=True)
sell a given quantity of a product for a given price at some time t

Parameters
• product (int) – Index/ID of the product to be sold
• quantity (int) – quantity to be sold
• price (int) – unit price
• t (int) – time
• ignore_money_shortage (bool) – If True, shortage in money will be ignored and the wallet can go negative
• ignore_inventory_shortage (bool) – Ignore shortage in the product which may lead to negative storag[product]

Return type bool

Returns Success or failure

Remarks:
• sell cannot ever have space shortage

See also:
buy

set_state (t, storage, wallet, loans, line_schedules)
Sets the current state at the given time-step. It implicitly causes a fix_before(t + 1)

Parameters
• t (int) – Time step to set the state at
• storage (<built-in function array>) – quantity of every product (array of integers of size n_products)
• wallet (float) – Cash in wallet
• loans (float) – Loans
• line_schedules (<built-in function array>) – Line schedules (array of process numbers/NO_PRODUCTION of size n_lines)

Return type None

storage_at (t)
Returns the storage of all products at time t
Parameters $t$ (int) – Time

Return type <built-in function array>

Returns An array of size $n\_products$ giving the quantity of each product in storage at time-step $t$.

See also:

storage_to wallet_at

storage_to ($t$)
Returns the storage of all products up to time $t$

Parameters $t$ (int) – Time

Return type <built-in function array>

Returns An array of size $n\_products \times t$ giving the quantity of each product in storage at every step up to $t$.

total_storage_at ($t$)
The total storage at a given time

Parameters $t$ (int) – time

Return type int

Returns an integer giving the total quantity of stored products in the inventory at timestep $t$

See also:

total_storage_to storage_at

total_storage_to ($t$)
The total storage up to a given time

Parameters $t$ (int) – time

Return type <built-in function array>

Returns an array of size $t$ giving the total quantity of stored products in the inventory up to timestep $t$

See also:

total_storage_at storage_to

transport_to ($product$, $quantity$, $t$, ignore_inventory_shortage=True, ignore_space_shortage=True)
Simulates transporting products to/from storage at time $t$

Parameters

- $product$ (int) – product ID (index)
- $quantity$ (int) – quantity to transport
- $t$ (int) – time
- $ignore\_inventory\_shortage$ (bool) – Ignore shortage in the $product$ which may lead to negative storage[$product$]
- $ignore\_space\_shortage$ (bool) – Ignore the limit on total storage which may lead to total_storage > max_storage

Return type bool

Returns Success or failure

wallet_at ($t$)
Returns the cash in wallet at a given timestep (given all simulated actions)

Parameters $t$ (int) –
Returns:

**Return type** float

wallet_to(t)
Returns the cash in wallet up to and including time t.

**Parameters** t (int) – Time

Returns:

**Return type** <built-in function array>

### DefaultGreedyManager

class negmas.apps.scml.DefaultGreedyManager(*args,
reserved_value=0.0,
negotiator_params=None,
optimism=0.0,
negotiator_type='negmas.sao.AspirationNegotiator',
n_retrials=5,
use_consumer=True,
reactive=True,
sign_only_guaranteedContracts=False,
riskiness=0.0,
maxInsurancePremium=inf,
**kwargs)

Bases: negmas.apps.scml.GreedyFactoryManager

#### Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td>name</td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

#### Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_expect_agreement(cfp, margin)</td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable</td>
</tr>
<tr>
<td>can_produce(cfp[...])</td>
<td>Whether or not we can produce the required item in time</td>
</tr>
<tr>
<td>can_secure_needs(schedule, step)</td>
<td>Finds if it is possible in principle to arrange these needs at the given time.</td>
</tr>
<tr>
<td>confirm_contract_execution(contract)</td>
<td>Called before executing any agreement</td>
</tr>
<tr>
<td>confirm_loan(loan, bankrupt_if_rejected)</td>
<td>called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached</td>
</tr>
<tr>
<td>confirm_partial_execution(contract, breaches)</td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 61 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create(*args, **kwargs)</code></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>from_config(config[, section, ...])</code></td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><code>init()</code></td>
<td>The initialization function called by the world directly.</td>
</tr>
<tr>
<td><code>notify(notifiable, notification)</code></td>
<td>Will be called whenever any agent goes bankrupt</td>
</tr>
<tr>
<td><code>on_agent_bankrupt(agent_id)</code></td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td><code>on_cash_transfer(amount, cause)</code></td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td><code>on_contract_breached(contract, breaches, ...)</code></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><code>on_contract_cancelled(contract)</code></td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td><code>on_contract_executed(contract)</code></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><code>on_contract_nullified(contract, ...)</code></td>
<td>Called whenever a contract the agent is involved in is nullified because another partner went bankrupt</td>
</tr>
<tr>
<td><code>on_contract_signed(contract)</code></td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td><code>on_event(event, sender)</code></td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td><code>on_inventory_change(product, quantity, cause)</code></td>
<td>Received whenever something moves in or out of the factory’s storage</td>
</tr>
<tr>
<td><code>on_neg_request_accepted(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><code>on_neg_request_accepted_(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td><code>on_neg_request_rejected(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><code>on_neg_request_rejected_(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td><code>on_negotiation_failure(partners, annotation, ...)</code></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><code>on_negotiation_failure_(partners, ...)</code></td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td><code>on_negotiation_success(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><code>on_negotiation_success_(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td><code>on_new_cfp(cfp)</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is published</td>
</tr>
<tr>
<td><code>on_new_report(report)</code></td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td><code>on_production_failure(failures)</code></td>
<td>Called with a list of <code>ProductionFailure</code> records on production failure.</td>
</tr>
<tr>
<td><code>on_production_success(reports)</code></td>
<td>Called with a list of <code>ProductionReport</code> records on production success.</td>
</tr>
<tr>
<td><code>on_remove_cfp(cfp)</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td><code>read_config(config[, section])</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 61 – continued from previous page

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>request_negotiation(cfp[, negotiator, ufun])</code></td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests.</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request(cfp, partner)</code></td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request(...)</code></td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request(...)</code></td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td><code>set_renegotiation_agenda(contract, breaches)</code></td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td><code>sign_contract(contract)</code></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called by the simulator at every simulation step</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called at every time-step.</td>
</tr>
<tr>
<td><code>total_utility([contracts])</code></td>
<td>Calculates the total utility for the agent of a collection of contracts</td>
</tr>
</tbody>
</table>

### Attributes Documentation

- **awi**
  Returns the Agent-World-Interface through which the agent does all of its actions in the world. A single exception is `request_negotiation` for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

  **Return type** *SCMLAWI*

- **id**
  The unique ID of this entity

- **name**
  A convenient name of the entity (intended primarily for printing/logging/debugging).

- **requested_negotiations**
  The negotiations currently requested by the agent.

  **Return type** *List[NegotiationRequestInfo]*

  **Returns** A list of negotiation request information objects (*NegotiationRequestInfo*)

- **running_negotiations**
  The negotiations currently requested by the agent.

  **Return type** *List[RunningNegotiationInfo]*

  **Returns** A list of negotiation information objects (*RunningNegotiationInfo*)

- **short_type_name**
  Returns a short name of the type of this entity

- **type_name**
  Returns the name of the type of this entity

- **unsigned_contracts**
  All contracts that are not yet signed.

  **Return type** *List[Contract]*

- **uuid**
  The unique ID of this entity
Methods Documentation

**can_expect_agreement** (*cfp, margin*)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

- **margin** (*int*)
- **cfp** (*CFP*)

Returns:

**can_produce** (*cfp, assume_no_further_negotiations=False*)
Whether or not we can produce the required item in time

Return type *bool*

**can_secure_needs** (*schedule, step*)
Finds if it is possible in principle to arrange these needs at the given time.

Parameters

- **schedule** (*ScheduleInfo*)
- **step** (*int*)

Returns:

**confirm_contract_execution** (*contract*)
Called before executing any agreement

Return type *bool*

**confirm_loan** (*loan, bankrupt_if_rejected*)
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

Return type *bool*

**confirm_partial_execution** (*contract, breaches*)
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters

- **contract** (*Contract*) – The contract that was breached
- **breaches** (*List[Breach]*) – A list of all the breaches committed.

Remarks:

- Will not be called if both partners committed breaches.

Return type *bool*

**classmethod create** (*args, **kwargs*)
Creates an object and returns a proxy to it.

**classmethod from_config** (*config, section=None, ignore_children=True, try_parsing_children=True, scope=None*)
Creates an object of this class given the configuration info

Parameters

- **config** (*Union[str, dict]*) – Either a file name or a dictionary
- **section** (*Optional[str]*) – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** (*bool*) – If true then children will be ignored and there will be a single return
try_parsing_children (bool) – If true the children will first be parsed as ConfigReader classes if they are not

types (e.g. int, str, float, Iterable[int|str|float] (simple)) –

scope – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are

any children that are to be parsed. (having) –

Returns An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

init()

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

init_()

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.

2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.

3. registers interest in all products that the agent can produce or consume in its factory.

4. finally it calls any custom initialization logic implemented in ‘init’()

See also:

init, step

notify (notifiable, notification)

on_agent_bankrupt (agent_id)

Will be called whenever any agent goes bankrupt

Parameters agent_id (str) – The ID of the agent that went bankrupt

Remarks:

• Agents can go bankrupt in two cases:

1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.

2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).

• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None
on_cash_transfer \((amount, cause)\)
Received whenever money is transferred to the factory or from it.

**Parameters**

- **amount** (*float*) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
- **cause** (*str*) – The cause of the change. Possibilities include:
  - contract: Contract execution
  - insurance: Received from insurance company
  - bankruptcy: Liquidated due to bankruptcy
  - transfer: Arrival of transferred money (when transfer delay in the system is > 0).

**Return type** None

on_contract_breached \((contract, breaches, resolution)\)
Called after complete processing of a contract that involved a breach.

**Parameters**

- **contract** (*Contract*) – The contract
- **breaches** (*List[Breach]*) – All breaches committed (even if they were resolved)
- **resolution** (*Optional[Contract]*) – The resolution contract if re-negotiation was successful. None if not.

**Return type** None

on_contract_cancelled \((contract, rejectors)\)
Called whenever at least a partner did not sign the contract

**Return type** None

on_contract_cancelled_ \((contract, rejectors)\)
Called whenever at least a partner did not sign the contract

**Return type** None

on_contract_executed \((contract)\)
Called after successful contract execution for which the agent is one of the partners.

**Return type** None

on_contract_nullified \((contract, bankrupt_partner, compensation)\)
Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

**Return type** None

on_contract_signed \((contract)\)
Called whenever a contract is signed by all partners

on_contract_signed_ \((contract)\)
Called whenever a contract is signed by all partners

**Return type** None

on_event \((event, sender)\)

on_inventory_change \((product, quantity, cause)\)
Received whenever something moves in or out of the factory’s storage

**Parameters**

- **product** (*int*) – Product index.
• **quantity** (*int*) – Negative value for products moving out and positive value for products moving in

• **cause** (*str*) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transport: Arrival of goods (when transportation delay in the system is \( > 0 \)).

**Return type** None

**on_neg_request_accepted** (*req_id, mechanism*)
Called when a requested negotiation is accepted

**on_neg_request_accepted_** (*req_id, mechanism*)
Called when a requested negotiation is accepted

**on_neg_request_rejected** (*req_id, by*)
Called when a requested negotiation is rejected

**Parameters**

• **req_id** (*str*) – The request ID passed to _request_negotiation

• **by** (*Optional[List[str]]*) – A list of agents that refused to participate or None if the failure was for another reason

**on_neg_request_rejected_** (*req_id, by*)
Called when a requested negotiation is rejected

**Parameters**

• **req_id** (*str*) – The request ID passed to _request_negotiation

• **by** (*Optional[List[str]]*) – A list of agents that refused to participate or None if the failure was for another reason

**on_negotiation_failure** (*partners, annotation, mechanism, state*)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_failure_** (*partners, annotation, mechanism, state*)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_success** (*contract, mechanism*)
Called whenever a negotiation ends with agreement

**on_negotiation_success_** (*contract, mechanism*)
Called whenever a negotiation ends with agreement

**Return type** None

**on_new_cfp** (*cfp*)
Called when a new CFP for a product for which the agent registered interest is published

**Return type** None

**on_new_report** (*report*)
Called whenever a financial report is published.

**Parameters**

• **report** (*FinancialReport*) – The financial report giving details of the standing of an agent at some time (see *FinancialReport*)

**Remarks:**
• Agents must opt-in to receive these calls by calling `receive_financial_reports` on their AWI.

`on_production_failure(failures)`
Called with a list of `ProductionFailure` records on production failure.

Return type: None

`on_production_success(reports)`
Called with a list of `ProductionReport` records on production success

Return type: None

`on_remove_cfp(cfp)`
Called when a new CFP for a product for which the agent registered interest is removed

Return type: None

`classmethod read_config(config, section=None)`
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• `config` ([str, dict]) – Either a file name or a dictionary

• `section` (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Return type: Dict[str, Any]

Returns: A dict ready to be parsed by from_config

Remarks:

`request_negotiation(cfp, negotiator=None, ufun=None)`
Requests a negotiation from the AWI while keeping track of available negotiation requests

Parameters

• `cfp` (CFP) –

• `negotiator` (Optional[Negotiator]) –

• `ufun` (Optional[UtilityFunction]) –

Return type: bool

Returns: Whether the negotiation request was successful indicating that the partner accepted the negotiation

`respond_to_negotiation_request(cfp, partner)`
Called when a prospective partner requests a negotiation to start

Return type: Optional[Negotiator]

`respond_to_negotiation_request_(initiator, partners, issues, annotation, mechanism, role, req_id)`
Called when a negotiation request is received

Return type: Optional[Negotiator]

`respond_to_renegotiation_request(contract, breaches, agenda)`
Called to respond to a renegotiation request

Parameters

• `agenda` (RenegotiationRequest) –

• `contract` (Contract) –

• `breaches` (List[Breach]) –

Returns:
Return type  Optional[Negotiator]

set_renegotiation_agenda(contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters

• contract (Contract) – The contract being breached
• breaches (List[Breach]) – All breaches on contract

Return type  Optional[RenegotiationRequest]

Returns  Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

sign_contract(contract)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

step()
Called by the simulator at every simulation step

step()
Called at every time-step. This function is called directly by the world.

total_utility(contracts=())
Calculates the total utility for the agent of a collection of contracts

Return type  float

ScheduleInfo
class negmas.apps.scml.ScheduleInfo(final_balance, valid=True, start=None, end=None, needs=<factory>, jobs=<factory>, failed_contracts=<factory>, ignored_contracts=<factory>)

Bases: object

Attributes Summary

end  The step after the last step in this simulation
start  The starting step of this schedule
valid  Is this a valid schedule?

Methods Summary

combine(other)  rtype  None

Attributes Documentation

end = None
The step after the last step in this simulation

start = None
The starting step of this schedule

valid = True
Is this a valid schedule?
Methods Documentation

`combine(other)`

Return type None

Scheduler

class negmas.apps.scml.Scheduler(manager_id, awi, max_insurance_premium=inf, horizon=None)

Bases: abc.ABC

Base class for all schedulers

Methods Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bookmark()</code></td>
<td>Sets a bookmark to the current location</td>
</tr>
<tr>
<td><code>delete_bookmark(bookmark_id)</code></td>
<td>Commits everything since the bookmark so it cannot be rolled back</td>
</tr>
<tr>
<td><code>find_schedule(contracts, start, end[, ...])</code></td>
<td>Schedules a set of contracts and returns either the search_for_schedule or None if infeasible</td>
</tr>
<tr>
<td><code>init(simulator, products, processes, ...)</code></td>
<td>Called by the FactoryManager after it is initialized</td>
</tr>
<tr>
<td><code>rollback(bookmark_id)</code></td>
<td>Rolls back to the given bookmark ID</td>
</tr>
<tr>
<td><code>schedule([contracts, ...])</code></td>
<td>Schedules a set of contracts and returns either the search_for_schedule or None if infeasible</td>
</tr>
</tbody>
</table>

Methods Documentation

`bookmark()`

Sets a bookmark to the current location

Return type int

Returns bookmark ID

`delete_bookmark(bookmark_id)`

Commits everything since the bookmark so it cannot be rolled back

Parameters The bookmark ID returned from bookmark(bookmark_id) –

Remarks:

• You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the deletion will fail (return False)

Return type bool

`find_schedule(contracts, start, end, assume_no_further_negotiations=False, ensure_storage_for=0, start_at=0)`

Schedules a set of contracts and returns either the search_for_schedule or None if infeasible

Parameters

• start(int) –
• end(int) –
• contracts(Collection[Contract]) –
• assume_no_further_negotiations –
• ensure_storage_for(int) –
• **start_at** *(int)* – The time at which to start scheduling. No jobs will be scheduled before this time.

**Return type** ScheduleInfo

**Returns** Schedule information (See ScheduleInfo for its contents).

*init* *(simulator, products, processes, profiles, producing)*

Called by the FactoryManager after it is initialized

*rollback* *(bookmark_id)*

Rolls back to the given bookmark ID

**Parameters** The bookmark ID returned from bookmark *(bookmark_id)* –

**Remarks:**

• You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the rollback will fail (return False)

**Return type** bool

*schedule* *(contracts=(), assume_no_further_negotiations=False, ensure_storage_for=0, start_at=0)*

Schedules a set of contracts and returns either the search_for_schedule or None if infeasible

**Parameters**

• it has scheduled before. If the state is given, it is taken as the initial state for scheduling *(whatever)* –

• **contracts** *(Collection[Contract])* – The contracts to be scheduled

• **assume_no_further_negotiations** – whether to assume that more negotiations can take place (to secure needs) *(production)* –

• **ensure_storage_for** *(int)* – A minimum time to ensure that products are available in storage before contract delivery

• **times** *(sell contracts)* –

• **start_at** *(int)* – The time at which to start scheduling. No jobs will be scheduled before this time.

**Return type** ScheduleInfo

**Returns** ScheduleInfo describing the schedule and any production needs and updates to be carried out.

**GreedyScheduler**

*class* negmas.apps.scml.GreedyScheduler *(manager_id,awi,max_insurance_premium=inf,horizon=None,add_catalog_prices=True,strategy='latest',profile_sorter='total-cost>time')*

**Bases:** negmas.apps.scml.Scheduler

Default scheduler used by the DefaultFactoryManager

**Methods Summary**

*bookmark()* Sets a bookmark to the current location

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delete_bookmark(bookmark_id)</code></td>
<td>Commits everything since the bookmark so it cannot be rolled back</td>
</tr>
<tr>
<td><code>find_schedule(contrats, start, end...)</code></td>
<td>Schedules a set of contracts and returns either the search_for_schedule or None if infeasible</td>
</tr>
<tr>
<td><code>init(simulator, products, processes,...)</code></td>
<td>Called by the FactoryManager after it is initialized</td>
</tr>
<tr>
<td><code>input_cost(info)</code></td>
<td></td>
</tr>
<tr>
<td><code>input_unit_cost(info)</code></td>
<td><strong>rtype</strong> float</td>
</tr>
<tr>
<td><code>production_cost(info)</code></td>
<td><strong>rtype</strong> float</td>
</tr>
<tr>
<td><code>production_unit_cost(info)</code></td>
<td><strong>rtype</strong> float</td>
</tr>
<tr>
<td><code>rollback(bookmark_id)</code></td>
<td>Rolls back to the given bookmark ID</td>
</tr>
<tr>
<td><code>schedule([contracts, ...])</code></td>
<td>Schedules a set of contracts and returns either the search_for_schedule or None if infeasible</td>
</tr>
<tr>
<td><code>schedule_contract(contract[...])</code></td>
<td>Schedules this contract if possible and returns information about the resulting schedule</td>
</tr>
<tr>
<td><code>schedule_contracts(contrats[, end, ...])</code></td>
<td>Schedules a set of contracts and returns the ScheduleInfo.</td>
</tr>
<tr>
<td><code>total_cost(info)</code></td>
<td><strong>rtype</strong> float</td>
</tr>
<tr>
<td><code>total_unit_cost(info)</code></td>
<td><strong>rtype</strong> float</td>
</tr>
<tr>
<td><code>unit_time(info)</code></td>
<td><strong>rtype</strong> float</td>
</tr>
</tbody>
</table>

### Methods Documentation

**bookmark()**

Sets a bookmark to the current location

**Return type** int

**Returns** bookmark ID

**delete_bookmark(bookmark_id)**

Commits everything since the bookmark so it cannot be rolled back

**Parameters** The bookmark ID returned from bookmark (bookmark_id) –

**Remarks:**

• You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the deletion will fail (return False)

**Return type** bool

**find_schedule(contrats, start, end, assume_no_further_negotiations=False, ensure_storage_for=0, start_at=0)**

Schedules a set of contracts and returns either the search_for_schedule or None if infeasible

**Parameters**

• **start(int)** –
NegMAS Documentation, Release 0.3.2

• end (int) –
• contracts (Collection[Contract]) –
• assume_no_further_negotiations –
• ensure_storage_for (int) –
• start_at (int) – The time at which to start scheduling. No jobs will be scheduled before this time.

**Returns** Schedule information (See ScheduleInfo for its contents).

**init** (*simulator, products, processes, profiles, producing*)

Called by the FactoryManager after it is initialized

**input_cost** (*info*)

**input_unit_cost** (*info*)

Return type *float*

**production_cost** (*info*)

Return type *float*

**production_unit_cost** (*info*)

Return type *float*

**rollback** (*bookmark_id*)

Rolls back to the given bookmark ID

Parameters The bookmark ID returned from bookmark (*bookmark_id*) –

Remarks:

• You can only rollback in the reverse order of bookmarks. If the bookmark ID given here is not the one at the top of the bookmarks stack, the rollback will fail (return False)

Return type *bool*

**schedule** (*contracts=(), assume_no_further_negotiations=False, ensure_storage_for=0, start_at=0*)

Schedules a set of contracts and returns either the search_for_schedule or None if infeasible

Parameters

• it has scheduled before. If the state is given, it is taken as the initial state for scheduling (whatever) –
• contracts (Collection[Contract]) – The contracts to be scheduled
• assume_no_further_negotiations – whether to assume that more negotiations can take place (to secure
• needs) (production) –
• ensure_storage_for (int) – A minimum time to ensure that products are available in storage before contract delivery
• times (sell contracts) –
• start_at (int) – The time at which to start scheduling. No jobs will be scheduled before this time.

Return type ScheduleInfo

**Returns** ScheduleInfo describing the schedule and any production needs and updates to be carried out.
**schedule_contract** *(contract, assume_no_further_negotiations=False, end=None, ensure_storage_for=0, start_at=0)*

Schedules this contract if possible and returns information about the resulting schedule.

**Parameters**

- **contract** *(Contract)* – The contract being scheduled.
- **assume_no_further_negotiations** – If true, no further negotiations will be assumed possible.
- **end** *(Optional[int])* – The scheduling horizon (None for the default).
- **ensure_storage_for** *(int)* – The number of steps all needs must be in storage before they are consumed in production.
- **start_at** *(int)* – No jobs will be scheduled before that time.

**Return type** *ScheduleInfo*

**Returns** Full schedule information including validity, line schedulers, production needs, etc (see SchedulerInfo).

**schedule_contracts** *(contracts, end=None, assume_no_further_negotiations=False, ensure_storage_for=0, start_at=0)*

Schedules a set of contracts and returns the ScheduleInfo.

**Parameters**

- **contracts** *(Collection[Contract])* – Contracts to schedule.
- **assume_no_further_negotiations** – If true, no further negotiations will be assumed to be possible.
- **end** *(Optional[int])* – The end of the simulation for the schedule (exclusive).
- **ensure_storage_for** *(int)* – Ensure that the outcome will be at the storage for at least this time.
- **start_at** *(int)* – The timestep at which to start scheduling.

**Return type** *ScheduleInfo*

**Returns** ScheduleInfo giving the schedule after these contracts is included. valid member can be used to check whether this is a valid contract.

**total_cost** *(info)*

**Return type** *float*

**total_unit_cost** *(info)*

**Return type** *float*

**unit_time** *(info)*

**Return type** *float*
class negmas.apps.scml.SCMLWorld(products, processes, factories, consumers, miners, factory_managers=None, n_steps=100, time_limit=5400, mechanisms=None, neg_n_steps=20, neg_time_limit=120, neg_step_time_limit=60, negotiation_speed=21, no_bank=False, minimum_balance=0, interest_rate=0.1, interest_max=0.3, installment_interest=0.2, interest_time_increment=0.02, balance_at_max_interest=None, loan_installments=1, no_insurance=False, premium=0.03, premium_time_increment=0.03, premium_breach_increment=0.001, max_allowed_breach_level=None, breach_processing=<BreachProcessing.VICTIM_THEN_PERPETRATOR: 1>, breach_penalty_society=0.1, breach_penalty_society_min=0.0, breach_penalty_victim=0.0, breach_move_max_product=True, initial_wallet_balances=None, money_resolution=0.5, default_signing_delay=0, transportation_delay=0, transfer_delay=0, start_negotiations_immediately=False, catalog_profit=0.15, avg_process_cost_is_public=True, catalog_prices_are_public=True, strip_annotations=True, financialReports_period=10, ignore_negotiated_penalties=False, prevent_cfp_tampering=False, default_price_for_products_without_one=1, compensation_fraction=0.5, log_to_file=True, log_to_screen=False, log_file_level=10, log_screen_level=40, log_file_name='log.txt', log_sums=False, log_negotiations=False, log_folder=None, save_mechanism_state_in_contract=False, compact=False, save_signed_contracts=True, save_cancelled_contracts=True, save_negotiations=True, save_resolved_breaches=True, save_unresolved_breaches=True, ignore_agent_exceptions=False, ignore_contract_execution_exceptions=False, name=None, **kwargs)

Bases: negmas.situated.World

The World class running a simulation of supply chain management.

Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agreement_rate</td>
<td>Fraction of negotiations ending in agreement and leading to signed contracts</td>
</tr>
<tr>
<td>breach_rate</td>
<td>Fraction of signed contracts that led to breaches</td>
</tr>
<tr>
<td>business_size</td>
<td>The total business size defined as the total money transferred within the system</td>
</tr>
<tr>
<td>cancellation_rate</td>
<td>Fraction of negotiations ending in agreement and leading to signed contracts</td>
</tr>
<tr>
<td><strong>Table 66 – continued from previous page</strong></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>cancelled_contracts</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>contract_execution_fraction</strong></td>
<td></td>
</tr>
<tr>
<td><strong>n_negotiation_rounds_failed</strong></td>
<td></td>
</tr>
<tr>
<td><strong>n_negotiation_rounds_successful</strong></td>
<td></td>
</tr>
<tr>
<td><strong>relative_time</strong></td>
<td></td>
</tr>
<tr>
<td><strong>remaining_steps</strong></td>
<td></td>
</tr>
<tr>
<td><strong>remaining_time</strong></td>
<td></td>
</tr>
<tr>
<td><strong>resolved_breaches</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>saved_breaches</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>saved_contracts</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>saved_negotiations</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>signed_contracts</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>stats</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>time</strong></td>
<td></td>
</tr>
<tr>
<td><strong>unresolved_breaches</strong></td>
<td><strong>rtype</strong></td>
</tr>
<tr>
<td><strong>winners</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Methods Summary**

| **announce(event)** | | Raises an event and informs all event sinks that are registered for notifications on this event type |
| **append_stats()** | | |
| **assign_managers([factory_managers, params])** | | Assigns existing factories to new factory managers created from the given types and parameters or manager objects. |
| **buy_insurance(contract, agent)** | | Buys insurance for the contract by the premium calculated by the insurance company. |
| **chain_world([n_intermediate_levels, ...])** | | Creates a very small world in which only one raw material and one final product. |
| **evaluate_insurance(contract, agent[, t])** | | Can be called to evaluate the premium for insuring the given contract against breaches committed by others |
| **execute(action, agent[, callback])** | | Executes the given action by the given agent |

Continued on next page
Table 67 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_config</code></td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td><code>get_private_state</code></td>
<td>Reads the private state of the given agent</td>
</tr>
<tr>
<td><code>join</code></td>
<td>Add an agent to the world.</td>
</tr>
<tr>
<td><code>logdebug</code></td>
<td>logs debug-level information</td>
</tr>
<tr>
<td><code>logerror</code></td>
<td>logs error-level information</td>
</tr>
<tr>
<td><code>loginfo</code></td>
<td>logs info-level information</td>
</tr>
<tr>
<td><code>logwarning</code></td>
<td>logs warning-level information</td>
</tr>
<tr>
<td><code>make_bankrupt</code></td>
<td>Marks the agent as bankrupt</td>
</tr>
<tr>
<td><code>nullify_contract</code></td>
<td>Called whenever a concluded contract is not signed (cancelled)</td>
</tr>
<tr>
<td><code>on_contract_cancelled</code></td>
<td>Called to add a contract to the existing set of contract after it is signed</td>
</tr>
<tr>
<td><code>on_contract_concluded</code></td>
<td>Called to add a contract to the existing set of contract after it is signed</td>
</tr>
<tr>
<td><code>on_contract_signed</code></td>
<td>Called whenever an event is raised for which the World is registered as a listener</td>
</tr>
<tr>
<td><code>random</code></td>
<td>Creates a random SCML scenario with adjustable parameters.</td>
</tr>
<tr>
<td><code>random_small</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td><code>receive_financial_reports</code></td>
<td>Registers interest/disinterest in receiving financial reports</td>
</tr>
<tr>
<td><code>register</code></td>
<td>Registers an entity in the world so it can be looked up by name.</td>
</tr>
<tr>
<td><code>register_interest</code></td>
<td></td>
</tr>
<tr>
<td><code>register_listener</code></td>
<td></td>
</tr>
<tr>
<td><code>register_stats_monitor</code></td>
<td></td>
</tr>
<tr>
<td><code>register_world_monitor</code></td>
<td></td>
</tr>
<tr>
<td><code>request_negotiation_about</code></td>
<td>Requests to start a negotiation with some other agents</td>
</tr>
<tr>
<td><code>run</code></td>
<td>Runs the simulation until it ends</td>
</tr>
<tr>
<td><code>run_negotiation</code></td>
<td>Requests to start a negotiation with some other agents</td>
</tr>
<tr>
<td><code>save_config</code></td>
<td>Saves the config of the world as a yaml file</td>
</tr>
<tr>
<td><code>set_bulletin_board</code></td>
<td></td>
</tr>
<tr>
<td><code>set_consumers</code></td>
<td></td>
</tr>
<tr>
<td><code>set_factory_managers</code></td>
<td></td>
</tr>
<tr>
<td><code>set_miners</code></td>
<td></td>
</tr>
<tr>
<td><code>set_processes</code></td>
<td></td>
</tr>
<tr>
<td><code>set_products</code></td>
<td></td>
</tr>
<tr>
<td><code>step</code></td>
<td>A single simulation step</td>
</tr>
<tr>
<td><code>unregister_interest</code></td>
<td></td>
</tr>
<tr>
<td><code>unregister_listener</code></td>
<td></td>
</tr>
<tr>
<td><code>unregister_stats_monitor</code></td>
<td></td>
</tr>
<tr>
<td><code>unregister_world_monitor</code></td>
<td></td>
</tr>
</tbody>
</table>
Attributes Documentation

**agreement_rate**
Fraction of negotiations ending in agreement and leading to signed contracts

**Return type** float

**breach_rate**
Fraction of signed contracts that led to breaches

**Return type** float

**business_size**
The total business size defined as the total money transferred within the system

**Return type** float

**cancellation_rate**
Fraction of negotiations ending in agreement and leading to signed contracts

**Return type** float

**cancelled_contracts**

**Return type** List[Dict[str, Any]]

**contract_execution_fraction**
Fraction of signed contracts successfully executed

**Return type** float

**n_negotiation_rounds_failed**
Average number of rounds in a successful negotiation

**Return type** float

**n_negotiation_rounds_successful**
Average number of rounds in a successful negotiation

**Return type** float

**relative_time**
Returns a number between 0 and 1 indicating elapsed relative time or steps.

**Return type** float

**remaining_steps**
Returns the remaining number of steps until the end of the mechanism run. None if unlimited

**Return type** Optional[int]

**remaining_time**
Returns remaining time in seconds. None if no time limit is given.

**Return type** Optional[float]

**resolved_breaches**

**Return type** List[Dict[str, Any]]

**saved_breaches**

**Return type** List[Dict[str, Any]]

**saved_contracts**

**Return type** List[Dict[str, Any]]

**saved_negotiations**

**Return type** List[Dict[str, Any]]

**signed_contracts**
Return type: \texttt{List[Dict[str, \text{Any}]]} \\
\textbf{stats} \\
Return type: \texttt{Dict[str, \text{Any}]} \\
\textbf{time} \\
Elapsed time since mechanism started in seconds. None if the mechanism did not start running \\
Return type: \texttt{Optional[float]} \\
\textbf{unresolved_breaches} \\
Return type: \texttt{List[Dict[str, \text{Any}]]} \\
\textbf{winners} \\
The winners of this world (factory managers with maximum wallet balance

\textbf{Methods Documentation}

\textbf{announce} \((\text{event})\) \\
Raises an event and informs all event sinks that are registerd for notifications on this event type \\
\textbf{append_stats} () \\
\textbf{assign_managers} \((\text{factory_managers}=\text{typing.Iterable[typing.Union[\text{str, typing.Type[negmas.apps.scml.factory_managers.FactoryManager}, negmas.apps.scml.factory_managers.FactoryManager]]]}, \text{params}=\text{None})\) \\
Assigns existing factories to new factory managers created from the given types and parameters or manager objects. \\
\textbf{Parameters} \\
\begin{itemize} \\
\item \textbf{factory_managers} – An iterable of \texttt{FactoryManager} objects type names or \texttt{FactoryManager} types to assign to \\
\item \textbf{params} (\texttt{Optional[Iterable[Dict[str, \text{Any}]]]}) – parameters of the newly created managers \\
\end{itemize} \\
\textbf{Remarks}: \\
\begin{itemize} \\
\item factories are assigned in the same order they exist in the local \texttt{factories} attribute cycling through the input managers or types/params \\
\item If a \texttt{FactoryManager} object is given instead of a type or a string in the \texttt{factory_managers} collection, and the number of \texttt{factory_managers} is less than the number of factories in the world causing this object to cycle for more than one factory, it is assigned to the first such factory but then deep copies of it with new ids and names are assigned to the rest of the factories. That ensures that each manager has exactly one factory and that all factories are assigned exactly one unique manager. \\
\end{itemize} \\
Return type: \texttt{None} \\
\textbf{buy_insurance} \((\text{contract}, \text{agent})\) \\
Buys insurance for the contract by the premium calculated by the insurance company. \\
\textbf{Remarks}: The agent can call \texttt{evaluate_insurance} to find the premium that will be used. \\
Return type: \texttt{bool}
class method chain_world (n_intermediate_levels=0, n_miners=5, n_factories_per_level=5, n_consumers=5, n_steps=100, n_lines_per_factory=10, n_max_assignable_factories=None, log_file_name=None, agent_names_reveal_type=False, negotiator_type='negmas.sao.AspirationNegotiator', miner_type=<class 'negmas.apps.scml.miners.ReactiveMiner'>, consumer_type=<class 'negmas.apps.scml.consumers.ScheduleDrivenConsumer'>, max_storage=922372036854775807, default_manager_params=None, miner_kwargs=None, consumption=(0, 5), consumer_kwargs=None, negotiation_speed=None, manager_types=(<class 'negmas.apps.scml.factory_managers.GreedyFactoryManager'>, ), manager_params=None, n_default_per_level=0, default_factory_manager_type=<class 'negmas.apps.scml.factory_managers.GreedyFactoryManager'>, randomize=True, initial_wallet_balances=1000, process_cost=(1.0, 5.0), process_time=1, interest_rate=inf, interest_max=inf, shared_profile_per_factory=False, **kwargs)

Creates a very small world in which only one raw material and one final product. The production graph is a series with n_intermediate_levels intermediate levels between the single raw material and single final product.

Parameters

- n_max_assignable_factories – The maximum number of factories assigned to managers other than the default
- randomize (bool) – If true, the factory assignment is randomized
- n_default_per_level (int) – The number of GreedyFactoryManager objects guaranteed at every level
- default_factory_manager_type (Type[FactoryManager]) – The FactoryManager type to use as the base for default_factory_managers. You can specify how many of this type exist at every level by specifying n_default_per_level. If n_default_per_level is zero, this parameter has no effect.
- manager_types (Sequence[Type[FactoryManager]]) – A sequence of factory manager types to control the factories.
- manager_params (Optional[Sequence[Dict[str, Any]]]) – An optional sequence of dictionaries giving the parameters to pass to manager_types.
- consumer_type (Union[str, Type[Consumer]]) – Consumer type to use for all consumers
- miner_type (Union[str, Type[Miner]]) – Miner type to use for all miners
- consumption (Union[int, Tuple[int, int]]) – Consumption schedule
- n_intermediate_levels – The number of intermediate products
- n_miners – number of miners of the single raw material
- n_factories_per_level – number of factories at every production level
- n_consumers (Union[int, Tuple[int, int], List[int]]) – number of consumers of the final product
- n_steps – number of simulation steps
- n_lines_per_factory – number of lines in each factory
- process_cost (Union[float, Tuple[float, float]]) – The range of process costs. A uniform distribution will be used.
• **process_time** (Union[int, Tuple[int, int]]) – The range of process times. A uniform distribution will be used.

• **log_file_name** (Optional[str]) – File name to store the logs.

• **agent_names_reveal_type** (bool) – If true, agent names will start with a snake_case version of their type name.

• **negotiator_type** (str) – The negotiation factory used to create all negotiators.

• **max_storage** (int) – Maximum storage capacity for all factory managers. If None then it is unlimited.

• **default_manager_params** (Optional[Dict[str, Any]]) – Keyword arguments to be used for constructing factory managers.

• **consumer_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to be used for constructing consumers.

• **miner_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to be used for constructing miners.

• **negotiation_speed** (Optional[int]) – The number of negotiation steps per simulation step. None means infinite.

• **interest_max** – Maximum interest rate.

• **interest_rate** – Minimum interest rate.

• **initial_wallet_balances** – Initial wallet balances for all factories.

• **shared_profile_per_factory** – If true, all lines in the same factory will have the same profile costs.

• **kwargs** – Any other parameters are just passed to the world constructor.

**Returns** SCMLWorld ready to run.

**Remarks:**

• Every production level \( n \) has one process only that takes \( n \) steps to complete.

**evaluate_insurance** (contract, agent, \( t=\text{None} \))

Can be called to evaluate the premium for insuring the given contract against breaches committed by others.

**Parameters**

• **agent** (:class:`SCMLAgent`) – The agent buying the contract.

• **contract** (:class:`Contract`) – Hypothetical contract.

• **\( t \)** (Optional[int]) – Time at which the policy is to be bought. If None, it means current step.

**Return type** Optional[float]

**execute** (action, agent, callback=\text{None})

Executes the given action by the given agent.

**Return type** bool

**classmethod from_config** (config, \( \text{section=\text{None}}, \ \text{ignore_children=\text{True}}, \ \text{try_parsing_children=\text{True}}, \ \text{scope=\text{None}} \))

Creates an object of this class given the configuration info.

**Parameters**

• **config** (Union[str, dict]) – Either a file name or a dictionary.

• **\text{section}** (Optional[str]) – A section in the file or a key in the dictionary to use for loading params.
• **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return

• **try_parsing_children** *(bool)* – If true the children will first be parsed as *ConfigReader* classes if they are not

• **types** *(e.g. int, str, float, Iterable[int|str|float] (simple) –

• **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are

• **any children that are to be parsed.** *(having)* –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**get_private_state** *(agent)*

Reads the private state of the given agent

**Return type** *FactoryState*

**join** *(x, simulation_priority=0)*

Add an agent to the world.

**Parameters**

• **x** *(Agent)* – The agent to be registered

• **simulation_priority** *(int)* – The simulation priority. Entities with lower priorities will be stepped first during

**Returns:**

**logdebug** *(s)*

logs debug-level information

**Parameters** *s (str)* – The string to log

**Return type** *None*

**logerror** *(s)*

logs error-level information

**Parameters** *s (str)* – The string to log

**Return type** *None*

**loginfo** *(s)*

logs info-level information

**Parameters** *s (str)* – The string to log

**Return type** *None*

**logwarning** *(s)*

logs warning-level information

**Parameters** *s (str)* – The string to log
Return type None

make_bankrupt (agent, amount, beneficiary, contract)
Mark the agent as bankrupt

Return type None

nullify_contract (contract)

on_contract_cancelled (contract)
Called whenever a concluded contract is not signed (cancelled)

Parameters contract – The contract to add

Remarks:
• By default this function just adds the contract to the set of contracts maintained by the world.
• You should ALWAYS call this function when overriding it.

on_contract_concluded (contract, to_be_signed_at)
Called to add a contract to the existing set of contract after it is signed

Parameters
• contract (Contract) – The contract to add
• to_be_signed_at (int) – The timestep at which the contract is to be signed

Remarks:
• By default this function just adds the contract to the set of contracts maintained by the world.
• You should ALWAYS call this function when overriding it.

Return type None

on_contract_signed (contract)
Called to add a contract to the existing set of contract after it is signed

Parameters contract (Contract) – The contract to add

Remarks:
• By default this function just adds the contract to the set of contracts maintained by the world.
• You should ALWAYS call this function when overriding it.

on_event (event, sender)
Called whenever an event is raised for which the World is registered as a listener

Parameters
• event (Event) – The event
• sender (EventSource) – The sender

Return type None

Returns None
classmethod random(n_raw_materials=(5, 10), raw_material_price=(1.0, 30.0),
    n_final_products=(3, 5), n_production_levels=(3, 5),
    n_products_per_level=(6, 10), n_inputs_per_process=(2, 5),
    bias_toward_last_level_products=0.0,
    quantity_per_input=(1, 10), input_step=0.0, quantity_per_output=(1, 1),
    output_step=1.0, process_relative_cost=(0.05, 0.4),
    n_outputs_per_process=(1, 1), n_lines=(3, 5), lines_are_similar=False,
    n_processes_per_line=None, cost_for_line=(5.0, 50.0),
    n_production_steps=(2, 10), max_storage=2000, n_factories=20,
    n_consumers=5, n_products_per_consumer=None,
    n_miners=5, n_products_per_miner=None,
    factory_manager_types=<class 'negmas.apps.scml.factory_managers.GreedyFactoryManager'>,
    consumer_types=<class 'negmas.apps.scml.consumers.ScheduleDrivenConsumer'>,
    miner_types=<class 'negmas.apps.scml.miners.ReactiveMiner'>,
    negotiator_type='negmas.sao.AspirationNegotiator',
    initial_wallet_balance=1000, factory_kwargs=None, miner_kwargs=None,
    consumer_kwargs=None, **kwargs)

Creates a random SCML scenario with adjustable parameters.

Parameters

- **n_raw_materials** (Union[int, Tuple[int, int]]): Number of raw materials. Can be a value or a range.
- **raw_material_price** (Union[Union[int, Tuple[int, int]]]: Catalog prices for raw materials. Can be a value or a range.
- **n_final_products** (Union[int, Tuple[int, int]]): Number of final products. Can be a value or a range.
- **n_production_levels** (Union[int, Tuple[int, int]]): How deep is the production graph (number of intermediate products). Can be a value or a range.
- **n_products_per_level** (Union[int, Tuple[int, int]]): How many intermediate products per intermediate level. Can be a value or a range.
- **n_processes_per_level** (Union[int, Tuple[int, int]]): Number of processes in intermediate levels. Can be a value or a range.
- **n_inputs_per_process** (Union[int, Tuple[int, int]]): Number of inputs per process. Can be a value or a range.
- **bias_toward_last_level_products** (float): How biased are production processes toward using products from the last level.
- **quantity_per_input** (Union[int, Tuple[int, int]]): How many items are needed for each input to a process. Can be a value or a range.
- **input_step** (Union[Union[int, Tuple[int, int]]]: When are inputs consumed during the production process. Can be a value or a range. Default 0.
- **quantity_per_output** (Union[int, Tuple[int, int]]): How many items are produced per output. Can be a value or a range.
- **output_step** (Union[Union[int, Tuple[int, int]]]: When are outputs created during the production process. Can be a value or a range. Default 1.
- **process_relative_cost** (Union[Union[int, Tuple[int, int]]]: Intrinsic relative cost of processes. [Outputs will be produced.
- **a cost of sum(at)**
• `n_outputs_per_process (Union[int, Tuple[int, int]])` – Number of outputs per process. Can be a value or a range.

• `n_lines (Union[int, Tuple[int, int]])` – Number of lines per factory. Can be a value or a range.

• `lines_are_similar (bool)` – If true then all lines of the same factory will have the same production processes.

• `n_processes_per_line (Union[int, Tuple[int, int], None])` – Number of processes that can be run on each line per factory. Can be a value or a range.

• `cost_for_line (Union[float, Tuple[float, float]])` – Cost for running a process on a line. Can be a value or a range.

• `n_production_steps (Union[int, Tuple[int, int]])` – Number of production steps per line. Can be a value or a range.

• `max_storage (Union[int, Tuple[int, int]])` – Maximum storage per factory. Can be a value or a range.

• `n_factories (Union[int, Tuple[int, int]])` – Number of factories. Can be a value or a range.

• `n_consumers (Union[int, Tuple[int, int]])` – Number of consumers. Can be a value or a range.

• `n_products_per_consumer (Union[int, Tuple[int, int], None])` – Number of products per miner. If None then all final products will be assigned to every consumer. Can be a value or a range.

• `n_miners (Union[int, Tuple[int, int]])` – Number of miners. Can be a value or a range.

• `n_products_per_miner (Union[int, Tuple[int, int], None])` – Number of products per miner. If None then all raw materials will be assigned to every miner. Can be a value or a range.

• `factory_manager_types (Union[Type[FactoryManager], List[Type[FactoryManager]]])` – A callable for creating factory managers for the factories.

• `consumer_types (Union[Type[Consumer], List[Type[Consumer]]])` – A callable for creating Consumer objects.

• `miner_types (Union[Type[Miner], List[Type[Miner]]])` – A callable for creating Miner objects.

• `negotiator_type` – A string that can be evaluated to a negotiator.

• `initial_wallet_balance (Union[float, Tuple[float, float]])` – The initial balance of all wallets.

• `factory_kwargs (Optional[Dict[str, Any]])` – Keyword arguments to be used for constructing factory managers.

• `consumer_kwargs (Optional[Dict[str, Any]])` – Keyword arguments to be used for constructing consumers.

• `miner_kwargs (Optional[Dict[str, Any]])` – Keyword arguments to be used for constructing miners.

**kwargs –

Returns `SCMLWorld` The random world generated.

Remarks:
• Most parameters accept either a single value or a 2-valued tuple. In the later case, it will sample a value within the range specified by the tuple (low, high) inclusive. For example the number of lines (n_lines) follows this pattern

```python
classmethod random_small(n_production_levels=1, n_factories=10, factory_kwargs=None, miner_kwargs=None, consumer_kwargs=None, **kwargs)
```

```python
classmethod read_config(config, section=None)
```

Reads the configuration from a file or a dict and prepares it for parsing

**Parameters**

- `config` ([str, dict]) – Either a file name or a dictionary
- `section` (Optional[None]) – A section in the file or a key in the dictionary to use for loading params

**Return type** Dict[str, Any]

**Returns** A dict ready to be parsed by from_config

Remarks:

```python
receive_financial_reports(agent, receive, agents)
```

Registers interest/disinterest in receiving financial reports

```python
register(x, simulation_priority=0)
```

Registers an entity in the world so it can be looked up by name. Should not be called directly

**Parameters**

- `x` (Entity) – The entity to be registered
- `simulation_priority` (int) – The simulation priority. Entities with lower priorities will be stepped first during

**Returns:**

```python
register_interest(agent, products)
```

**Return type** None

```python
register_listener(event_type, listener)
```

```python
register_stats_monitor(m)
```

```python
register_world_monitor(m)
```

```python
request_negotiation_about(req_id, caller, issues, partners, roles=None, annotation=None, mechanism_name=None, mechanism_params=None)
```

Requests to start a negotiation with some other agents

**Parameters**

- `req_id` (str) – An ID For the request that is unique to the caller
- `caller` (Agent) – The agent requesting the negotiation
- `partners` (List[Agent]) – The list of partners that the agent wants to negotiate with. Roles will be determined by these agents.
- `issues` (List[Issue]) – Negotiation issues
- `annotation` (Optional[Dict[str, Any]]) – Extra information to be passed to the partners when asking them to join the negotiation
- `partners` – A list of partners to participate in the negotiation
- `roles` (Optional[List[str]]) – The roles of different partners. If None then each role for each partner will be None
• mechanism_name (Optional[\text{str}]) – Name of the mechanism to use. It must be one of the mechanism_names that are supported by the

• or None which means that the World should select the mechanism. If None, then roles and my_role (\text{World}) –

• also be None (\text{must}) –

• mechanism_params (Optional[\text{Dict[\text{str, Any}]}) – A dict of parameters used to initialize the mechanism object

Returns None. The caller will be informed by a callback function on_neg_request_accepted or on_neg_request_rejected about the status of the negotiation.

run()
Runs the simulation until it ends

run_negotiation (caller, issues, partners, roles=None, annotation=None, mechanism_name=None, mechanism_params=None)
Requests to start a negotiation with some other agents

Parameters

• caller (\text{Agent}) – The agent requesting the negotiation

• partners (\text{Collection[Agent]}) – The list of partners that the agent wants to negotiate with. Roles will be determined by these agents.

• issues (\text{Collection[Issue]}) – Negotiation issues

• annotation (Optional[\text{Dict[\text{str, Any}]}) – Extra information to be passed to the partners when asking them to join the negotiation

• partners – A list of partners to participate in the negotiation

• roles (Optional[\text{Collection[\text{str}]}) – The roles of different partners. If None then each role for each partner will be None

• mechanism_name (Optional[\text{str}]) – Name of the mechanism to use. It must be one of the mechanism_names that are supported by the

• or None which means that the World should select the mechanism. If None, then roles and my_role (\text{World}) –

• also be None (\text{must}) –

• mechanism_params (Optional[\text{Dict[\text{str, Any}]}) – A dict of parameters used to initialize the mechanism object

Returns The agreed upon contract if negotiation was successful otherwise, None.

Return type \text{Contract}

save_config (file_name)
Saves the config of the world as a yaml file

Parameters

file_name (\text{str}) – Name of file to save the config to

Returns:

Return type None

set_bulletin_board (bulletin_board)
set_consumers (consumers)
set_factory_managers (factory_managers)
set_miners (miners)
set_processes (processes)
**set_products** *(products)*

**step()**
A single simulation step

**Return type** `bool`

**unregister_interest** *(agent, products)*

**Return type** `None`

**unregister_stats_monitor** *(m)*

**unregister_world_monitor** *(m)*

**Consumer**

class `negmas.apps.scml.Consumer` *(name=None)*

Bases: `negmas.apps.scml.SCMLAgent, abc.ABC`

Base class of all consumer classes

**Attributes Summary**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>awi</code></td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>The unique ID of this entity</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
<tr>
<td><code>requested_negotiations</code></td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td><code>running_negotiations</code></td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td><code>short_type_name</code></td>
<td>Returns a short name of the type of this entity</td>
</tr>
<tr>
<td><code>type_name</code></td>
<td>Returns the name of the type of this entity</td>
</tr>
<tr>
<td><code>unsigned_contracts</code></td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td><code>uuid</code></td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

**Methods Summary**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>can_expect_agreement</code> <em>(cfp, margin)</em></td>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable</td>
</tr>
<tr>
<td><code>confirm_contract_execution</code> <em>(contract)</em></td>
<td>Called before executing any agreement</td>
</tr>
<tr>
<td><code>confirm_loan</code> <em>(loan, bankrupt_if_rejected)</em></td>
<td>called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached</td>
</tr>
<tr>
<td><code>confirm_partial_execution</code> <em>(contract, breaches)</em></td>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
<tr>
<td><code>create</code> *(args, *<em>kwargs)</em></td>
<td>Creates an object and returns a proxy to it.</td>
</tr>
<tr>
<td><code>from_config</code> <em>(config[, section, ...])</em></td>
<td>Creates an object of this class given the configuration info</td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
<tr>
<td><code>init_()</code></td>
<td>The initialization function called by the world directly.</td>
</tr>
<tr>
<td><code>notify</code> <em>(notifiable, notification)</em></td>
<td>Will be called whenever any agent goes bankrupt</td>
</tr>
</tbody>
</table>

Continued on next page
Table 69 – continued from previous page

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_cash_transfer(amount, cause)</td>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
<tr>
<td>on_contract_breached(contract, breaches, ...)</td>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
<tr>
<td>on_contract_cancelled(contract, rejectors)</td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td>on_contract_cancelled_(contract, rejectors)</td>
<td>Called whenever at least a partner did not sign the contract</td>
</tr>
<tr>
<td>on_contract_executed(contract)</td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td>on_contract_nullified(contract, ...)</td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt</td>
</tr>
<tr>
<td>on_contract_signed(contract)</td>
<td>Called whenever a contract is signed by all partners</td>
</tr>
<tr>
<td>on_event(event, sender)</td>
<td></td>
</tr>
<tr>
<td>on_inventory_change(product, quantity, cause)</td>
<td>Received whenever something moves in or out of the factory’s storage</td>
</tr>
<tr>
<td>on_neg_request_accepted(req_id, mechanism)</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>on_neg_request_accepted_(req_id, mechanism)</td>
<td>Called when a requested negotiation is accepted</td>
</tr>
<tr>
<td>on_neg_request_rejected(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_neg_request_rejected_(req_id, by)</td>
<td>Called when a requested negotiation is rejected</td>
</tr>
<tr>
<td>on_negotiation_failure(partners, annotation, ...)</td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td>on_negotiation_failure_(partners, ...)</td>
<td>Called whenever a negotiation ends without agreement</td>
</tr>
<tr>
<td>on_negotiation_success(contract, mechanism)</td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td>on_negotiation_success_(contract, mechanism)</td>
<td>Called whenever a negotiation ends with agreement</td>
</tr>
<tr>
<td>on_new_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is published</td>
</tr>
<tr>
<td>on_new_report(report)</td>
<td>Called whenever a financial report is published</td>
</tr>
<tr>
<td>on_remove_cfp(cfp)</td>
<td>Called when a new CFP for a product for which the agent registered interest is removed</td>
</tr>
<tr>
<td>read_config(config[, section])</td>
<td>Reads the configuration from a file or a dict and prepares it for parsing</td>
</tr>
<tr>
<td>request_negotiation(cfp, negotiator, uf)</td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests</td>
</tr>
<tr>
<td>respond_to_negotiation_request(cfp, partner)</td>
<td>Called when a prospective partner requests a negotiation to start</td>
</tr>
<tr>
<td>respond_to_negotiation_request_(initiator, ...)</td>
<td>Called when a negotiation request is received</td>
</tr>
<tr>
<td>respond_to_renegotiation_request_(initiator, ...)</td>
<td>Called to respond to a renegotiation request</td>
</tr>
<tr>
<td>set_renegotiation_agenda(contract, breaches)</td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails</td>
</tr>
<tr>
<td>sign_contract(contract)</td>
<td>Called after the signing delay from contract conclusion to sign the contract</td>
</tr>
<tr>
<td>step()</td>
<td>Called by the simulator at every simulation step</td>
</tr>
</tbody>
</table>

Continued on next page
Attributes Documentation

awi
Returns the Agent-World-Interface through which the agent does all of its actions in the world.

A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

Return type  SCMLAWI

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations
The negotiations currently requested by the agent.

Return type  List[NegotiationRequestInfo]

Returns  A list of negotiation request information objects (NegotiationRequestInfo)

running_negotiations
The negotiations currently requested by the agent.

Return type  List[RunningNegotiationInfo]

Returns  A list of negotiation information objects (RunningNegotiationInfo)

short_type_name
Returns a short name of the type of this entity

type_name
Returns the name of the type of this entity

unsigned_contracts
All contracts that are not yet signed.

Return type  List[Contract]

uuid
The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

• margin (int) –
• cfp (CFP) –

Returns:

confirm_contract_execution (contract)
Called before executing any agreement

Return type  bool

confirm_loan (loan, bankrupt_if_rejected)
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached
Return type  
**bool**

**confirm_partial_execution** *(contract, breaches)*
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

**Parameters**

- **contract** *(Contract)* – The contract that was breached
- **breaches** *(List[Breach])* – A list of all the breaches committed.

**Remarks:**
- Will not be called if both partners committed breaches.

Return type  
**bool**

**classmethod create** *(\*args, **kwargs)*
Creates an object and returns a proxy to it.

**classmethod from_config** *(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)*
Creates an object of this class given the configuration info

**Parameters**

- **config** *(Union[str, dict])* – Either a file name or a dictionary
- **section** *(Optional[str])* – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** *(bool)* – If true then children will be ignored and there will be a single return
- **try_parsing_children** *(bool)* – If true the children will first be parsed as ConfigReader classes if they are not
- **types** *(e.g. int, str, float, Iterable[int|str|float]) (simple)* –
- **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
- **any children that are to be parsed. (having)** –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

**Remarks:**

- This function will return an object of its class after passing the key-value pairs found in the config to the init function.
- Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**init()**
Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**init_()**
The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.
2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
3. registers interest in all products that the agent can produce or consume in its factory.
4. finally it calls any custom initialization logic implemented in ‘init’()

See also:
init, step

notify (notifiable, notification)

on_agent_bankrupt (agent_id)
Will be called whenever any agent goes bankrupt

Parameters agent_id (str) – The ID of the agent that went bankrupt

Remarks:
• Agents can go bankrupt in two cases:
  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).
• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)
Received whenever money is transferred to the factory or from it.

Parameters
  • amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
  • cause (str) – The cause of the change. Possibilities include:
    – contract: Contract execution
    – insurance: Received from insurance company
    – bankruptcy: Liquidated due to bankruptcy
    – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breach (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters
  • contract (Contract) – The contract
  • breaches (List[Breach]) – All breaches committed (even if they were resolved)
  • resolution (Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

on_contract_cancelled (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None
on_contract_cancelled (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

on_contract_executed (contract)
Called after successful contract execution for which the agent is one of the partners.

Return type None

on_contract_nullified (contract, bankruptPartner, compensation)
Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.

Return type None

on_contract_signed (contract)
Called whenever a contract is signed by all partners

Return type None

on_contract_signed_ (contract)
Called whenever a contract is signed by all partners

Return type None

on_event (event, sender)
on_inventory_change (product, quantity, cause)
Received whenever something moves in or out of the factory’s storage

Parameters

• product (int) – Product index.
• quantity (int) – Negative value for products moving out and positive value for products moving in
• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transport: Arrival of goods (when transportation delay in the system is > 0).

Return type None

on_neg_request_accepted (req_id, mechanism)
Called when a requested negotiation is accepted

on_neg_request_accepted_ (req_id, mechanism)
Called when a requested negotiation is accepted

on_neg_request_rejected (req_id, by)
Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_neg_request_rejected_ (req_id, by)
Called when a requested negotiation is rejected

Parameters

• req_id (str) – The request ID passed to _request_negotiation
• by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

**on_negotiation_failure** (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_failure_** (partners, annotation, mechanism, state)
Called whenever a negotiation ends without agreement

**Return type** None

**on_negotiation_success** (contract, mechanism)
Called whenever a negotiation ends with agreement

**Return type** None

**on_negotiation_success_** (contract, mechanism)
Called whenever a negotiation ends with agreement

**Return type** None

**on_new_cfp** (cfp)
Called when a new CFP for a product for which the agent registered interest is published

**on_new_report** (report)
Called whenever a financial report is published.

**Parameters**


**Remarks:**

- Agents must opt-in to receive these calls by calling *receive_financial_reports* on their AWI

**on_remove_cfp** (cfp)
Called when a new CFP for a product for which the agent registered interest is removed

**classmethod read_config** (config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

**Parameters**

- **config** (*Union*[str, dict]*) – Either a file name or a dictionary
- **section** (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

**Return type** Dict[str, Any]

**Returns** A dict ready to be parsed by from_config

**Remarks:**

**request_negotiation** (cfp, negotiator=None, ufun=None)
Requests a negotiation from the AWI while keeping track of available negotiation requests

**Parameters**

- **cfp** (*CFP*) –
- **negotiator** (Optional[Negotiator]) –
- **ufun** (Optional[UtilityFunction]) –

**Return type** bool

**Returns** Whether the negotiation request was successful indicating that the partner accepted the negotiation
respond_to_negotiation_request (cfp, partner)
Called when a prospective partner requests a negotiation to start

Return type Optional[Negotiator]

respond_to_negotiation_request_ (initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received

Return type Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)
Called to respond to a renegotiation request

Parameters
  • agenda (RenegotiationRequest) –
  • contract (Contract) –
  • breaches (List[Breach]) –

Returns:

Return type Optional[Negotiator]

set_renegotiation_agenda (contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters
  • contract (Contract) – The contract being breached
  • breaches (List[Breach]) – All breaches on contract

Return type Optional[RenegotiationRequest]

Returns
Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

sign_contract (contract)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type Optional[str]

step()
Called by the simulator at every simulation step

step_()
Called at every time-step. This function is called directly by the world.

ConsumptionProfile

class negmas.apps.scml.ConsumptionProfile (schedule=0, underconsumption=0.1, overconsumption=0.01, dynamicity=0.0, cv=0.1, alpha_q=0.5, alpha_u=1.0, beta_q=10.0, beta_u=10.0, tau_q=2, tau_u=0.25)

Bases: object

Attributes Summary

alpha_q
alpha_u
Table 70 – continued from previous page

<table>
<thead>
<tr>
<th>beta_q</th>
<th>beta_u</th>
<th>cv</th>
<th>dynamicity</th>
<th>overconsumption</th>
<th>schedule</th>
<th>tau_q</th>
<th>tau_u</th>
<th>underconsumption</th>
</tr>
</thead>
</table>

**Methods Summary**

```python
classmethod random()
schedule_at(time)               rtype int
schedule_within(time)           rtype int
set_schedule_at(time, value, n_steps) rtype None
```

**Attributes Documentation**

alpha_q = 0.5  
alpha_u = 1.0  
beta_q = 10.0  
beta_u = 10.0  
cv = 0.1  
dynamicity = 0.0  
overconsumption = 0.01  
schedule = 0  
tau_q = 2  
tau_u = 0.25  
underconsumption = 0.1

**Methods Documentation**

classmethod random()  
schedule_at(time)  
    Return type int  
schedule_within(time)  
    Return type int  
set_schedule_at(time, value, n_steps)  
    Return type None
ScheduleDrivenConsumer

class negmas.apps.scml.ScheduleDrivenConsumer(profiles=None, negotiator_type='negmas.sao.AspirationNegotiator', consumption_horizon=20, immediate_cfp_update=True, name=None)

Bases: negmas.apps.scml.Consumer

Consumer class

Attributes Summary

<table>
<thead>
<tr>
<th>MAX_UNIT_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RELATIVE_MAX_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The unique ID of this entity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>The unique ID of this entity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A convenient name of the entity (intended primarily for printing/logging/debugging).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>requested_negotiations</th>
</tr>
</thead>
<tbody>
<tr>
<td>The negotiations currently requested by the agent.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>running_negotiations</th>
</tr>
</thead>
<tbody>
<tr>
<td>The negotiations currently requested by the agent.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>short_type_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns a short name of the type of this entity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the name of the type of this entity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>unsigned_contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>All contracts that are not yet signed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uuid</th>
</tr>
</thead>
<tbody>
<tr>
<td>The unique ID of this entity.</td>
</tr>
</tbody>
</table>

Methods Summary

<table>
<thead>
<tr>
<th>can_expect_agreement(cfp, margin)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>confirm_contract_execution(contract)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called before executing any agreement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>confirm_loan(loan, bankrupt_if_rejected)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>confirm_partial_execution(contract, breaches)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will be called whenever a contract cannot be fully executed due to breaches by the other partner.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>create(*args, **kwargs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates an object and returns a proxy to it.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>from_config(config[, section, ...])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates an object of this class given the configuration info.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called to initialize the agent after the world is initialized.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>The initialization function called by the world directly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>notify(notifiable, notification)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will be called whenever any agent goes bankrupt.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>on_cash_transfer(amount, cause)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received whenever money is transferred to the factory or from it.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>on_contract_breached(contract, breaches, ...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called after complete processing of a contract that involved a breach.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>on_contract_cancelled(contract, rejectors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>on_contract_cancelled_(contract, rejectors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 73 – continued from previous page

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_contract_executed(contract)</code></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><code>on_contract_nullified(contract, ...)</code></td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.</td>
</tr>
<tr>
<td><code>on_contract_signed(contract)</code></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><code>on_contract_signed_(contract)</code></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><code>on_event(event, sender)</code></td>
<td></td>
</tr>
<tr>
<td><code>on_inventory_change(product, quantity, cause)</code></td>
<td>Received whenever something moves in or out of the factory’s storage.</td>
</tr>
<tr>
<td><code>on_neg_request_accepted(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><code>on_neg_request_accepted_(req_id, mechanism)</code></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><code>on_neg_request_rejected(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected.</td>
</tr>
<tr>
<td><code>on_neg_request_rejected_(req_id, by)</code></td>
<td>Called when a requested negotiation is rejected.</td>
</tr>
<tr>
<td><code>on_negotiation_failure(partners, annotation, ...)</code></td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td><code>on_negotiation_failure_(partners, ...)</code></td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td><code>on_negotiation_success(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td><code>on_negotiation_success_(contract, mechanism)</code></td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td><code>on_new_cfp(cfp)</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is published.</td>
</tr>
<tr>
<td><code>on_new_report(report)</code></td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td><code>on_remove_cfp(cfp)</code></td>
<td>Called when a new CFP for a product for which the agent registered interest is removed.</td>
</tr>
<tr>
<td><code>read_config(config[, section])</code></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td><code>register_product_cfps(p, t, profile)</code></td>
<td></td>
</tr>
<tr>
<td><code>request_negotiation(cfp[, negotiator, ufun])</code></td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests.</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request(cfp, partner)</code></td>
<td>Called when a prospective partner requests a negotiation to start.</td>
</tr>
<tr>
<td><code>respond_to_negotiation_request_(initiator, ...)</code></td>
<td>Called when a negotiation request is received.</td>
</tr>
<tr>
<td><code>respond_to_renegotiation_request_(contract, ...)</code></td>
<td>Called to respond to a renegotiation request.</td>
</tr>
<tr>
<td><code>set_profiles(profiles)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_renegotiation_agenda(contract, breaches)</code></td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails.</td>
</tr>
<tr>
<td><code>sign_contract(contract)</code></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><code>step()</code></td>
<td>Called by the simulator at every simulation step.</td>
</tr>
<tr>
<td><code>step_()</code></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

Attributes Documentation

MAX_UNIT_PRICE = 100.0
RELATIVE_MAX_PRICE = 1.5
awi

Returns the Agent-World-Interface through which the agent does all of its actions in the world.

A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

Return type: `SCMLAWI`

id

The unique ID of this entity

name

A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations

The negotiations currently requested by the agent.

  Return type: `List[NegotiationRequestInfo]`

  Returns: A list of negotiation request information objects (`NegotiationRequestInfo`)

running_negotiations

The negotiations currently requested by the agent.

  Return type: `List[RunningNegotiationInfo]`

  Returns: A list of negotiation information objects (`RunningNegotiationInfo`)

short_type_name

Returns a short name of the type of this entity

type_name

Returns the name of the type of this entity

unsigned_contracts

All contracts that are not yet signed.

  Return type: `List[Contract]`

uuid

The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)

Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters

- margin (int)
- cfp (CFP)

Returns:

confirm_contract_execution (contract)

Called before executing any agreement

  Return type: `bool`

confirm_loan (loan, bankrupt_if_rejected)

called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

  Return type: `bool`

confirm_partial_execution (contract, breaches)

Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters
• **contract** (*Contract*) – The contract that was breached

• **breaches** (*List[Breach]*) – A list of all the breaches committed.

Remarks:

• Will not be called if both partners committed breaches.

**Return type** bool

classmethod create(*args, **kwargs)

Creates an object and returns a proxy to it.

classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)

Creates an object of this class given the configuration info

Parameters

• **config** (*Union[str, dict]*) – Either a file name or a dictionary

• **section** (*Optional[str]*) – A section in the file or a key in the dictionary to use for loading params

• **ignore_children** (bool) – If true then children will be ignored and there will be a single return

• **try_parsing_children** (bool) – If true the children will first be parsed as ConfigReader classes if they are not types (e.g. int, str, float, Iterable[int|str|float])

• **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are

• **any children that are to be parsed.**

Returns

An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

init()

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

init__()

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.

2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.

3. registers interest in all products that the agent can produce or consume in its factory.

4. finally it calls any custom initialization logic implemented in ‘init()’
See also:

init, step

notify (notifiable, notification)

on_agent_bankrupt (agent_id)
Will be called whenever any agent goes bankrupt

Parameters agent_id (str) – The ID of the agent that went bankrupt

Remarks:

• Agents can go bankrupt in two cases:
  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).

• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)
Received whenever money is transferred to the factory or from it.

Parameters

• amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).

• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breached (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters

• contract (Contract) – The contract

• breaches (List[Breach]) – All breaches committed (even if they were resolved)

• resolution (Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

on_contract_cancelled (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

on_contract_cancelled_ (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None
on_contract_executed (contract)
   Called after successful contract execution for which the agent is one of the partners.
   
   Return type None

on_contract_nullified (contract, bankrupt_partner, compensation)
   Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt
   
   Return type None

on_contract_signed (contract)
   Called whenever a contract is signed by all partners

on_contract_signed_ (contract)
   Called whenever a contract is signed by all partners
   
   Return type None

on_event (event, sender)

on_inventory_change (product, quantity, cause)
   Received whenever something moves in or out of the factory’s storage

   Parameters
   • product (int) – Product index.
   • quantity (int) – Negative value for products moving out and positive value for products moving in
   • cause (str) – The cause of the change. Possibilities include:
     – contract: Contract execution
     – insurance: Received from insurance company
     – bankruptcy: Liquidated due to bankruptcy
     – transport: Arrival of goods (when transportation delay in the system is > 0).
   
   Return type None

on_neg_request_accepted (req_id, mechanism)
   Called when a requested negotiation is accepted

on_neg_request_accepted_ (req_id, mechanism)
   Called when a requested negotiation is accepted

on_neg_request_rejected (req_id, by)
   Called when a requested negotiation is rejected

   Parameters
   • req_id (str) – The request ID passed to _request_negotiation
   • by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_neg_request_rejected_ (req_id, by)
   Called when a requested negotiation is rejected

   Parameters
   • req_id (str) – The request ID passed to _request_negotiation
   • by (Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

on_negotiation_failure (partners, annotation, mechanism, state)
   Called whenever a negotiation ends without agreement
   
   Return type None

6.1. negmas.apps.scml Package
on_negotiation_failure (partners, annotation, mechanism, state)
   Called whenever a negotiation ends without agreement
   Return type None

on_negotiation_success (contract, mechanism)
   Called whenever a negotiation ends with agreement
   Return type None

on_negotiation_success_ (contract, mechanism)
   Called whenever a negotiation ends with agreement
   Return type None

on_new_cfp (cfp)
   Called when a new CFP for a product for which the agent registered interest is published
   Return type None

on_new_report (report)
   Called whenever a financial report is published.
   Parameters report (FinancialReport) – The financial report giving details of the
   standing of an agent at some time (see FinancialReport)
   Remarks:
   • Agents must opt-in to receive these calls by calling receive_financial_reports on their
   AWI

on_remove_cfp (cfp)
   Called when a new CFP for a product for which the agent registered interest is removed

classmethod read_config (config, section=None)
   Reads the configuration from a file or a dict and prepares it for parsing
   Parameters
   • config (Union[str, dict]) – Either a file name or a dictionary
   • section (Optional[str]) – A section in the file or a key in the dictionary to use
     for loading params
   Return type Dict[str, Any]
   Returns A dict ready to be parsed by from_config
   Remarks:
   register_product_cfps (p, t, profile)

request_negotiation (cfp, negotiator=None, ufun=None)
   Requests a negotiation from the AWI while keeping track of available negotiation requests
   Parameters
   • cfp (CFP) –
   • negotiator (Optional[Negotiator]) –
   • ufun (Optional[UtilityFunction]) –
   Return type bool
   Returns Whether the negotiation request was successful indicating that the partner accepted
   the negotiation

respond_to_negotiation_request (cfp, partner)
   Called when a prospective partner requests a negotiation to start
   Return type Optional[Negotiator]
respond_to_negotiation_request (initiator, partners, issues, annotation, mechanism, role, req_id)
Called when a negotiation request is received

Return type Optional[Negotiator]

respond_to_renegotiation_request (contract, breaches, agenda)
Called to respond to a renegotiation request

Parameters

• agenda (RenegotiationRequest) – Renegotiation agenda (issues to renegotiate about).
• contract (Contract) – The contract that was breached
• breaches (List[Breach]) – All breaches on that contract

Return type Optional[Negotiator]

Returns None to refuse to enter the negotiation, otherwise, a negotiator to use for this negotiation.

set_profiles (profiles)

set_renegotiation_agenda (contract, breaches)
Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

Parameters

• contract (Contract) – The contract that was breached about which renegotiation is offered
• breaches (List[Breach]) – The list of breaches by all parties for the breached contract.

Return type Optional[RenegotiationRequest]

Returns None if renegotiation is not to be started, otherwise a re-negotiation agenda.

sign_contract (contract)
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type Optional[str]

step ()
Called by the simulator at every simulation step

step ()
Called at every time-step. This function is called directly by the world.

Miner

class negmas.apps.scml.Miner (name=None)
Bases: negmas.apps.scml.SCMLAgent, abc.ABC
Base class of all miners

Attributes Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awi</td>
<td>Returns the Agent-World-Interface through which the agent does all of its actions in the world.</td>
</tr>
<tr>
<td>id</td>
<td>The unique ID of this entity</td>
</tr>
</tbody>
</table>

Continued on next page
Table 74 – continued from previous page

<table>
<thead>
<tr>
<th>name</th>
<th>A convenient name of the entity (intended primarily for printing/logging/debugging).</th>
</tr>
</thead>
<tbody>
<tr>
<td>requested_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>running_negotiations</td>
<td>The negotiations currently requested by the agent.</td>
</tr>
<tr>
<td>short_type_name</td>
<td>Returns a short name of the type of this entity.</td>
</tr>
<tr>
<td>type_name</td>
<td>Returns the name of the type of this entity.</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>All contracts that are not yet signed.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of this entity.</td>
</tr>
</tbody>
</table>

Methods Summary

- **can_expect_agreement**(cfp, margin) Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable.
- **confirm_contract_execution**(contract) Called before executing any agreement.
- **confirm_loan**(loan, bankrupt_if_rejected) Called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached.
- **confirm_partial_execution**(contract, breaches) Will be called whenever a contract cannot be fully executed due to breaches by the other partner.
- **create**(args, **kwargs) Creates an object and returns a proxy to it.
- **from_config**(config[, section, ...]) Creates an object of this class given the configuration info.
- **init**(Config) Called to initialize the agent after the world is initialized.
- **init_**() The initialization function called by the world directly.
- **notify**(notifiable, notification) Will be called whenever any agent goes bankrupt.
- **on_agent_bankrupt**(agent_id) Will be called whenever any agent goes bankrupt.
- **on_cash_transfer**(amount, cause) Received whenever money is transferred to the factory or from it.
- **on_contract_breached**(contract, breaches, ...) Called after complete processing of a contract that involved a breach.
- **on_contract_cancelled**(contract, rejec-tors) Called whenever at least a partner did not sign the contract.
- **on_contract_cancelled_**(contract, rejec-tors) Called whenever at least a partner did not sign the contract.
- **on_contract_executed**(contract) Called after successful contract execution for which the agent is one of the partners.
- **on_contract_nullified**(contract, ...) Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.
- **on_contract_signed**(contract, ...) Called whenever a contract is signed by all partners.
- **on_contract_signed_**(contract) Called whenever a contract is signed by all partners.
- **on_event**(event, sender) Called when an event occurs.
- **on_inventory_change**(product, quantity, cause) Received whenever something moves in or out of the factory’s storage.
- **on_neg_request_accepted**(req_id, mechanism) Called when a requested negotiation is accepted.
- **on_neg_request_accepted_**(req_id, mechanism) Called when a requested negotiation is accepted.
- **on_neg_request_rejected**(req_id, by) Called when a requested negotiation is rejected.

Continued on next page
on_neg_request_rejected_(req_id, by) Called when a requested negotiation is rejected

on_negotiation_failure_(partners, annotation, ...) Called whenever a negotiation ends without agreement

on_negotiation_failure_(partners, ...) Called whenever a negotiation ends without agreement

on_negotiation_success_(contract, mechanism) Called whenever a negotiation ends with agreement

on_negotiation_success_(contract, mechanism) Called whenever a negotiation ends with agreement

on_new_cfp_(cfp) Called when a new CFP for a product for which the agent registered interest is published

on_new_report_(report) Called whenever a financial report is published.

on_remove_cfp_(cfp) Called when a new CFP for a product for which the agent registered interest is removed

read_config_(config[, section]) Reads the configuration from a file or a dict and prepares it for parsing

request_negotiation_(cfp[, negotiator, ufumi]) Requests a negotiation from the AWI while keeping track of available negotiation requests

respond_to_negotiation_request_(cfp, partner) Called when a prospective partner requests a negotiation to start

respond_to_negotiation_request_(initiator, ...) Called when a negotiation request is received ...

respond_to_renegotiation_request_(contract, breaches) Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

sign_contract_(contract) Called after the signing delay from contract conclusion to sign the contract.

step() Called by the simulator at every simulation step

step_() Called at every time-step.

Attributes Documentation

awi
Returns the Agent-World-Interface through which the agent does all of its actions in the world.

A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

Return type SCMLAWI

id
The unique ID of this entity

name
A convenient name of the entity (intended primarily for printing/logging/debugging).

requested_negotiations
The negotiations currently requested by the agent.

Return type List[NegotiationRequestInfo]

Returns A list of negotiation request information objects (NegotiationRequestInfo)

running_negotiations
The negotiations currently requested by the agent.

Return type List[RunningNegotiationInfo]

Returns A list of negotiation information objects (RunningNegotiationInfo)
short_type_name
Returns a short name of the type of this entity

type_name
Returns the name of the type of this entity

unsigned_contracts
All contracts that are not yet signed.
  Return type List[Contract]

uuid
The unique ID of this entity

Methods Documentation

can_expect_agreement (cfp, margin)
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

Parameters
  • margin (int) –
  • cfp (CFP) –

Returns:

confirm_contract_execution (contract)
Called before executing any agreement

Return type bool

confirm_loan (loan, bankrupt_if_rejected)
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

Return type bool

confirm_partial_execution (contract, breaches)
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

Parameters
  • contract (Contract) – The contract that was breached
  • breaches (List[Breach]) – A list of all the breaches committed.

Remarks:
  • Will not be called if both partners committed breaches.

Return type bool

classmethod create (*args, **kwargs)
Creates an object and returns a proxy to it.

classmethod from_config (config, section=None, ignore_children=True,
  try_parsing_children=True, scope=None)
Creates an object of this class given the configuration info

Parameters
  • config (Union[str, dict]) – Either a file name or a dictionary
  • section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params
  • ignore_children (bool) – If true then children will be ignored and there will be a single return
• **try_parsing_children** *(bool)* – If true the children will first be parsed as `ConfigReader` classes if they are not

• **types** *(e.g. int, str, float, Iterable[int|str|float]|(simple)) –*

• **scope** – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are

• **any children that are to be parsed.** *(having)* –

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.

• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

**init**()

Called to initialize the agent after the world is initialized. the AWI is accessible at this point.

**init_**()

The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.

2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.

3. registers interest in all products that the agent can produce or consume in its factory.

4. finally it calls any custom initialization logic implemented in ‘init’()

**See also:**

*init, step*

**notify** *(notifiable, notification)*

**on_agent_bankrupt** *(agent_id)*

Will be called whenever any agent goes bankrupt

**Parameters**

agent_id *(str)* – The ID of the agent that went bankrupt

Remarks:

• Agents can go bankrupt in two cases:

1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.

2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).

• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their `on_new_cfp` and `respond_to_negotiation_request` callbacks by pulling the bulletin-board using the helper function `is_bankrupt` of their AWI.

**Return type** None
on_cash_transfer (amount, cause)
Received whenever money is transferred to the factory or from it.

Parameters

• amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).
• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breached (contract, breaches, resolution)
Called after complete processing of a contract that involved a breach.

Parameters

• contract (Contract) – The contract
• breaches (List[Breach]) – All breaches committed (even if they were resolved)
• resolution (Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

on_contract_cancelled (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

on_contract_cancelled_ (contract, rejectors)
Called whenever at least a partner did not sign the contract

Return type None

on_contract_executed (contract)
Called after successful contract execution for which the agent is one of the partners.

Return type None

on_contract_nullified (contract, bankrupt_partner, compensation)
Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt

Return type None

on_contract_signed (contract)
Called whenever a contract is signed by all partners

Return type None

on_contract_signed_ (contract)
Called whenever a contract is signed by all partners

Return type None

on_event (event, sender)

on_inventory_change (product, quantity, cause)
Received whenever something moves in or out of the factory’s storage

Parameters

• product (int) – Product index.
• **quantity** (int) – Negative value for products moving out and positive value for products moving in

• **cause** (str) – The cause of the change. Possibilities include:
  - contract: Contract execution
  - insurance: Received from insurance company
  - bankruptcy: Liquidated due to bankruptcy
  - transport: Arrival of goods (when transportation delay in the system is > 0).

  **Return type** None

  **on_neg_request_accepted**(req_id, mechanism)
  Called when a requested negotiation is accepted

  **on_neg_request_accepted_**(req_id, mechanism)
  Called when a requested negotiation is accepted

  **on_neg_request_rejected**(req_id, by)
  Called when a requested negotiation is rejected

  **Parameters**
  - **req_id**(str) – The request ID passed to _request_negotiation
  - **by**(Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

  **on_neg_request_rejected_**(req_id, by)
  Called when a requested negotiation is rejected

  **Parameters**
  - **req_id**(str) – The request ID passed to _request_negotiation
  - **by**(Optional[List[str]]) – A list of agents that refused to participate or None if the failure was for another reason

  **on_negotiation_failure**(partners, annotation, mechanism, state)
  Called whenever a negotiation ends without agreement

  **Return type** None

  **on_negotiation_failure_**(partners, annotation, mechanism, state)
  Called whenever a negotiation ends without agreement

  **Return type** None

  **on_negotiation_success**(contract, mechanism)
  Called whenever a negotiation ends with agreement

  **Return type** None

  **on_negotiation_success_**(contract, mechanism)
  Called whenever a negotiation ends with agreement

  **Return type** None

  **on_new_cfp**(cfp)
  Called when a new CFP for a product for which the agent registered interest is published

  **on_new_report**(report)
  Called whenever a financial report is published.

  **Parameters**

  **Remarks:**
Agents must opt-in to receive these calls by calling `receive_financial_reports` on their AWI

```python
on_remove_cfp(cfp)
```

Called when a new CFP for a product for which the agent registered interest is removed

```python
classmethod read_config(config, section=None)
```

Reads the configuration from a file or a dict and prepares it for parsing

**Parameters**

- `config` (`Union[str, dict]`) – Either a file name or a dictionary
- `section` (`Optional[str]`) – A section in the file or a key in the dictionary to use for loading params

**Returns**

A dict ready to be parsed by `from_config`

**Remarks:**

```python
request_negotiation(cfp, negotiator=None, ufun=None)
```

Requests a negotiation from the AWI while keeping track of available negotiation requests

**Parameters**

- `cfp` (`CFP`) –
- `negotiator` (`Optional[Negotiator]`) –
- `ufun` (`Optional[UtilityFunction]`) –

**Returns**

Whether the negotiation request was successful indicating that the partner accepted the negotiation

```python
respond_to_negotiation_request(cfp, partner)
```

Called when a prospective partner requests a negotiation to start

**Return type**

`Optional[Negotiator]`

```python
respond_to_negotiation_request_(initiator, partners, issues, annotation, mechanism, role, req_id)
```

Called when a negotiation request is received

**Return type**

`Optional[Negotiator]`

```python
respond_to_renegotiation_request(contract, breaches, agenda)
```

Called to respond to a renegotiation request

**Parameters**

- `agenda` (`RenegotiationRequest`) –
- `contract` (`Contract`) –
- `breaches` (`List[Breach]`) –

**Returns:**

**Return type**

`Optional[Negotiator]`

```python
set_renegotiation_agenda(contract, breaches)
```

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- `contract` (`Contract`) – The contract being breached
- `breaches` (`List[Breach]`) – All breaches on `contract`
Return type  Optional[RenegotiationRequest]

Returns  Renegotiation agenda (issues to negotiate about to avoid reporting the breaches).

**sign_contract** *(contract)*
Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

Return type  Optional[str]

**step** ()
Called by the simulator at every simulation step

**step_** ()
Called at every time-step. This function is called directly by the world.

**MiningProfile**

```python
class negmas.apps.scml.MiningProfile(cv=0.05, alpha_t=1.0, alpha_q=1.0, alpha_u=1.0, 
beta_t=1.0, beta_q=100.0, beta_u=100.0, tau_t=-0.25, tau_q=0.25, tau_u=1.0)

Bases: object
```

**Attributes Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha_q</td>
<td>1.0</td>
</tr>
<tr>
<td>alpha_t</td>
<td>1.0</td>
</tr>
<tr>
<td>alpha_u</td>
<td>1.0</td>
</tr>
<tr>
<td>beta_q</td>
<td>100.0</td>
</tr>
<tr>
<td>beta_t</td>
<td>1.0</td>
</tr>
<tr>
<td>beta_u</td>
<td>100.0</td>
</tr>
<tr>
<td>cv</td>
<td>0.05</td>
</tr>
<tr>
<td>tau_q</td>
<td>0.25</td>
</tr>
<tr>
<td>tau_t</td>
<td>-0.25</td>
</tr>
<tr>
<td>tau_u</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Methods Summary**

**Attributes Documentation**

- **alpha_q** = 1.0
- **alpha_t** = 1.0
- **alpha_u** = 1.0
- **beta_q** = 100.0
- **beta_t** = 1.0
- **beta_u** = 100.0
- **cv** = 0.05
- **tau_q** = 0.25
- **tau_t** = -0.25
- **tau_u** = 1.0
Methods Documentation

```
classmethod random()
```

ReactiveMiner

```
class negmas.apps.scml.ReactiveMiner(profiles=None, negotiator_type='negmas.sao.AspirationNegotiator', n_retrials=0, reactive=True, name=None)
```

Bases: negmas.apps.scml.Mineral

Raw Material Generator

Attributes Summary

- **awi**: Returns the Agent-World-Interface through which the agent does all of its actions in the world.
- **id**: The unique ID of this entity
- **name**: A convenient name of the entity (intended primarily for printing/logging/debugging).
- **requested_negotiations**: The negotiations currently requested by the agent.
- **running_negotiations**: The negotiations currently requested by the agent.
- **short_type_name**: Returns a short name of the type of this entity
- **type_name**: Returns the name of the type of this entity
- **unsigned Contracts**: All contracts that are not yet signed.
- **uuid**: The unique ID of this entity

Methods Summary

- **can_expect_agreement(cfp, margin)**: Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable
- **confirm_contract_execution(contract)**: Called before executing any agreement
- **confirm_loan(loan, bankrupt_if_rejected)**: called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached
- **confirm_partial_execution(contract, breaches)**: Will be called whenever a contract cannot be fully executed due to breaches by the other partner.
- **create(*args, **kwargs)**: Creates an object and returns a proxy to it.
- **from_config(config[, section, ...])**: Creates an object of this class given the configuration info
- **init()**: Called to initialize the agent after the world is initialized.
- **init_()**: The initialization function called by the world directly.
- **notify(notifiable, notification)**: Will be called whenever any agent goes bankrupt
- **on_agent_bankrupt(agent_id)**: Received whenever money is transferred to the factory or from it.
- **on_cash_transfer(amount, cause)**: Called after complete processing of a contract that involved a breach.
- **on_contract_cancelled(contract, rejectors)**: Called whenever at least a partner did not sign the contract

Continued on next page
Table 79 – continued from previous page

<table>
<thead>
<tr>
<th>Event NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>on_contract_cancelled</strong></td>
<td>Called whenever at least a partner did not sign the contract.</td>
</tr>
<tr>
<td><strong>on_contract_executed</strong></td>
<td>Called after successful contract execution for which the agent is one of the partners.</td>
</tr>
<tr>
<td><strong>on_contract_nullified</strong></td>
<td>Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.</td>
</tr>
<tr>
<td><strong>on_contract_signed</strong></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><strong>on_contract_signed_</strong></td>
<td>Called whenever a contract is signed by all partners.</td>
</tr>
<tr>
<td><strong>on_event</strong></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><strong>on_inventory_change</strong></td>
<td>Received whenever something moves in or out of the factory’s storage.</td>
</tr>
<tr>
<td><strong>on_neg_request_accepted</strong></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><strong>on_neg_request_accepted_</strong></td>
<td>Called when a requested negotiation is accepted.</td>
</tr>
<tr>
<td><strong>on_neg_request_rejected</strong></td>
<td>Called when a requested negotiation is rejected.</td>
</tr>
<tr>
<td><strong>on_neg_request_rejected_</strong></td>
<td>Called when a requested negotiation is rejected.</td>
</tr>
<tr>
<td><strong>on_negotiation_failure</strong></td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td><strong>on_negotiation_failure_</strong></td>
<td>Called whenever a negotiation ends without agreement.</td>
</tr>
<tr>
<td><strong>on_negotiation_success</strong></td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td><strong>on_negotiation_success_</strong></td>
<td>Called whenever a negotiation ends with agreement.</td>
</tr>
<tr>
<td><strong>on_new_cfp</strong></td>
<td>Called when a new CFP for a product for which the agent registered interest is published.</td>
</tr>
<tr>
<td><strong>on_new_report</strong></td>
<td>Called whenever a financial report is published.</td>
</tr>
<tr>
<td><strong>on_remove_cfp</strong></td>
<td>Called when a new CFP for a product for which the agent registered interest is removed.</td>
</tr>
<tr>
<td><strong>read_config</strong></td>
<td>Reads the configuration from a file or a dict and prepares it for parsing.</td>
</tr>
<tr>
<td><strong>request_negotiation</strong></td>
<td>Requests a negotiation from the AWI while keeping track of available negotiation requests.</td>
</tr>
<tr>
<td><strong>respond_to_negotiation_request</strong></td>
<td>Called when a prospective partner requests a negotiation to start.</td>
</tr>
<tr>
<td><strong>respond_to_negotiation_request</strong></td>
<td>Called when a negotiation request is received.</td>
</tr>
<tr>
<td><strong>respond_to_renegotiation_request</strong></td>
<td>Called to respond to a renegotiation request.</td>
</tr>
<tr>
<td><strong>set_profiles</strong></td>
<td>Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails.</td>
</tr>
<tr>
<td><strong>sign_contract</strong></td>
<td>Called after the signing delay from contract conclusion to sign the contract.</td>
</tr>
<tr>
<td><strong>step</strong></td>
<td>Called by the simulator at every simulation step.</td>
</tr>
<tr>
<td><strong>step_</strong></td>
<td>Called at every time-step.</td>
</tr>
</tbody>
</table>

### Attributes Documentation

**awi**

Returns the Agent-World-Interface through which the agent does all of its actions in the world.
A single exception is request_negotiation for which it is recommended to actually call the helper method on the agent itself instead of directly calling the AWI version.

**Return type** \textit{SCMLAWI}

**id**
The unique ID of this entity

**name**
A convenient name of the entity (intended primarily for printing/logging/debugging).

**requested_negotiations**
The negotiations currently requested by the agent.

**Return type** \textit{List[NegotiationRequestInfo]}

**Returns** A list of negotiation request information objects (NegotiationRequestInfo)

**running_negotiations**
The negotiations currently requested by the agent.

**Return type** \textit{List[RunningNegotiationInfo]}

**Returns** A list of negotiation information objects (RunningNegotiationInfo)

**short_type_name**
Returns a short name of the type of this entity

**type_name**
Returns the name of the type of this entity

**unsigned_contracts**
All contracts that are not yet signed.

**Return type** \textit{List[Contract]}

**uuid**
The unique ID of this entity

### Methods Documentation

**can_expect_agreement** \textit{(cfp, margin)}
Checks if it is possible in principle to get an agreement on this CFP by the time it becomes executable

**Parameters**

- margin \textit{(int)} –
- cfp \textit{(CFP)} –

**Returns:**

**confirm_contract_execution** \textit{(contract)}
Called before executing any agreement

**Return type** \textit{bool}

**confirm_loan** \textit{(loan, bankrupt_if_rejected)}
called by the world manager to confirm a loan if needed by the buyer of a contract that is about to be breached

**Return type** \textit{bool}

**confirm_partial_execution** \textit{(contract, breaches)}
Will be called whenever a contract cannot be fully executed due to breaches by the other partner.

**Parameters**

- \textit{contract (Contract)} – The contract that was breached
• **breaches** (*List[Breach]*) – A list of all the breaches committed.

Remarks:
• Will not be called if both partners committed breaches.

**Return type** `bool`

```python
classmethod create(*args, **kwargs)
```
Creates an object and returns a proxy to it.

```python
classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)
```
Creates an object of this class given the configuration info

**Parameters**

- **config** (*Union[str, dict]*) – Either a file name or a dictionary
- **section** (*Optional[str]*) – A section in the file or a key in the dictionary to use for loading params
- **ignore_children** (*bool*) – If true then children will be ignored and there will be a single return
- **try_parsing_children** (*bool*) – If true the children will first be parsed as `ConfigReader` classes if they are not

**types** *(e.g. int, str, float, Iterable[int|str|float]) (simple)* –

- **scope** – The scope at which to evaluate any child classes. This MUST be passed as `scope=globals()` if you are

**Returns** An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:
• This function will return an object of its class after passing the key-value pairs found in the config to the init function.
• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

```python
init()
```
Called to initialize the agent after the world is initialized. The AWI is accessible at this point.

```python
init_()
```
The initialization function called by the world directly.

It does the following actions by default:

1. copies some of the static world settings to the agent to make them available without calling the AWI.
2. prepares production related properties like producing, consuming, line_profiles, compiled_profiles, etc.
3. registers interest in all products that the agent can produce or consume in its factory.
4. finally it calls any custom initialization logic implemented in ‘init()’

6.1. negmas.apps.scml Package
See also:

init, step

notify (notifiable, notification)

on_agent_bankrupt (agent_id)

Will be called whenever any agent goes bankrupt

Parameters agent_id (str) – The ID of the agent that went bankrupt

Remarks:

• Agents can go bankrupt in two cases:
  1. Failing to pay one installments of a loan they bought and refusing (or being unable to) get another loan to pay it.
  2. Failing to pay a penalty on a sell contract they failed to honor (and refusing or being unable to get a loan to pay for it).

• All built-in agents ignore this call and they use the bankruptcy list ONLY to decide whether or not to negotiate in their on_new_cfp and respond_to_negotiation_request callbacks by pulling the bulletin-board using the helper function is_bankrupt of their AWI.

Return type None

on_cash_transfer (amount, cause)

Received whenever money is transferred to the factory or from it.

Parameters

• amount (float) – Amount of money (negative for transfers out of the factory, positive for transfers to it).

• cause (str) – The cause of the change. Possibilities include:
  – contract: Contract execution
  – insurance: Received from insurance company
  – bankruptcy: Liquidated due to bankruptcy
  – transfer: Arrival of transferred money (when transfer delay in the system is > 0).

Return type None

on_contract_breached (contract, breaches, resolution)

Called after complete processing of a contract that involved a breach.

Parameters

• contract (Contract) – The contract

• breaches (List[Breach]) – All breaches committed (even if they were resolved)

• resolution (Optional[Contract]) – The resolution contract if re-negotiation was successful. None if not.

Return type None

on_contract_cancelled (contract, rejectors)

Called whenever at least a partner did not sign the contract.

Return type None

on_contract_cancelled_ (contract, rejectors)

Called whenever at least a partner did not sign the contract.

Return type None
on_contract_executed \( (\text{contract}) \)

Called after successful contract execution for which the agent is one of the partners.

Return type None

on_contract_nullified \( (\text{contract, bankrupt_partner, compensation}) \)

Will be called whenever a contract the agent is involved in is nullified because another partner went bankrupt.

Return type None

on_contract_signed \( (\text{contract}) \)

Called whenever a contract is signed by all partners.

Return type None

on_contract_signed_ \( (\text{contract}) \)

Called whenever a contract is signed by all partners.

Return type None

on_event \( (\text{event, sender}) \)

on_inventory_change \( (\text{product, quantity, cause}) \)

Received whenever something moves in or out of the factory’s storage.

Parameters
- product \( (\text{int}) \) – Product index.
- quantity \( (\text{int}) \) – Negative value for products moving out and positive value for products moving in.
- cause \( (\text{str}) \) – The cause of the change. Possibilities include:
  - contract: Contract execution
  - insurance: Received from insurance company
  - bankruptcy: Liquidated due to bankruptcy
  - transport: Arrival of goods (when transportation delay in the system is > 0).

Return type None

on_neg_request_accepted \( (\text{req_id, mechanism}) \)

Called when a requested negotiation is accepted.

on_neg_request_accepted_ \( (\text{req_id, mechanism}) \)

Called when a requested negotiation is accepted.

on_neg_request_rejected \( (\text{req_id, by}) \)

Called when a requested negotiation is rejected.

Parameters
- req_id \( (\text{str}) \) – The request ID passed to _request_negotiation.
- by \( (\text{Optional[List[str]]}) \) – A list of agents that refused to participate or None if the failure was for another reason.

on_neg_request_rejected_ \( (\text{req_id, by}) \)

Called when a requested negotiation is rejected.

Parameters
- req_id \( (\text{str}) \) – The request ID passed to _request_negotiation.
- by \( (\text{Optional[List[str]]}) \) – A list of agents that refused to participate or None if the failure was for another reason.

on_negotiation_failure \( (\text{partners, annotation, mechanism, state}) \)

Called whenever a negotiation ends without agreement.
Return type None

`on_negotiation_failure_(partners, annotation, mechanism, state)`
Called whenever a negotiation ends without agreement

Return type None

`on_negotiation_success_(contract, mechanism)`
Called whenever a negotiation ends with agreement

Return type None

`on_negotiation_success_(contract, mechanism)`
Called whenever a negotiation ends with agreement

Return type None

`on_new_cfp(cfp)`
Called when a new CFP for a product for which the agent registered interest is published

`on_new_report_(report)`
Called whenever a financial report is published.

Parameters `report (FinancialReport)` – The financial report giving details of the standing of an agent at some time (see `FinancialReport`)

Remarks:
• Agents must opt-in to receive these calls by calling `receive_financial_reports` on their AWI

`on_remove_cfp(cfp)`
Called when a new CFP for a product for which the agent registered interest is removed

`classmethod read_config(config, section=None)`
Reads the configuration from a file or a dict and prepares it for parsing

Parameters
• `config (Union[str, dict])` – Either a file name or a dictionary
• `section (Optional[str])` – A section in the file or a key in the dictionary to use for loading params

Return type `Dict[str, Any]`

Returns A dict ready to be parsed by from_config

Remarks:

`request_negotiation (cfp, negotiator=None, ufun=None)`
Requests a negotiation from the AWI while keeping track of available negotiation requests

Parameters
• `cfp (CFP)` –
• `negotiator (Optional[Negotiator])` –
• `ufun (Optional[UtilityFunction])` –

Return type `bool`

Returns Whether the negotiation request was successful indicating that the partner accepted the negotiation

`respond_to_negotiation_request(cfp, partner)`
Called when a prospective partner requests a negotiation to start

Return type `Optional[Negotiator]`
**respond_to_negotiation_request** *(initiator, partners, issues, annotation, mechanism, role, req_id)*

Called when a negotiation request is received

**Return type** `Optional[Negotiator]`

**respond_to_renegotiation_request** *(contract, breaches, agenda)*

Called to respond to a renegotiation request

**Parameters**

- **agenda** *(RenegotiationRequest)* –
- **contract** *(Contract)* –
- **breaches** *(List[Breach])* –

**Returns:**

**Return type** `Optional[Negotiator]`

**set_profiles** *(profiles)*

**set_renegotiation_agenda** *(contract, breaches)*

Received by partners in ascending order of their total breach levels in order to set the renegotiation agenda when contract execution fails

**Parameters**

- **contract** *(Contract)* – The contract being breached
- **breaches** *(List[Breach])* – All breaches on contract

**Returns** `Optional[RenegotiationRequest]`

**sign_contract** *(contract)*

Called after the signing delay from contract conclusion to sign the contract. Contracts become binding only after they are signed.

**Return type** `Optional[str]`

**step**

Called by the simulator at every simulation step

**step_**

Called at every time-step. This function is called directly by the world.
6.1.7 Class Inheritance Diagram
This part of the documentation describes helper modules.

### 7.1 negmas.helpers Module

A set of utilities that can be used by agents developed for the platform.

This set of utilities can be extended but must be backward compatible for at least two versions

#### 7.1.1 Functions

- **create_loggers([file_name, module_name, ...])**
  
  Create a set of loggers to report feedback.

- **snake_case(s)**
  
  Converts a string from CamelCase to snake_case.

- **camel_case(s[, capitalize_first, lower_first])**
  
  Converts a string from snake_case to CamelCase.

- **unique_name(base[, add_time, rand_digits])**
  
  Return a unique name.

- **is_nonzero_file(path)**
  
  Whether or not the path is for an existing nonzero file.

- **pretty_string(src[, tab_size, compact])**
  
  Recursively print nested elements.

- **get_class(class_name[, module_name, scope])**
  
  Imports and creates a class object for the given class name.

- **import_by_name(full_name)**
  
  Imports something form a module using its full name.

- **get_full_type_name(t)**
  
  Gets the full typename of a type.

- **instantiate(class_name[, module_name, scope])**
  
  Imports and instantiates an object of a class.

- **humanize_time(secs[, align, ...])**
  
  Prints time that is given as seconds in human readable form.

- **gmap(group, param)**
  
  Calls or indexes the group by the param.

- **ikeys(x)**
  
  Returns all keys of the iterable.

- **dump(d, file_name)**
  
  Saves an object depending on the extension of the file given.

- **add_records(file_name, data[, col_names])**
  
  Adds records to a csv file.
create_loggers

```python
negmas.helpers.create_loggers(file_name=None, module_name=None, screen_level=30,
    file_level=10, format_str='%(asctime)s - %(levelname)s - %(message)s', colored=True, app_wide_log_file=True,
    module_wide_log_file=False)
```

Create a set of loggers to report feedback.

The logger created can log to both a file and the screen at the same time with adjustable level for each of them. The default is to log everything to the file and to log WARNING at least to the screen.

**Parameters**

- `module_wide_log_file` (bool) –
- `app_wide_log_file` (bool) –
- `file_name` (Optional[str]) – The file to export to the logs to. If None only the screen is used for logging. If empty, a time-stamp is used.
- `module_name` (Optional[str]) – The module name to use. If not given the file name without .py is used.
- `screen_level` (Optional[int]) – level of the screen logger.
- `file_level` (Optional[int]) – level of the file logger.
- `format_str` (str) – the format of logged items.
- `colored` (bool) – whether or not to try using colored logs.

**Returns**

The logger

**Return type** `logging.Logger`

---

**snake_case**

```python
negmas.helpers.snake_case(s)
```

Converts a string from CamelCase to snake_case.

**Example**

```python
>>> print(snake_case('ThisIsATest'))
this_is_a_test
```

**Parameters**

- `s` (str) – input string

**Returns**

converted string

**Return type** `str`

---

**camel_case**

```python
negmas.helpers.camel_case(s, capitalize_first=False, lower_first=False)
```

Converts a string from snake_case to CamelCase.

**Example**

```python
```
Parameters

- \textit{s (str)} – input string
- \textit{capitalize_first (bool)} – if true, the first character will be capitalized
- \textit{lower_first (bool)} – If true, the first character will be lowered

Returns converted string

Return type \texttt{str}

\textbf{unique_name}

\texttt{negmas.helpers.unique_name (base, add_time=True, rand_digits=8)}

Return a unique name.

Can be used to return a unique directory name on the given base.

Parameters

- \textit{base (str)} – str (str): base path/string
- \textit{add_time (bool, optional)} – Defaults to True. Add current time
- \textit{rand_digits (int, optional)} – Defaults to 8. The number of random characters to add to the name

Examples

\begin{verbatim}
>>> a = unique_name('')
>>> len(a) == 8 + 1 + 6 + 8
True
\end{verbatim}

Returns The unique name.

Return type \texttt{str}

\textbf{is_nonzero_file}

\texttt{negmas.helpers.is_nonzero_file (fpath)}

Whether or not the path is for an existing nonzero file.

Parameters \texttt{fpath (str)} – path to the file to test. It accepts both str and \texttt{pathlib.Path}

Return type \texttt{bool}
pretty_string

`negmas.helpers.pretty_string(src, tab_size=2, compact=False)`

Recursively print nested elements.

**Parameters**

- `src (Any)` – The source to be converted to a printable string
- `tab_size (int)` – Tab size in spaces
- `compact (bool)` – If true the output is converted into a single line

**Returns**
The pretty version of the input

**Return type**
`str`

**Remarks:**

- This function assumes that the patterns `''`` and `:` do not appear anywhere in the input. If they appear, the space, `:` will be removed.

get_class

`negmas.helpers.get_class(class_name, module_name=None, scope=None)`

Imports and creates a class object for the given class name

**Return type**
`Type` [+CT_co]

import_by_name

`negmas.helpers.import_by_name(full_name)`

Imports something from a module using its full name

**Return type**
`Any`

get_full_type_name

`negmas.helpers.get_full_type_name(t)`

Gets the full typename of a type. You should not pass an instance to this function but it may just work.

An exception is that if the input is of type `str` or if it is `None`, it will be returned as it is

**Return type**
`str`

instantiate

`negmas.helpers.instantiate(class_name, module_name=None, scope=None, **kwargs)`

Imports and instantiates an object of a class

**Return type**
`Any`

humanize_time

`negmas.helpers.humanize_time(secs, align=False, always_show_all_units=False)`

Prints time that is given as seconds in human readable form. Useful only for times >=1sec.

**Parameters**

- `secs` – float: number of seconds
• **align**(bool) – bool, optional: whether to align outputs so that they all take the same size (not implemented)

• **always_show_all_units**(bool) – bool, optional: Whether to always show days, hours, and minutes even when they are zeros. default False

**Returns**  str: formated string with the humanized form

**dump**

def negmas.helpers.dump(d, file_name)

Saves an object depending on the extension of the file given. If the filename given has no extension, **DEFAULT_DUMP_EXTENSION** will be used

**Parameters**

• **d**(Any) – Object to save

• **file_name**(Union[str, PathLike]) – file name

**Remarks:**

• Supported formats are json, yaml

• If None is given, the file will be created but will be empty

**Return type**  None

**add_records**

def negmas.helpers.add_records(file_name, data, col_names=None)

Adds records to a csv file

**Parameters**

• **file_name**(Union[str, PathLike]) – file name

• **data**(Any) – data to use for creating the record

• **col_names**(Optional[List[str]]) – Names in the data.

**Return type**  None

**Returns**  None

### 7.1.2 Classes

<table>
<thead>
<tr>
<th>ReturnCause</th>
<th>An enumeration.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distribution</strong>(dtype, **kwargs)</td>
<td>Any distribution from scipy.stats with overloading of addition and multiplication.</td>
</tr>
<tr>
<td><strong>ConfigReader</strong></td>
<td>alias of typing.List</td>
</tr>
</tbody>
</table>

**ReturnCause**

class negmas.helpers.ReturnCause

Bases: enum.Enum

An enumeration.
Attributes Summary

<table>
<thead>
<tr>
<th>FAILURE</th>
<th>SUCCESS</th>
<th>TIMEOUT</th>
</tr>
</thead>
</table>

Attributes Documentation

FAILURE = 2
SUCCESS = 1
TIMEOUT = 0

Distribution

class negmas.helpers.Distribution(dtype, **kwargs)

Bases: object

Any distribution from scipy.stats with overloading of addition and multiplication.

Parameters

- **dtype** (str) – Data type of the distribution as a string. It must be one defined in scipy.stats
- **loc** (float) – The location of the distribution (corresponds to mean in Gaussian)
- **scale** (float) – The _scale of the distribution (corresponds to standard deviation in Gaussian)
- **multipliers** – An iterable of other distribution to multiply with this one
- **adders** – An iterable of other utility_priors to add to this one
- ****kwargs –

Examples

```python
>>> d2 = Distribution('uniform')
>>> print(d2.mean())
0.5

>>> try:
...     d = Distribution('something crazy')
... except ValueError as e:
...     print(str(e))
Unknown distribution something crazy
```

Attributes Summary

<table>
<thead>
<tr>
<th>loc</th>
<th>scale</th>
</tr>
</thead>
</table>

Methods Summary
around([value, range, uncertainty]) Generates a uniform distribution around the input value in the given range with given uncertainty

mean() rtype float

prob(val) Returns the probability for the given value

sample([size]) rtype ndarray

Attributes Documentation

loc
scale

Methods Documentation

classmethod around(value=0.5, range=(0.0, 1.0), uncertainty=0.5) Generates a uniform distribution around the input value in the given range with given uncertainty

Parameters

• value (float) – The value to generate the distribution around
• range (Tuple[float, float]) – The range of possible values
• uncertainty (float) – The uncertainty level required. 0.0 means no uncertainty and 1.0 means full uncertainty

Return type Distribution

Returns Distribution A uniform distribution around value with uncertainty (scale)

mean()

Return type float

prob(val) Returns the probability for the given value

Return type float

sample(size=1)

Return type ndarray

ConfigReader
class negmas.helpers.ConfigReader Bases: object

Methods Summary

from_config(config[, section, ...]) Creates an object of this class given the configuration info

read_config(config[, section]) Reads the configuration from a file or a dict and prepares it for parsing

7.1. negmas.helpers Module
Methods Documentation

classmethod from_config(config, section=None, ignore_children=True, try_parsing_children=True, scope=None)
Creates an object of this class given the configuration info

Parameters

• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params
• ignore_children (bool) – If true then children will be ignored and there will be a single return
• try_parsing_children (bool) – If true the children will first be parsed as ConfigReader classes if they are not types (e.g. int, str, float, Iterable[int|str|float]) (simple)–
• scope – The scope at which to evaluate any child classes. This MUST be passed as scope=globals() if you are
• any children that are to be parsed. (having)–

Returns An object of cls if ignore_children is True or a tuple with an object of cls and a dictionary with children that were not parsed.

Remarks:

• This function will return an object of its class after passing the key-value pairs found in the config to the init function.
• Requiring passing scope=globals() to this function is to get around the fact that in python eval() will be called with a globals dictionary based on the module in which the function is defined not called. This means that in general when eval() is called to create the children, it will not have access to the class definitions of these children (except if they happen to be imported in this file). To avoid this problem causing an undefined_name exception, the caller must pass her globals() as the scope.

classmethod read_config(config, section=None)
Reads the configuration from a file or a dict and prepares it for parsing

Parameters

• config (Union[str, dict]) – Either a file name or a dictionary
• section (Optional[str]) – A section in the file or a key in the dictionary to use for loading params

Returns A dict ready to be parsed by from_config

Remarks:
7.1.3 Class Inheritance Diagram

![Class Inheritance Diagram]

```
Enum          ReturnCause
              
Distribution

ConfigReader
```

7.2 `negmas.inout` Module

Defines import/export functionality

7.2.1 Functions

- `load_genius_domain`(`domain_file_name[, ...]`) Loads a genius domain, creates appropriate negotiators if necessary
- `load_genius_domain_from_folder`(`folder_name`) Loads a genius domain from a folder.
- `convert_genius_domain`(`src_domain_file_name, ...`) Loads a genius domain from a folder.
- `convert_genius_domain_from_folder`(`...`) Loads a genius domain from a folder.
- `find_domain_and_utility_files`(`folder_name`) Finds the domain and utility_function files in a folder
- `get_domain_issues`(`domain_file_name[, ...]`) Returns the issues of a given XML domain (Genius Format)
load_genius_domain

```
load_genius_domain(domain_file_name, utility_file_names=None, agent_factories=None, force_single_issue=False, force_numeric=False, cache_and_discretize_outcomes=False, max_n_outcomes=1000000.0, n_discretization=None, keep_issue_names=True, keep_value_names=True, normalize_utilities=True, n_steps=None, time_limit=180, max_n_agents=None, dynamic_entry=True, safe_parsing=False, ignore_reserved=False, ignore_discount=False)
```

Loads a genius domain, creates appropriate negotiators if necessary

**Parameters**

- `domain_file_name (str)`
- `utility_file_names (Optional[List[str]])`
- `agent_factories (Union[Callable[[], Negotiator], List[Callable[[], Negotiator]], None])`
- `force_single_issue`
- `cache_and_discretize_outcomes`
- `max_n_outcomes (int)`
- `n_discretization (Optional[int])`
- `keep_issue_names`
- `keep_value_names`
- `normalize_utilities`
- `n_steps`
- `time_limit`
- `max_n_agents`
- `dynamic_entry`
- `safe_parsing`
- `ignore_reserved`
- `ignore_discount`

**Returns**

A mechanism for the given issues - agent_info (List[Dict]): All Negotiator functions from the given file - issues Union[Issue, Dict[str, Issue], List[Issue]] : The issues

**Return type**

- mechanism (SAOMechanism)
load_genius_domain_from_folder

```python
negmas.inout.load_genius_domain_from_folder(folder_name, agent_factories=None, force_single_issue=False, force_numeric=False, cache_and_discretize_outcomes=False, max_n_outcomes=1000000, n_discretization=None, keep_issue_names=True, keep_value_names=True, normalize_utilities=True, n_steps=None, time_limit=60, max_n_agents=None, dynamic_entry=True, safe_parsing=False, ignore_reserved=False, ignore_discount=False)
```

Loads a genius domain from a folder. See `load_genius_domain` for more details.

**Parameters**

- `folder_name (str)` -
- `agent_factories (Union[Callable[[], Negotiator], List[Callable[[], Negotiator]], None])` -
- `force_single_issue` -
- `cache_and_discretize_outcomes` -
- `max_n_outcomes (int)` -
- `n_discretization (Optional[int])` -
- `keep_issue_names` -
- `keep_value_names` -
- `normalize_utilities` -
- `n_steps` -
- `time_limit` -
- `max_n_agents` -
- `dynamic_entry` -
- `safe_parsing` -
- `ignore_reserved` -
- `ignore_discount` -

**Returns** A mechanism for the given issues - agent_info (List[Dict]): All Negotiator functions from the given file - issues Union[Issue, Dict[str, Issue], List[Issue]]: The issues

**Return type**

- mechanism (SAOMechanism)

**Examples**

```python
>>> folder_name = pkg_resources.resource_filename('negmas', resource_name=˓→'tests/data/10issues')
>>> mechanism, negotiators, issues = load_genius_domain_from_folder(folder_name ˓→names=False, force_single_issue=False, keep_issue_names=False)
```
... , keep_value_names=False, normalize_
utilities=False)

>>> print(len(issues))
10

>>> print(len(negotiators))
2

>>> print([type(a['ufun']) for a in negotiators])
[<class 'negmas.utilities.HyperRectangleUtilityFunction'>, <class 'negmas."
utilities.HyperRectangleUtilityFunction'>]

>>> print(negotiators[0]['ufun'].outcome_ranges[0])
{1: (7.0, 9.0), 3: (2.0, 7.0), 5: (0.0, 8.0), 8: (0.0, 7.0)}

>>> print(negotiators[0]['ufun'].mappings[0])
97.0

>>> u = negotiators[0]['ufun']

>>> print(u(tuple([0.0] * len(issues))))
0.0

>>> print(u(tuple([0.5] * len(issues))))
186.0

Try loading and running a domain with predetermined agents:

>>> mechanism, agents, issues = load_genius_domain_from_folder(... pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop') ... , agent_factories=AspirationNegotiator ... , force_single_issue=True, keep_issue_names=False ... , keep_value_names=False) >>> state = mechanism.run() >>> state.agreement is not None True

>>> mechanism, negotiators, issues = load_genius_domain_from_folder(... pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop'))

>>> len(issues), len(negotiators)
(3, 2)

>>> [type(a['ufun']) for a in negotiators]
[<class 'negmas.utilities.LinearUtilityAggregationFunction'>, <class 'negmas."
utilities.LinearUtilityAggregationFunction'>]

>>> mechanism, negotiators, issues = load_genius_domain_from_folder(... pkg_resources.resource_filename('negmas', resource_name='tests/data/Laptop') ... , force_single_issue=True, keep_issue_names=False ... , keep_value_names=False)

>>> len(issues), len(negotiators)
(1, 2)

>>> [type(a['ufun']) for a in negotiators]
[<class 'negmas.utilities.MappingUtilityFunction'>, <class 'negmas.utilities."
MappingUtilityFunction']]

convert_genius_domain_from_folder

negmas.inout.convert_genius_domain_from_folder(src_folder_name, dst_folder_name, **kwargs)

Loads a genius domain from a folder. See load_genius_domain for more details.

Return type bool
convert_genius_domain

```python
def convert_genius_domain(src_domain_file_name, dst_domain_file_name, src_utility_file_names=None, dst_utility_file_names=None, force_single_issue=False, cache_and_discretize_outcomes=False, max_n_outcomes=1000000.0, n_discretization=None, keep_issue_names=True, keep_value_names=True, normalize_utilities=True, safe_parsing=False):
    # Return type: bool
```

find_domain_and_utility_files

```python
def find_domain_and_utility_files(folder_name):
    # Finds the domain and utility_function files in a folder
    # Return type: Tuple[str, List[str]]
```

get_domain_issues

```python
def get_domain_issues(domain_file_name, force_single_issue=False, max_n_outcomes=1000000.0, n_discretization=None, keep_issue_names=True, keep_value_names=True, safe_parsing=False):
    # Returns the issues of a given XML domain (Genius Format)
    # Parameters
    #   - domain_file_name (str)
    #   - force_single_issue
    #   - max_n_outcomes (int)
    #   - n_discretization (Optional[int])
    #   - keep_issue_names
    #   - keep_value_names
    #   - safe_parsing
    # Return type: Union[Dict[str, Issue], List[Issue]]
    # Returns: List or Dict of issues
```

### 7.3 negmas.java Module

Implements Java interoperability allowing parts of negmas to work smoothly with their Java counterparts in jnegmas

#### 7.3.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to_java(value[, add_type_field, ...])</code></td>
<td>Encodes the given value as nothing not more complex than simple dict of either dicts, lists or builtin numeric or string values</td>
</tr>
</tbody>
</table>

Continued on next page
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_java(d[, deep, remove_type_field, ...])</code></td>
<td>Decodes a dict coming from java recovering all objects in the way</td>
</tr>
<tr>
<td><code>jnegmas_bridge_is_running([port])</code></td>
<td>Checks whether a JNegMAS Bridge is running.</td>
</tr>
<tr>
<td><code>init_jnegmas_bridge([path, port])</code></td>
<td></td>
</tr>
<tr>
<td><code>jnegmas_connection([init, path, java_port, ...])</code></td>
<td>A connection to jnegmas that closes automatically</td>
</tr>
<tr>
<td><code>from_java(d[, deep, remove_type_field, ...])</code></td>
<td>Decodes a dict coming from java recovering all objects in the way</td>
</tr>
<tr>
<td><code>java_link(obj[, map])</code></td>
<td>Creates a link in java to the object given without copying it.</td>
</tr>
<tr>
<td><code>to_dict(value[, deep, add_type_field, camel])</code></td>
<td>Encodes the given value as nothing not more complex than simple dict of either dicts, lists or builtin numeric or string values</td>
</tr>
<tr>
<td><code>to_flat_dict(value[, deep])</code></td>
<td>Encodes the given value as a flat dictionary</td>
</tr>
</tbody>
</table>

**to_java**

`negmas.java.to_java(value, add_type_field=True, python_class_name=None)`

Encodes the given value as nothing not more complex than simple dict of either dicts, lists or builtin numeric or string values.

**Parameters**

- **value** – Any object
- **add_type_field** – If true, the `PYTHON_CLASS_IDENTIFIER` will be added with the python class field on it
- **python_class_name** (Optional[str]) – It given it overrides the class name written when `add_type_field` is True otherwise, the
  - name will be inferred as the `__class__` of value. (class)

**Remarks:**

- All iterables are converted to lists when `deep` is true.
- If the `value` object has a `to_java` member, it will be called to do the conversion, otherwise its `__dict__` or `__slots__` member will be used.

**See also:**

`from_java.PYTHON_CLASS_IDENTIFIER`

**from_java**

`negmas.java.from_java(d, deep=True, remove_type_field=True, fallback_class_name=None)`

Decodes a dict coming from java recovering all objects in the way

**Parameters**

- **d** (Any) – The value to be decoded. If it is not a dict, it is returned as is.
- **deep** – If true, decode recursively
- **remove_type_field** – If true the field called `PYTHON_CLASS_IDENTIFIER` will be removed if found.
- **fallback_class_name** (Optional[str]) – If given, it is used as the fall-back type if `'PYTHON_CLASS_IDENTIFIER' is not in the dict.

**Remarks:**
• If the object is not a dict or if it has no `PYTHON_CLASS_IDENTIFIER` field and no `fallback_class_name` is given, the input d is returned as it is. It will not even be copied.

See also:
`to_java`, `PYTHON_CLASS_IDENTIFIER`

`jnegas_bridge_is_running`

`negmas.java.jnegas_bridge_is_running(port=None)`
Checks whether a JNegMAS Bridge is running. This bridge is needed to use any objects in the jnegmas package

Remarks:
You can start a JNegMAS Bridge in at least two ways:
• execute the python function `init_jnegmas_bridge()` in this module
• run “negmas jnegmas” on the terminal

Return type `bool`

`init_jnegmas_bridge`

`negmas.java.init_jnegmas_bridge(path=None, port=0)`

`jnegas_connection`

`negmas.java.jnegas_connection(init=False, path=None, java_port=0, python_port=0, client_server=True, shutdown=True)`
A connection to jnegmas that closes automatically

`java_link`

`negmas.java.java_link(obj, map=None)`
Creates a link in java to the object given without copying it.

Parameters
• `obj` – The object for which to create a java shadow
• `map` – construction parameters
• `copyable` – If true, we will assume that the java object is PyCopyable otherwise PyConstructable. Only checked if
• `is not None(map)` –

Returns A java object. Cannot be used directly in python but can be used as an argument to a call to of a java object.

`to_dict`

`negmas.java.to_dict(value, deep=True, add_type_field=True, camel=True)`
Encodes the given value as nothing not more complex than simple dict of either dicts, lists or builtin numeric or string values

Parameters
• `value` – Any object
• **deep** – Whether we should go deep in the encoding or do a shallow encoding

• **add_type_field** – Whether to add a type field. If True, A field named `PYTHON_CLASS_IDENTIFIER` will be added

• **the type of value** *(giving)* –

• **camel** – Convert to camel_case if True

Remarks:

• All iterables are converted to lists when `deep` is true.

• If the `value` object has a `to_java` member, it will be called to do the conversion, otherwise its `_dict_` or `_slots_` member will be used.

See also:

`from_java`, `PYTHON_CLASS_IDENTIFIER`

---

### to_flat_dict

```python
negmas.java.to_flat_dict(value, deep=True)
```

Encodes the given value as a flat dictionary

**Parameters**

- `value`
- `deep`

**Returns:**

Return type: `Dict[str, Any]`

---

### 7.3.2 Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaCallerMixin</td>
<td>A mixin to enable calling a java counterpart.</td>
</tr>
<tr>
<td>JNegmasGateway</td>
<td></td>
</tr>
</tbody>
</table>

---

### JavaCallerMixin

```python
class negmas.java.JavaCallerMixin
```

**Bases:** `object`

A mixin to enable calling a java counterpart. This mixin can ONLY be used with a `NamedObject` because it uses its id property.

Other than inheriting this mixin, you should call its `init_java_bridge` in your `__init__` (or whenever your object is initialized and you need to create the Java counterpart). You should then implement all your functions as calls to `java_object`.

If for example you have a function `do_this`, you can just define it as follows:

Notice that you cannot use named arguments when calling the function in `java_object` and that the names are converted to camelCase instead of snake_case. Moreover, property `x` will be implemented as a pair `getX, setX` on the Java side.

If your class needs just to call the corresponding java object but is never called back from it then you are done after inheriting from this mixin.

If your objects need to be called from the java counterpart, then you need to add the following to your class definition:

This assumes that your class is named `ClassName` and that there is an interface called `ClassName`
defined in jnegmas that has the same public interface as your class (or whatever part of it to be called from Java).

**Methods Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_dict(java_object, *args, **kwargs)</code></td>
<td>Creates a Python object representing the corresponding Java object</td>
</tr>
<tr>
<td><code>init_java_bridge(java_object, java_class_name)</code></td>
<td>Initializes a connection to the java bridge creating a member called java_object that can be used to access the counterpart object in Java</td>
</tr>
</tbody>
</table>

**Methods Documentation**

**classmethod from_dict (java_object, *args, **kwargs)**

Creates a Python object representing the corresponding Java object

**init_java_bridge (java_object, java_class_name, auto_load_java=False, python_shadow_object=None)**

Initializes a connection to the Java bridge creating a member called java_object that can be used to access the counterpart object in Java

**Parameters**

- `java_object` – A java object that already exists of the correct type. If given no new objects will be created
- `java_class_name` (str) – The type of the Java object to be created
- `auto_load_java` (bool) – When true, a JVM will be automatically created (if one is not available)
- `python_shadow_object` (Optional[Any]) – A python object to shadow the java object. The object will just call the corresponding

**Remarks:**

- Sets a member called java_object that can be used to access the corresponding Java object crated
- If python_shadow_object is given, it must be an object of a type that has an internal class called Java which has a single member called ‘implements’ which is a list of one string element representing the Java interface being implemented (it must be either jnegmas.PyCallable or an extension of it).

**JNegmasGateway**

**class negmas.java.JNegmasGateway**

Bases: object

**Attributes Summary**

- `DEFAULT_JAVA_PORT`
- `DEFAULT_PYTHON_PORT`
- `gateway`

**Methods Summary**

7.3. `negmas.java Module`
### Methods Documentation

#### connect

<table>
<thead>
<tr>
<th>Method: <code>connect</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature: <code>(java_port=None, python_port=None, auto_load_java=False, client_server=True)</code></td>
</tr>
<tr>
<td>Description: Connects to JNegMAS.</td>
</tr>
<tr>
<td>Return type: <code>bool</code></td>
</tr>
</tbody>
</table>

#### is_running

<table>
<thead>
<tr>
<th>Method: <code>is_running</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature: <code>(port)</code></td>
</tr>
<tr>
<td>Description: Checks whether a JNegMAS Bridge is running. This bridge is needed to use any objects in the jnegmas package.</td>
</tr>
<tr>
<td>Remarks: You can start a JNegMAS Bridge in at least two ways:</td>
</tr>
<tr>
<td>- execute the python function <code>init_jnegmas_bridge()</code> in this module</td>
</tr>
<tr>
<td>- run “negmas jnegmas” on the terminal</td>
</tr>
</tbody>
</table>

#### shutdown

<table>
<thead>
<tr>
<th>Method: <code>shutdown</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature: <code>()</code></td>
</tr>
<tr>
<td>Description: Closes a connection to JNegMAS.</td>
</tr>
</tbody>
</table>

#### start_java_side

<table>
<thead>
<tr>
<th>Method: <code>start_java_side</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature: <code>(path=None, java_port=0)</code></td>
</tr>
<tr>
<td>Description: Initializes a connection to JNegMAS.</td>
</tr>
<tr>
<td>Examples:</td>
</tr>
<tr>
<td>```python</td>
</tr>
<tr>
<td># &gt;&gt;&gt; start_java_side(port=35337) # &gt;&gt;&gt; a = JNegmasGateway.do_nothing_manager() # &gt;&gt;&gt; a.java_uuid.startswith('jnegmas') # True # &gt;&gt;&gt; len(a.java_uuid)- len(a.java_class_name) == 36 # length of UUID # True</td>
</tr>
<tr>
<td>Return type: <code>None</code></td>
</tr>
</tbody>
</table>
| ```
7.3.3 Class Inheritance Diagram

JavaCallerMixin

JNegmasGateway

7.4 negmas.tournaments Module

Tournament generation and management.

7.4.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tournament(competitors, config_generator, ...)</td>
<td>Runs a tournament</td>
</tr>
<tr>
<td>run_world(world_params[, dry_run, ...])</td>
<td>Runs a world and returns stats.</td>
</tr>
<tr>
<td>process_world_run(run_id, results, [...])</td>
<td>Generates a data-frame with the results of this world run</td>
</tr>
<tr>
<td>evaluate_tournament(tournament_path, [...])</td>
<td>Evaluates the results of a tournament</td>
</tr>
<tr>
<td>combine_tournaments(sources[, dest, verbose])</td>
<td>Combines results of several tournament runs in the destination path</td>
</tr>
<tr>
<td>combine_tournament_stats(sources[, dest, ...])</td>
<td>Combines statistical results of several tournament runs in the destination path</td>
</tr>
<tr>
<td>create_tournament(competitors, [...])</td>
<td>Runs a tournament</td>
</tr>
<tr>
<td>run_tournament(tournament_path[, ...])</td>
<td>Runs a tournament</td>
</tr>
</tbody>
</table>
tournament

tournament(competitors, config_generator, config_assigner, world_generator, score_calculator, competitor_params=None, n_competitors_per_world=None, round_robin=False, stage_winners_fraction=0.5, agent_names_reveal_type=False, n_agents_per_competitor=1, n_configs=10, max_worlds_per_config=100, n_runs_per_world=5, max_n_configs=None, n_runs_per_config=None, tournament_path='./logs/tournaments', total_timeout=None, parallelism='parallel', scheduler_ip=None, scheduler_port=None, tournament_progress_callback=None, world_progress_callback=None, non_competitors=None, non_competitor_params=None, name=None, verbose=False, configs_only=False, compact=False, print_exceptions=True, **kwargs)

Runs a tournament

Parameters

- **name** *(Optional[str])* – Tournament name
- **config_generator** *(ConfigGenerator)* – Used to generate unique configs that will be used to evaluate competitors
- **config_assigner** *(ConfigAssigner)* – Used to generate assignments of competitors to the configs created by the config_generator
- **world_generator** *(WorldGenerator)* – A function to generate worlds for the tournament that follows the assignments made by the config_assigner
- **score_calculator** *(Callable[[List[World], Dict[str, Any], bool, WorldRunResults]], Dict[str, Any])* – A function for calculating the score of all agents in a world. After it finishes running. The second parameter is a dict describing any scoring context that may have been added by the world config generator or assigner. The third parameter is a boolean specifying whether this is a dry_run. For dry runs, scores are not expected but names and types should exist in the returned WorldRunResults.
- **competitors** *(Sequence[Union[str, Type[Agent]]])* – A list of class names for the competitors
- **competitor_params** *(Optional[Sequence[Dict[str, Any]]])* – A list of competitor parameters (used to initialize the competitors).
- **n_competitors_per_world** *(Optional[int])* – The number of competitors allowed in every world. It must be >= 1 and <= len(competitors) or None.
  - If None or len(competitors), then all competitors will exist in every world.
  - If 1, then each world will have one competitor
- **round_robin** *(bool)* – Only effective if 1 < n_competitors_per_world < len(competitors). If True, all combinations will be tried otherwise n_competitors_per_world must divide len(competitors) and every competitor appears only in one set.
- **stage_winners_fraction** *(float)* – in [0, 1). Fraction of agents to go to the next stage at every stage. If zero, and round_robin, it becomes a single stage competition.
- **agent_names_reveal_type** – If true then the type of an agent should be readable in its name (most likely at its beginning).
- **n_configs** *(int)* – The number of different world configs (up to competitor assignment) to be generated.
- **max_worlds_per_config** (*int*) – The maximum number of worlds to run per config. If None, then all possible assignments of competitors within each config will be tried (all permutations).

- **n_runs_per_world** (*int*) – Number of runs per world. All of these world runs will have identical competitor assignment and identical world configuration.

- **n_agents_per_competitor** – The number of agents of each competing type to be instantiated in the world.

- **max_n_configs** ([Optional]int]) – [Depricated] The number of configs to use (it is replaced by separately setting n_config and max_worlds_per_config)

- **n_runs_per_config** ([Optional]int]) – [Depricated] The number of runs (simulation) for every config. It is replaced by n_runs_per_world

- **total_timeout** ([Optional]int]) – Total timeout for the complete process

- **tournament_path** (*str*) – Path at which to store all results. A new folder with the name of the tournament will be created at this path. A scores.csv file will keep the scores and logs folder will keep detailed logs

- **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed! For parallel, you can add the fraction of CPUs to use after a colon (e.g. parallel:0.5 to use half of the CPU in the machine). By defaults parallel uses all CPUs in the machine

- **scheduler_port** ([Optional][str]) – Port of the dask scheduler if parallelism is dask, dist, or distributed

- **scheduler_ip** ([Optional][str]) – IP Address of the dask scheduler if parallelism is dask, dist, or distributed

- **world_progress_callback** ([Optional][Callable[[Optional][World]], None]]) – A function to be called after every step of every world run (only allowed for serial and parallel evaluation and should be used with cautious).

- **tournament_progress_callback** ([Optional][Callable[[Optional][WorldRunResults], int, int], None]]) – A function to be called with WorldRunResults after each world finished processing

- **non_competitors** ([Optional][Tuple[Union[str, Any]]]) – A list of agent types that will not be competing in the sabotage competition but will exist in the world

- **non_competitor_params** ([Optional][Tuple[Dict[str, Any]]]) – paramters of non competitor agents

- **verbose** (*bool*) – Verbosity

- **configs_only** (*bool*) – If true, a config file for each

- **compact** (*bool*) – If true, compact logs will be created and effort will be made to reduce the memory footprint

- **kwargs** – Arguments to pass to the config_generator function

Return type: Union[TournamentResults, PathLike]

Returns: TournamentResults The results of the tournament or a PathLike giving the location where configs were saved

**run_world**

negmas.tournaments.run_world(world_params, dry_run=False, save_world_stats=True)

Runs a world and returns stats. This function is designed to be used with distributed systems like dask.

**Parameters**
• `world_params (dict)` – World info dict. See remarks for its parameters
• `dry_run (bool)` – If true, the world will not be run. Only configs will be saved
• `save_world_stats (bool)` – If true, saves individual world stats

Remarks:
The `world_params` dict should have the following members:
• `name`: world name [Defaults to random]
• `log_file_name`: file name to store the world log [Defaults to random]
• `__dir_name`: directory to store the world stats [Defaults to random]
• `__world_generator`: full name of the world generator function (including its module) [Required]
• `__score_calculator`: full name of the score calculator function [Required]
• `__tournament_name`: name of the tournament [Defaults to random]
• `others`: values of all other keys are passed to the world generator as kwargs

Return type: `Tuple[str, WorldRunResults]`

`process_world_run`

`negmas.tournaments.process_world_run(run_id, results, tournament_name, save_world_stats=True)`
Generates a data-frame with the results of this world run

Parameters
• `run_id (str)` – The ID of this run (should be unique per tournament)
• `results (WorldRunResults)` – Results of the world run
• `tournament_name (str)` – tournament name
• `save_world_stats (bool)` – If True, it will be assumed that world stats are saved

Return type: `List[Dict[str, Any]]`

Returns: A pandas DataFrame with agent_name, agent_type, score, log_file, world, and stats_folder columns

`evaluate_tournament`

`negmas.tournaments.evaluate_tournament(tournament_path, scores=None, stats=None, metric='mean', verbose=False, recursive=False)`
Evaluates the results of a tournament

Parameters
• `tournament_path (Union[str, PathLike, None])` – Path to save the results to. If scores is not given, it is also used as the source of scores. Pass None to avoid saving the results to disk.
• `scores (Optional[DataFrame])` – Optionally the scores of all agents in all world runs. If not given they will be read from the file scores.csv in `tournament_path`
• `stats (Optional[DataFrame])` – Optionally the stats of all world runs. If not given they will be read from the file stats.csv in `tournament_path`
• **metric** (Union[str, Callable[[DataFrame], float]]) – The metric used for evaluation. Possibilities are: mean, median, std, var, sum or a callable that receives a pandas data-frame and returns a float.

• **verbose** (bool) – If true, the winners will be printed

• **recursive** (bool) – If true, ALL scores.csv files in all subdirectories of the given tournament_path will be combined

• **independent_test** (#) – True if you want an independent t-test

Returns:

Return type **TournamentResults**

**combine_tournaments**

negmas.tournaments.combine_tournaments(sources, dest=None, verbose=False)

Combines results of several tournament runs in the destination path.

Return type **DataFrame**

**combine_tournament_stats**

negmas.tournaments.combine_tournament_stats(sources, dest=None, verbose=False)

Combines statistical results of several tournament runs in the destination path.

Return type **DataFrame**

**create_tournament**

negmas.tournaments.create_tournament(competitors, config_generator, config_assigner, world_generator, score_calculator, competitor_params=None, n_competitors_per_world=None, round_robin=True, agent_names_reveal_type=False, n_agents_per_competitor=1, n_configs=10, max_worlds_per_config=100, n_runs_per_world=5, max_n_configs=None, n_runs_per_config=None, base_tournament_path='./logs/tournaments', total_timeout=None, parallelism='parallel', scheduler_ip=None, scheduler_port=None, non_competitors=None, non_competitor_params=None, name=None, verbose=False, compact=False, **kwargs)

Runs a tournament

Parameters

• **name** (Optional[str]) – Tournament name

• **config_generator** (ConfigGenerator) – Used to generate unique configs that will be used to evaluate competitors

• **config_assigner** (ConfigAssigner) – Used to generate assignments of competitors to the configs created by the config_generator

• **world_generator** (WorldGenerator) – A functions to generate worlds for the tournament that follows the assignments made by the config_assigner
• **score_calculator** ([Callable[List[World], Dict[str, Any], bool], WorldRunResults]) – A function for calculating the score of all agents in a world **After it finishes running**. The second parameter is a dict describing any scoring context that may have been added by the world config generator or assigner. The third parameter is a boolean specifying whether this is a dry_run. For dry runs, scores are not expected but names and types should exist in the returned `WorldRunResults`.

• **competitors** ([Sequence[Union[str, Type[Agent]]]]) – A list of class names for the competitors

• **competitor_params** ([Optional][Sequence[Dict[str, Any]]]) – A list of competitor parameters (used to initialize the competitors).

• **n_competitors_per_world** ([Optional][int]) – The number of competitors allowed in every world. It must be >= 1 and <= len(competitors) or None.
  
  – If None or len(competitors), then all competitors will exist in every world.
  
  – If 1, then each world will have one competitor

• **round_robin** (bool) – Only effective if 1 < n_competitors_per_world < len(competitors). If True, all combinations will be tried otherwise n_competitors_per_world must divide len(competitors) and every competitor appears only in one set.

• **agent_names_reveal_type** – If true then the type of an agent should be readable in its name (most likely at its beginning).

• **n_configs** (int) – The number of different world configs (up to competitor assignment) to be generated.

• **max_worlds_per_config** (int) – The maximum number of worlds to run per config. If None, then all possible assignments of competitors within each config will be tried (all permutations).

• **n_runs_per_world** (int) – Number of runs per world. All of these world runs will have identical competitor assignment and identical world configuration.

• **n_agents_per_competitor** – The number of agents of each competing type to be instantiated in the world.

• **max_n_configs** ([Optional][int]) – [Deprecated] The number of configs to use (it is replaced by separately setting n_config and max_worlds_per_config)

• **n_runs_per_config** ([Optional][int]) – [Deprecated] The number of runs (simulation) for every config. It is replaced by n_runs_per_world

• **total_timeout** ([Optional][int]) – Total timeout for the complete process

• **base_tournament_path** (str) – Path at which to store all results. A new folder with the name of the tournament will be created at this path. A scores.csv file will keep the scores and logs folder will keep detailed logs

• **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed! For parallel, you can add the fraction of CPUs to use after a colon (e.g. parallel:0.5 to use half of the CPU in the machine). By defaults parallel uses all CPUs in the machine

• **scheduler_port** ([Optional][str]) – Port of the dask scheduler if parallelism is `dask`, `dist`, or `distributed`

• **scheduler_ip** ([Optional][str]) – IP Address of the dask scheduler if parallelism is `dask`, `dist`, or `distributed`

• **non_competitors** ([Optional][Tuple[Union[str, Any]]]) – A list of agent types that will not be competing in the sabotage competition but will exist in the world
run_tournament

def run_tournament(tournament_path, world_generator=None, score_calculator=None, total_timeout=None, parallelism='parallel', scheduler_ip=None, scheduler_port=None, tournament_progress_callback=None, world_progress_callback=None, verbose=False, compact=None, print_exceptions=True):

    Runs a tournament

    Parameters

    • **tournament_path** (Union[str, PathLike]) – Path at which configs of this tournament are stored

    • **world_generator** (Optional[WorldGenerator]) – A function to generate worlds for the tournament that follows the assignments made by the config_assigner

    • **score_calculator** (Optional[Callable[[List[World], Dict[str, Any], bool], WorldRunResults]]) – A function for calculating the score of all agents in a world. After it finishes running. The second parameter is a dict describing any scoring context that may have been added by the world config generator or assigner. The third parameter is a boolean specifying whether this is a dry_run. For dry runs, scores are not expected but names and types should exist in the returned WorldRunResults.

    • **total_timeout** (Optional[int]) – Total timeout for the complete process

    • **parallelism** – Type of parallelism. Can be ‘serial’ for serial, ‘parallel’ for parallel and ‘distributed’ for distributed! For parallel, you can add the fraction of CPUs to use after a colon (e.g. parallel:0.5 to use half of the CPU in the machine). By defaults parallel uses all CPUs in the machine

    • **scheduler_port** (Optional[str]) – Port of the dask scheduler if parallelism is dask, dist, or distributed

    • **scheduler_ip** (Optional[str]) – IP Address of the dask scheduler if parallelism is dask, dist, or distributed

    • **world_progress_callback** (Optional[Callable[[Optional[WorldRunResults], int, int], None]]) – A function to be called after every step of every world run (only allowed for serial and parallel evaluation and should be used with cautious).

    • **tournament_progress_callback** (Optional[Callable[[Optional[WorldRunResults], int, int], None]]) – A function to be called with WorldRunResults after each world finished processing

    • **verbose** (bool) – Verbosity

    • **compact** (Optional[bool]) – If true, compact logs will be created and effort will be made to reduce the memory footprint

    Returns

    The path at which tournament configs are stored
• **print_exceptions** *(bool)* – If true, exceptions encountered during world simulation will be printed to stdout

**Return type** None

### 7.4.2 Classes

**WorldGenerator**

```python
def WorldGenerator(*args, **kwargs):
    Bases: typing_extensions.Protocol
    A callback-protocol specifying the signature of a world generator function that can be passed to `tournament`

    Parameters** kwags** – key-value pairs of arguments.

    See also:
    `tournament`

    Methods Summary

    `__call__`(**kwargs) Call self as a function.

    **Methods Documentation**

    `__call__`(**kwargs)
    Call self as a function.

    **Return type** `World`
```

**WorldRunResults**

```python
def WorldRunResults(world_names, log_file_names):
    Bases: object
    Results of a world run
```

**TournamentResults**

```python
def TournamentResults(scores, total_scores, ..., ttest=None, ktest=None, stats=None, agg_stats=None, score_stats=None)
    Bases: object
```
Attributes Summary

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>agg_stats</td>
</tr>
<tr>
<td>kstest</td>
</tr>
<tr>
<td>score_stats</td>
</tr>
<tr>
<td>stats</td>
</tr>
<tr>
<td>ttest</td>
</tr>
</tbody>
</table>

Attributes Documentation

`agg_stats = None`
`kstest = None`
`score_stats = None`
`stats = None`
`ttest = None`

7.4.3 Class Inheritance Diagram

```
Protocol
WorldGenerator

TournamentResults

WorldRunResults
```

Protocol ─────────── WorldGenerator
When installing NegMAS through the pip command, you get one command line tool that can be used to aid your development and testing. This tool provides a unified interface to all negmas commands.

The set of supported commands are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>genius</td>
<td>Run a Genius Bridge. This bridge allows you to use GeniusNegotiator agents. Please notice that this command by-default runs in the foreground preventing further input to the terminal.</td>
</tr>
<tr>
<td>genius-setup</td>
<td>Downloads the genius bridge and updates your settings.</td>
</tr>
<tr>
<td>jnegmas</td>
<td>Start the bridge to JNegMAS (to use Java agents in worlds)</td>
</tr>
<tr>
<td>jnegmas-setup</td>
<td>Downloads jnegmas and updates your settings</td>
</tr>
<tr>
<td>scml</td>
<td>Runs an SCML world</td>
</tr>
<tr>
<td>tournament</td>
<td>Runs a tournament</td>
</tr>
<tr>
<td>version</td>
<td>Prints NegMAS version</td>
</tr>
</tbody>
</table>

### 8.1 Genius Bridge (negmas genius)

The command `genius` can be used to start a JVM running the Genius platform allowing GeniusNegotiator objects to interact with existing GENIUS agents (Thanks for Tim Baarslag Lead Developer of GENIUS for allowing us to ship it within NegMAS).

You can get help on this tool by running:

```
$ negmas genius --help
```

This tool supports the following optional arguments:
8.2 SCML World Runner (negmas scml)

The SCML World Runner command (scml) runs an SCML world with default factory managers and reports the results of this run.

You can get help on this tool by running:

```
$ negmas scml --help
```

These are the optional arguments of this tool:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p/-path TEXT</td>
<td>Path to genius-8.0.4.jar with embedded NegLoader [OPTIONAL]</td>
</tr>
<tr>
<td>-r/-port INTEGER</td>
<td>Port to run the NegLoader on. Pass 0 for the default value [OPTIONAL]</td>
</tr>
<tr>
<td>--force/–no-force</td>
<td>Force trial even if an earlier instance exists [OPTIONAL]</td>
</tr>
<tr>
<td>--help</td>
<td>Show help message and exit.</td>
</tr>
<tr>
<td>–steps INTEGER</td>
<td>Number of steps. [default: 120]</td>
</tr>
<tr>
<td>–levels INTEGER</td>
<td>Number of intermediate production levels (processes). -1 means a single product and no factories. [default: 3]</td>
</tr>
<tr>
<td>–neg-speedup INTEGER</td>
<td>Negotiation Speedup. [default: 21]</td>
</tr>
<tr>
<td>–negotiator TEXT</td>
<td>Negotiator type to use for builtin agents. [default: negmas.sao.AspirationNegotiator]</td>
</tr>
<tr>
<td>–min-consumption INTEGER</td>
<td>The minimum number of units consumed by each consumer at every time-step. [default: 3]</td>
</tr>
<tr>
<td>–max-consumption INTEGER</td>
<td>The maximum number of units consumed by each consumer at every time-step. [default: 5]</td>
</tr>
<tr>
<td>–agents INTEGER</td>
<td>Number of agents (miners/negmas.consumers) per production level [default: 5]</td>
</tr>
<tr>
<td>–horizon INTEGER</td>
<td>Consumption horizon. [default: 20]</td>
</tr>
<tr>
<td>–transport INTEGER</td>
<td>Transportation Delay. [default: 0]</td>
</tr>
<tr>
<td>–time INTEGER</td>
<td>Total time limit. [default: 5400]</td>
</tr>
<tr>
<td>–neg-time INTEGER</td>
<td>Time limit per single negotiation [default: 240]</td>
</tr>
<tr>
<td>–neg-steps INTEGER</td>
<td>Number of rounds per single negotiation [default: 20]</td>
</tr>
<tr>
<td>–sign INTEGER</td>
<td>The default delay between contract conclusion and signing [default: 1]</td>
</tr>
<tr>
<td>–guaranteed TEXT</td>
<td>Whether to only sign contracts that are guaranteed not to cause breaches [default: False]</td>
</tr>
<tr>
<td>–lines INTEGER</td>
<td>The number of lines per factory [default: 10]</td>
</tr>
<tr>
<td>–retrials INTEGER</td>
<td>The number of times an agent re-tries on failed negotiations [default: 5]</td>
</tr>
<tr>
<td>–use-consumer TEXT</td>
<td>Use internal consumer object in factory managers [default: True]</td>
</tr>
<tr>
<td>–max-insurance INTEGER</td>
<td>Use insurance against partner in factory managers up to this premium [default: 100]</td>
</tr>
<tr>
<td>–riskiness FLOAT</td>
<td>How risky is the default factory manager [default: 0.0]</td>
</tr>
<tr>
<td>–log TEXT</td>
<td>Default location to save logs (A folder will be created under it) [default: ~/negmas/logs]</td>
</tr>
<tr>
<td>–compact / –debug</td>
<td>If –compact, effort is exerted to reduce the memory footprint whichincludes reducing logs dramatically. [default: –compact]</td>
</tr>
<tr>
<td>–log-ufuns</td>
<td>If given, ufuns are logged [default: False] Only used if –debug is given</td>
</tr>
<tr>
<td>–log-negs</td>
<td>If given, all negotiations and their offers are logged [default: False]</td>
</tr>
<tr>
<td>–config FILE-NAME</td>
<td>configuration file name. If given all of the parameters given above can be entered in this file instead of being inputed on the command line.</td>
</tr>
<tr>
<td>–help</td>
<td>Show help message and exit.</td>
</tr>
</tbody>
</table>
Upon completion, a complete log and several statistics are saved in a new folder under the log folder location specified by the `--log` argument (default is negmas/logs under the HOME directory). To avoid over-writing earlier results, a new folder will be created for each run named by the current date and time (within an scml folder). The folder will contain the following files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Format</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_contracts.csv</td>
<td>CSV</td>
<td>A record of all contracts [filled only if --debug is specified]</td>
</tr>
<tr>
<td>contracts_full_info.csv</td>
<td>CSV</td>
<td>A record of all contracts with added information about the CFPs [filled only if --debug is specified]</td>
</tr>
<tr>
<td>cancelled_contracts.csv</td>
<td>CSV</td>
<td>Contracts that were cancelled because one partner refused to sign it [filled only if --debug is specified]</td>
</tr>
<tr>
<td>signed_contracts.csv</td>
<td>CSV</td>
<td>Contracts that were actually signed</td>
</tr>
<tr>
<td>negotiations.csv</td>
<td>CSV</td>
<td>A record of all negotiations [filled only if --debug is specified]</td>
</tr>
<tr>
<td>breaches.csv</td>
<td>CSV</td>
<td>A record of all breaches</td>
</tr>
<tr>
<td>stats.csv</td>
<td>CSV</td>
<td>Helpful statistics about the state of the world at every timestep (e.g. N. negotiations, N. Contracts Executed, etc) in CSV format</td>
</tr>
<tr>
<td>stats.json</td>
<td>JSON</td>
<td>Helpful statistics about the state of the world at every timestep (e.g. N. negotiations, N. Contracts Executed, etc) in JSON format</td>
</tr>
<tr>
<td>params.json</td>
<td>JSON</td>
<td>The arguments used to run the world</td>
</tr>
<tr>
<td>logs.txt</td>
<td>TXT</td>
<td>A log file giving details of most important events during the simulation [filled only if --debug is specified]</td>
</tr>
<tr>
<td>negotiation_info.csv</td>
<td>CSV</td>
<td>Negotiation information for all negotiation session logged (only if --log-negs is given).</td>
</tr>
<tr>
<td>negotiations</td>
<td>Folder</td>
<td>A folder containing a file for each negotiation giving all offers exchanged (only if --log-negs is given).</td>
</tr>
</tbody>
</table>

### 8.3 Tournament Command (negmas tournament)

The **Tournament command** (tournament) allows you to run a tournament between different agents in some world and compare their relative performance. The tool is general enough to support several world types but currently only the ANAC 2019 SCML (anac2019) configuration is supported.

You can get help on this tool by running:

```
$ negmas tournament --help
```

These are the optional arguments of this tool:
## Argument

<table>
<thead>
<tr>
<th>Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n, --name TEXT</td>
<td>The name of the tournament. The special value “random” will result in a random name [default: random]</td>
</tr>
<tr>
<td>-s, --steps INTEGER</td>
<td>Number of steps. [default: 60]</td>
</tr>
<tr>
<td>-f, --config TEXT</td>
<td>The config to use. Default is ANAC 2019 [default: anac2019]</td>
</tr>
<tr>
<td>-t, --timeout INTEGER</td>
<td>Timeout after the given number of seconds (0 for infinite) [default: 0]</td>
</tr>
<tr>
<td>--runs INTEGER</td>
<td>Number of runs for each configuration [default: 5]</td>
</tr>
<tr>
<td>--max-runs INTEGER</td>
<td>Maximum total number of runs. Zero or negative numbers mean no limit [default: -1]</td>
</tr>
<tr>
<td>--configs INTEGER</td>
<td>Number of unique configurations to generate. [default: 5]</td>
</tr>
<tr>
<td>--factories INTEGER</td>
<td>Minimum numbers of factories to have per level. [default: 5]</td>
</tr>
<tr>
<td>--competitors TEXT</td>
<td>A semicolon (;) separated list of agent types to use for the competition. [default: negmas.apps.scml.DoNothingFactoryManager; negmas.apps.scml.GreedyFactoryManager]</td>
</tr>
<tr>
<td>--jcompetitors / --java-competitors</td>
<td>A semicolon (;) separated list of agent types to use for the competition.</td>
</tr>
<tr>
<td>--parallel / --serial</td>
<td>Run a parallel/serial tournament on a single machine [default: True]</td>
</tr>
<tr>
<td>--distributed / --single-machine</td>
<td>Run a distributed tournament using dask [default: False]</td>
</tr>
<tr>
<td>-i, --log TEXT</td>
<td>Default location to save logs (A folder will be created under it) [default: ~/negmas/logs/tournaments]</td>
</tr>
<tr>
<td>--verbosity INTEGER</td>
<td>Verbosity level (from 0 == silent to 1 == world progress) [default: 1]</td>
</tr>
<tr>
<td>--configs-only / --run</td>
<td>configs_only [default: False]</td>
</tr>
<tr>
<td>--reveal-names / --hidden-names</td>
<td>Reveal agent names (should be used only for debugging) [default: False]</td>
</tr>
<tr>
<td>--ip TEXT</td>
<td>The IP address for a dask scheduler to run the distributed tournament. Effective only if --distributed [default: 127.0.0.1]</td>
</tr>
<tr>
<td>--port INTEGER</td>
<td>The IP port number a dask scheduler to run the distributed tournament. Effective only if --distributed [default: 8786]</td>
</tr>
<tr>
<td>--compact / --debug</td>
<td>If --compact, effort is exerted to reduce the memory footprint which includes reducing logs dramatically. [default: --compact]</td>
</tr>
<tr>
<td>--log-ufuns</td>
<td>If given, ufuns are logged [default: False] Only used if --debug is given</td>
</tr>
<tr>
<td>--log-negs</td>
<td>If given, all negotiations and their offers are logged. Only used if --debug is given [default: False]</td>
</tr>
<tr>
<td>--config FILENAME</td>
<td>Configuration file name. If given all of the parameters given above can be entered in this file instead of being inputed on the command line.</td>
</tr>
<tr>
<td>--help</td>
<td>Show help message and exit.</td>
</tr>
</tbody>
</table>

Upon completion, a complete log and several statistics are saved in a new folder under the log folder location specified by the --log argument (default is negmas/logs/tournaments under the HOME directory). To avoid over-writing earlier results, a new folder will be created for each run named by the current date and time. The folder will contain the following files:
<table>
<thead>
<tr>
<th>File/Folder Name</th>
<th>Format</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>configs</td>
<td>FOLDER</td>
<td>Contains one json file for each world run tried during the tournament. You can re-run this world using <code>run_world</code> function in the <code>tournament</code> module.</td>
</tr>
<tr>
<td>params.json</td>
<td>JSON</td>
<td>The parameters used to create this tournament</td>
</tr>
<tr>
<td>base_configs.json</td>
<td>JSON</td>
<td>The base configurations used in the tournament (without agent/factory assignments).</td>
</tr>
<tr>
<td>assigned_configs.json</td>
<td>JSON</td>
<td>The configurations used after assigning factories to managers</td>
</tr>
<tr>
<td>scores.csv</td>
<td>CSV</td>
<td>Scores of every agent in every world</td>
</tr>
<tr>
<td>total_scores.csv</td>
<td>CSV</td>
<td>Scores of every agent type averaged over all runs</td>
</tr>
<tr>
<td>winners.csv</td>
<td>CSV</td>
<td>Winner types and their average scores</td>
</tr>
<tr>
<td>ttest.csv</td>
<td>CSV</td>
<td>Results of a factorial TTEST comparing the performance of all agent types</td>
</tr>
</tbody>
</table>

Other than these files, a folder with the same number as the corresponding config file in the `configs` folder, keeps full statistics/log of every world but only if `--debug` is specified (see the SCML World Runner section for the contents of this folder.)
Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

9.1 Types of Contributions

9.1.1 Report Bugs


If you are reporting a bug, please include:

• Your operating system name and version.
• Any details about your local setup that might be helpful in troubleshooting.
• Detailed steps to reproduce the bug.

9.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

9.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

9.1.4 Write Documentation

negmas could always use more documentation, whether as part of the official negmas docs, in docstrings, or even on the web in blog posts, articles, and such.
9.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/yasserfarouk/negmas/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome ;)

9.2 Get Started!

Ready to contribute? Here’s how to set up negmas for local development.

1. Fork the negmas repo on GitHub.
2. Clone your fork locally:

   $ git clone git@github.com:your_name_here/negmas.git

3. Install your local copy into a virtualenv. This is how you set up your fork for local development (assuming you are using poetry):

   $ python -m venv venv
   $ source venv/bin/activate
   $ poetry install

4. Create a branch for local development:

   $ git checkout -b name-of-your-bugfix-or-feature

   Now you can make your changes locally.

5. When you’re done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   $ flake8 negmas tests
   $ python setup.py test or py.test
   $ tox

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   $ git add .
   $ git delete_bookmark -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature

7. Submit a pull request through the GitHub website.

9.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, and 3.7. Check https://travis-ci.org/yasserfarouk/negmas/pull_requests and make sure that the tests pass for all supported Python versions.

9.4 Tips

To run a subset of tests:

```
$ py.test tests.test_scml
```

9.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.
10.1 Development Lead

- Yasser Mohammad <yasserfarouk@gmail.com>

10.2 Contributors

- Enrique Areyan <enrique_areyanviqueira@brown.edu>
- ?
CHAPTER 11

History

11.1 Release 0.3.2

- updating dependencies to latest versions

11.2 Release 0.3.1

- [Situated] Correcting multistage tournament implementation.

11.3 Release 0.3.0

- [Situated] adding StatsMonitor and WorldMonitor classes to situated
- [Situated] adding a parameter to monitor stats of a world in real-time
- [Situated] showing ttest/ktest results in evaluation (negmas tournament commands)
- [SCML] adding total_balance to take hidden money into account for Factory objects and using it in negmas tournament and negmas scml
- [SCML] enabling –cw for collusion
- [SCML] adding hidden money to agent balance when evaluating it.
- [SCML] adding more debugging information to log.txt
- [Situated] adding multistage tournaments to tournament() function
- [Situated] adding control of the number of competitor in each world to create_tournament() and to negmas tournament create command
- [Core] avoid invalid or incomplete outcome proposals in SAOMechanism
- [Situated] adding metric parameter to evaluate_tournaments and corresponding tournament command to control which metric is used for calculating the winner. Default is mean.
- [SCML] adding the ability to prevent CFP tampering and to ignore negotiated penalties to SCMLWorld
- [SCML] adding the possibility of ignore negotiated penalty in world simulation
• [SCML] saving bankruptcy events in stats (SCML)
• [SCML] improving bankruptcy processing
• [SCML] deep copying of parameters in collusion
• [Situated] saving extra score stats in evaluate_tournament
• [Core] avoiding a future warning in pandas
• [Situated] more printing in winners and combine commands
• [Situated] removing unnecessary balance/storage data from combine_tournament_stats
• [Situated] adding aggregate states to evaluate_tournament and negmas tournament commands
• [Situated] adding kstest
• [Situated] adding and disabling dependent t-tests to evaluate_tournament
• [Situated] adding negmas tournament combine to combine and evaluate multiple tournaments without a common root
• [Situated] avoiding an exception if combine_tournament is called with no scores
• [Situated] always save world stats in tournaments even in compact mode
• [SCML] reversing sabotage score
• [SCML] correcting factory number capping
• [SCML] more robust consumer
• [Core] avoid an exception if a ufun is not defined for a negotiator when logging
• [SCML] controlling number of colluding agents using –agents option of negmas tournament create
• [SCML] changing names of assigned worlds and multiple runs to have a unique log per world in tournament
• [SCML] controlling warnings and exception printing
• [SCML] increasing default world timeout by 50%
• [SCML] removing penalty processing from greedy
• [Core] avoid negotiation failure for negotiator exceptions
• [SCML] correcting sabotage implementation
• [CLI] adding winners subcommand to negmas tournament
• [CLI] saving all details of contracts
• [CLI] adding –steps-min and –steps-max to negmas tournament create to allow for tournaments with variable number of steps
• [CLI] removing the need to add greedy to std competition in anac 2019
• [CLI] saving log path in negmas tournament create
• [CLI] removing erroneous logs
• [CLI] enabling tournament resumption (bug fix)
• [CLI] avoiding a problem when trying to create two tournaments on the same place
• [CLI] fairer random assignment
• [CLI] more printing in negmas tournament
• [CLI] using median instead of mean for evaluating scores
• [CLI] Allowing for passing –world-config to tournament create command to change the default world settings
• [CLI] adding a print out of running competitors for verbose create_tournament
• [CLI] adding –world-config to negmas scml
• [CLI] displaying results of negmas tournament evaluate ordered by the choosen metric in the table.
• [CLI] preventing very long names
• [CLI] allowing for more configs/runs in the tournament by not trying all permutations of factory assignments.
• [CLI] adding –path to negmas tournament create
• [CLI] more printing in negmas tournament
• [CLI] reducing default n_retrials to 2
• [CLI] changing optimism from 0.0 to 0.5
• [CLI] setting reserved_value to 0.0
• [CLI] run_tournament does not call evaluate_tournament now
• [SCML] always adding greedy to std. competitions in negmas tournament
• [SCML] reducing # colluding agents to 3 by default
• [CLI] restructuring the tournament command in negmas to allow for pipelining and incremental running of tournaments.
• [SCML] adding DefaultGreedyManager to manage the behavior of default agents in the final tournament
• [CLI] avoiding overriding tournament folders if the name is repeated
• [SCML] avoiding missing reserved_value in some cases in AveragingNegotiatorUfun
• [CLI] adding the ability to control max-runs interactively to negmas tournament
• [CLI] adding the ability to use a fraction of all CPUs in tournament with parallel execution
• [SCML] exceptions in signing contracts are treated as refusal to sign them.
• [SCML] making contract execution more robust for edge cases (quantity or unit price is zero)
• [SCML] making collusion tournaments in SCML use the same number of worlds as std tournaments
• [Situated] adding ignore_contract_execution_exceptions to situated and apps.scml
• [CLI] adding --raise-exceptions/ignore-exceptions to control behavior on agent exception in negmas tournament and negmas scml commands
• [SCML] adding --path to negmas scml command to add to python path
• [SCML] supporting ignore_agent_exceptions in situated and apps.scml
• [Situated] removing total timeout by default

11.4 Release 0.2.25

• [Debugging support] making negmas scml behave similar to negmas tournament worlds
• [Improved robustness] making insurance calculations robust against rounding errors.
• [Internal change with no behavioral effect] renaming pay_insurance member of InsuranceCompany to is_insured to better document its nature
• [Debugging support] adding --balance to negmas scml to control the balance
11.5 Release 0.2.24

- separating PassThroughNegotiator, PassThroughSAONegotiator. This speeds up all simulations at the expense of backward incompatibility for the undocumented Controller pattern. If you are using this pattern, you need to create PassThroughSAONegotiator instead of SAONegotiator. If you are not using Controller or you do not know what that is, you probably safe and your code will just work.
- adding logging of negotiations and offers (very slow)
- preventing miners from buying in case sell CFPs are posted.
- avoiding exceptions if the simulator is used to buy/sell AFTER simulation time
- adding more stats to the output of negmas scml command
- revealing competitor_params parameters for anac2019_std/collusion/sabotage. This parameter always existed but was not shown in the method signature (passed as part of kwargs).

11.6 Release 0.2.23

- Avoiding backward incompatibility issue in version 0.2.23 by adding INVALID UTILITY back to both utilities and apps.scml.common

11.7 Release 0.2.22

- documentation update
- unifying the INVALID UTILITY value used by all agents/negotiators to be float(‘-inf’)
- Added reserved_value parameter to GreedyFactoryManager that allows for control of the reserved value used in all its ufuns.
- enable mechanism plotting without history and improving plotting visibility
- shortening negotiator names
- printing the average number of negotiation rounds in negmas scml command
- taking care of negotiation timeout possibility in SCML simulations

11.8 Release 0.2.21

- adding avoid_free_sales parameter to NegotiatorUtility to disable checks for zero price contracts
- adding an optional parameter “partner” to _create_annotation method to create correct contract annotations when response_to_negotiation_request is called
- Avoiding unnecessary assertion in insurance company evaluate method
- passing a copy of CFPs to on_new_cfp and on_cfp_removal methods to avoid modifications to them by agents.

11.9 Release 0.2.20

- logging name instead of ID in different debug log messages (CFP publication, rejection to negotiate)
- bug fix that caused GreedyFactoryManagers to reject valid negotiations
11.10 Release 0.2.19

- logging CFPs
- defaulting to buying insurance in negmas scml
- bug resolution related to recently added ability to use LinearUtilityFunction created by a dict with tuple outcomes
  - Adding force_numeric to lead_genius_*

11.11 Release 0.2.18

- minor updates

11.12 Release 0.2.17

- allowing anac2019_world to receive keyword arguments to pass to chain_world
- bug fix: enabling parameter passing to the mechanism if given implicitly in MechanismFactory()
  - receiving mechanisms explicitly in SCMLWorld and any other parameters of World implicitly

11.13 Release 0.2.16

- bug fix in GreedyFactoryManager to avoid unnecessary negotiation retrials.

11.14 Release 0.2.15

- Minor bug fix to avoid exceptions on consumers with None profile.
  - Small update to the README file.

11.15 Release 0.2.14

- Documentation update
  - simplifying continuous integration workflow (for development)

11.16 Release 0.2.13

- Adding new callbacks to simplify factory manager development in the SCM world - on_contract_executed, on_contract_breached - on_inventory_change, on_production_success, on_cash_transfer
- Supporting callbacks including onUfunChanged on jnegmas for SAONegotiator
- Installing jenegmas 0.2.6 by default in negmas jnegmas-setup command
11.17 Release 0.2.12

- updating run scml tutorial
- tox setting update to avoid a break in latest pip (19.1.0)
- handling an edge case with both partners committing breaches at the same time.
- testing reduced max-insurance setting
- resolving a bug in contract resolution when the same agent commits multiple money breaches on multiple contracts simultaneously.
- better assertion of correct contract execution
- resolving a bug in production that caused double counting of some production outputs when multiple lines are executed generating the same product type at the same step.
- ensuring that the storage reported through awi.state or simulator.storage_* are correct for the current step. That involves a slight change in an undocumented feature of production. In the past produced products were moved to the factory storage BEFORE the beginning of production on the next step. Now it is moved AFTER the END of production of the current step (the step production was completed). This ensures that when the factory manager reads its storage it reflects what it actually have at all times.
- improving printing of RunningCommandInfo and ProductionReport
- regenerating setup.py
- revealing jobs in FactoryState
- handling a bug that caused factories to have a single line sometimes.
- revealing the dict jobs in FactoryState which gives the scheduled jobs for each time/line
- adding always_concede option to NaiveTitForTatNegotiator
- updating insurance premium percents.
- adding more tests of NaiveTitForTatNegotiator
- removing relative_premium/premium confusion. Now evaluate_premium will always return a premium as a fraction of the contract total cost not as the full price of the insurance policy. For a contract of value 30, a premium of 0.1 means 3 money units not 0.1 money units.
- adding –config option to tournament and scml commands of negmas CLI to allow users to set default parameters in a file or using environment variables
- unifying the meaning of negative numbers for max_insurance_premium to mean never buying insurance in the scheduler, manager, and app. Now you have to set max_insurance_premium to inf to make the system
- enforcing argument types in negmas CLI
- Adding DEFAULT_NEGOTIATOR constant to apps.scml.common to control the default negotiator type used by built-agents
- making utility_function a property instead of a data member of negotiator
- adding on_ufun_changed() callback to Negotiator instead of relying on on_notification() [relying on on_notification still works].
- deprecating passing dynamic_ufun to constructors of all negotiators
- removing special treatment of AspirationNegotiator in miners
- modifications to the implementation of TitForTatNegotiator to make it more sane.
- deprecating changing the utility function directly (using negotiator.utility_function = x) AFTER the negotiation starts. It is still possible to change it up to the call to join()
- adding negmas.apps.scml.DEFAULT_NEGOTIATOR to control the default negotiator used
• improved parameter settings (for internal parameters not published in the SCML document)
• speeding up ufun dumping
• formatting update
• adding ufun logging as follows: * World and SCMLWorld has now log.ufuns_file which if not None gives a file to log the funs into. * negmas tournament and scml commands receive a --log-ufuns or --no-log-ufuns to control whether
  or not to log the ufuns into the tournament/world stats directory under the name ufuns.csv
• adding a helper add_records to add records into existing csv files.

11.18  Release 0.2.11

• minor bug fix

11.19  Release 0.2.10

• adding more control to negmas tournaments:
  1. adding --factories argument to control how many factories (at least) should exist on each production level
  2. adding --agents argument to control how many agents per competitor to instantiate. For the anac2019std type, this will be forced to 1
• adding sabotage track and anac2019_sabotage to run it
• updating test assertions for negotiators.
• tutorial update
• completed NaiveTitForTatNegotiator implementation

11.20  Release 0.2.9

• resolving a bug in AspirationNegotiator that caused an exception for ufuns with assume_normalized
• resolving a bug in ASOMechanism that caused agreements only on boundary offers.
• using jnegmas-0.2.4 instead of jnegmas-0.2.3 in negmas jnegmas-setup command

11.21  Release 0.2.8

• adding commands to FactoryState.
• Allowing JNegMAS to use GreedyFactoryManager. To do that, the Java factory manager must inherit from GreedyFactoryManager and its class name must end with either GreedyFactoryManager or GFM

11.22  Release 0.2.7

• improving naming of java factory managers in log files.
• guaranteeing serial tournaments when java factory managers are involved (to be lifter later).
• adding links to the YouTube playlist in README
• adhering to Black style

11.23 Release 0.2.6

• documentation update
• setting default world runs to 100 steps
• rounding catalog prices and historical costs to money resolution
• better defaults for negmas tournaments
• adding warnings when running too many simulations.
• added version command to negmas
• corrected the way min_factories_per_level is handled during tournament config creation.
• added --factories to negmas tournament command to control the minimum number of factories per level.
• improving naming of managers and factories for debugging purposes
• forcing reveal-names when giving debug option to any negmas command
• adding short_type_name to all Entity objects for convenient printing

11.24 Release 0.2.5

• improvements to ufun representation to speedup computation
• making default factory managers slightly less risky in their behavior in long simulations and more risky in short ones
• adding jnegmas-setup and genius-setup commands to download and install jnegmas and genius bridge
• removing the logger mixin and replaced it with parameters to World and SCMLWorld
• added compact parameter to SCMLWorld, tournament, and world generators to reduce the memory footprint
• added --compact/--debug to the command line tools to avoid memory and log explosion setting the default to --compact
• improving implementation of consumer ufun for cases with negative schedule
• changing the return type of SCMLAWI.state from Factory to FactoryState to avoid modifying the original factory. For efficiency reasons, the profiles list is passed as it is and it is possible to modify it but that is forbidden by the rules of the game.
• Speeding up and correcting financial report reception.
• Making bankruptcy reporting system-wide
• avoiding execution of contracts with negative or no quantity and logging ones with zero unit price.
• documentation update
• bug fix to resolve an issue with ufun calculation for consumers in case of over consumption.
• make the default behavior of negmas command to reveal agent types in their names
• preventing agents from publishing CFPs with the ID of other agents
• documentation update
• improved Java support
• added option default_dump_extension to ~/negmas/config.json to enable changing the format of dumps from json to yaml. Currently json is the default. This included adding a helper function helpers.dump() to dump in the selected format (or overriding it by providing a file extension).

• completing compatibility with SCML description (minor change to the consumer profile)

• added two new options to negmas tournament command: anac2019std and anac2019collusion to simulate these two tracks of the ANAC 2019 SCML. Sabotage version will be added later.

• added two new functions in apps.scml.utils anac2019_std, anac2019_collusion to simulate these two tracks of the ANAC 2019 SCML. Sabotage version will be added later.

• added assign_managers() method to SCMLWorld to allow post-init assignment of managers to factories.

• updating simulator documentation

11.25 Release 0.2.2

• modifications to achieve compatibility with JNegMAS 0.2.0

• removing the unnecessary ufun property in Negotiator

11.26 Release 0.2.0

• First ANAC 2019 SCML release

• compatible with JNegMAS 0.2.0

11.27 Release 0.1.45

• implemented money and inventory hiding

• added sugar methods to SCMLAWI that run execute for different commands: schedule_production, stop_production, schedule_job, hide_inventory, hide_money

• added a json file ~/negmas/config.json to store all global configs

• reading jar locations for both jnegmas and genius-bridge from config file

• completed bankruptcy and liquidation implementation.

• removed the unnecessary _world parameter from Entity

• Added parameters to the SCML world to control compensation parameters and default price for products with no catalog prices.

• Added contract nullification everywhere.

• updated documentation to show all inherited members of all classes and to show all non-private members

• Removing the bulletin-board from the public members of the AWI

11.28 Release 0.1.42

• documentation improvement

• basic bankruptcy implementation

• bug fixes
11.29 Release 0.1.40

- documentation update
- implementing bank and insurance company disable/enable switches
- implementing financial reports
- implementing checks for bankruptcy in all built-in agents in SCML
- implementing round timeout in SAOMechanism

11.30 Release 0.1.33

- Moving to Travis CI for continuous integration, ReadTheDocs for documentation and Codacy for code quality

11.31 Release 0.1.32

- Adding partial support to factory manager development using Java
- Adding annotation control to SCML world simulation disallowing factory managers from sending arbitrary information to co-specifics
- Removing some unnecessary dependencies
- Moving development to poetry. Now we do not keep a setup.py file and rely on poetry install

11.32 Release 0.1.3

- removing some unnecessary dependencies that may cause compilation issues

11.33 Release 0.1.2

- First public release
12.1 DEFAULT_NEGOTIATOR

```python
negmas.apps.scml.DEFAULT_NEGOTIATOR = 'negmas.sao.AspirationNegotiator'
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

12.2 INVALID_STEP

```python
negmas.apps.scml.INVALID_STEP = -1000
```

Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return x.__int__(). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, bytes, or bytearray instance representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

12.3 INVALIDUTILITY

```python
negmas.apps.scml.INVALIDUTILITY = -inf
```

Convert a string or number to a floating point number, if possible.
12.4 NO_PRODUCTION

negmas.apps.scml.NO_PRODUCTION = -1
int(x=0) -> integer int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return x.__int__(). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, bytes, or bytearray instance representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-’ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

12.5 GenericMapping


Something that can be indexed using [] or called using ()

12.6 IterableMapping


Something that can be iterated upon with Key-value pairs (e.g. list, dict, tuple).

12.7 gget

negmas.generics.gget(x, _key, default=None)
Get an item from an IterableMapping

Parameters

- **x** (Union[Callable[[Any], Any], Mapping[~KT, +VT_co], Sequence[+T_co]])
  - the generic mapping
- **_key** (Any) – key (must be immutable)
- **default** – default value if no value attached with the key is found

Examples

Example with a list

```python
>>> [gget([10, 20, 30], _) for _ in (0, 2, 1, -1, 4)]
[10, 30, 20, 30, None]
```

Example with a dictionary

```python
>>> [gget({'a':10, 'b':20, 'c':30}, _) for _ in ('a', 'c', 'b', -1, 'd')]
[10, 30, 20, None, None]
```

Example with a tuple

```python
>>> [gget((10, 20, 30), _) for _ in (0, 2, 1, -1, 4)]
[10, 30, 20, 30, None]
```

Example with a generator
[gget(range(10, 40, 10), _) for _ in (0, 2, 1, -1, 4)]
[10, 30, 20, 30, None]

Returns:

Return type **Any**

### 12.8 gmap

```python
negmas.generics.gmap(group, param)
```

 Calls or indexes the group by the param

**Parameters**

- **group** *(Union[Callable[[Any], Any], Mapping[~KT, +VT_co], Sequence[+T_co]])* – Either a Callable or a Mapping
- **param** *(Any)* – The parameters to use for mapping

**Examples**

```python
>>> gmap([1, 23, 44], 1)
23
>>> gmap({'a': 3, 'b': 5, 'c': 4}, 'c')
4
>>> gmap(lambda x: 3*x, 20)
60
```

Returns:

Return type **Any**

### 12.9 ienumerate

```python
negmas.generics.ienumerate(x)
```

Enumerates a GenericMapping.

**Parameters**

- **x** *(IterableMapping)* – A generic mapping (see GenericMapping)

**Examples**

Example with a list

```python
>>> for k, cutoff_utility in ienumerate([10, 20, 30]): print(k, cutoff_utility, end='-')
0 10-1 20-2 30-
```

Example with a dictionary

```python
>>> for k, cutoff_utility in ienumerate({'a': 10, 'b': 20, 'c': 30}): print(k, cutoff_utility, end='-')
a 10-b 20-c 30-
```

Example with a tuple

---

12.8. gmap  601
Example with a generator

```python
>>> for k, cutoff_utility in enumerate(range(10, 40, 10)): print(k, cutoff_utility, end='-')
0 10-1 20-2 30-
```

Return type `Iterable[Tuple[Any, Any]]`

Returns a generator/iterator with tuples of key-value pairs.

### 12.10 `iget`

`negmas.generics.iget(x, _key, default=None)`

Get an item from an IterableMapping

Parameters

- `x` (Union[Mapping[~KT, +VT_co], Sequence[+T_co]]) – the generic mapping
- `_key` (Any) – key (must be immutable)
- `default` – default value if no value attached with the key is found

Examples

Example with a list

```python
>>> [iget([10, 20, 30], _) for _ in (0, 2,1, -1, 4)]
[10, 30, 20, 30, None]
```

Example with a dictionary

```python
>>> [iget({'a':10, 'b':20, 'c':30}, _) for _ in ('a', 'c','b', -1, 'd')]
[10, 30, 20, None, None]
```

Example with a tuple

```python
>>> [iget((10, 20, 30), _) for _ in (0, 2,1, -1, 4)]
[10, 30, 20, 30, None]
```

Example with a generator

```python
>>> [iget(range(10, 40, 10), _) for _ in (0, 2,1, -1, 4)]
[10, 30, 20, 30, None]
```

Returns:

Return type `Any`

### 12.11 `iitems`

`negmas.generics.iitems(x)`

Enumerates a GenericMapping.

Parameters `x` (IterableMapping) – A generic mapping (see `GenericMapping`)

---

602 Chapter 12. Reference
Examples

Example with a list

```python
>>> for k, cutoff_utility in ienumerate([10, 20, 30]): print(k, cutoff_utility, end='-')
0 10-1 20-2 30-
```

Example with a dictionary

```python
>>> for k, cutoff_utility in ienumerate({'a': 10, 'b': 20, 'c': 30}): print(k, cutoff_utility, end='-')
a b c
```

Example with a tuple

```python
>>> for k, cutoff_utility in ienumerate((10, 20, 30)): print(k, cutoff_utility, end='-')
0 10-1 20-2 30-
```

Example with a generator

```python
>>> for k, cutoff_utility in ienumerate(range(10, 40, 10)): print(k, cutoff_utility, end='-')
0 10-1 20-2 30-
```

Return type `Iterable[Tuple[Any, Any]]`

Returns a generator/iterator with tuples of key-value pairs.

12.12 ikeys

`negmas.generics.ikeys(x)`

Returns all keys of the iterable.

Parameters `x (IterableMapping)` – A generic mapping (see `GenericMapping`)

Examples

Example with a list

```python
>>> for k in ikeys([10, 20, 30]): print(k, end='-')
0-1-2-
```

Example with a dictionary

```python
>>> for k in ikeys({'a': 10, 'b': 20, 'c': 30}): print(k, end='-')
a b c-
```

Example with a tuple

```python
>>> for k in ikeys((10, 20, 30)): print(k, end='-')
0-1-2-
```

Example with a generator

```python
>>> for k in ikeys(range(10, 40, 10)): print(k, end='-')
0-1-2-
```
Return type `Iterable[Any]`

Returns a generator/iterator with tuples of key-value pairs.

### 12.13 `ivalues`

`negmas.generics.ivalues(x)`

Returns all keys of the iterable.

**Parameters**

`x` (*IterableMapping*) – A generic mapping (see `GenericMapping`)

**Examples**

Example with a list

```python
>>> for k in ivalues([10, 20, 30]): print(k, end='-')
10-20-30-
```

Example with a dictionary

```python
>>> for k in ivalues({'a': 10, 'b': 20, 'c': 30}): print(k, end='-')
10-20-30-
```

Example with a tuple

```python
>>> for k in ivalues((10, 20, 30)): print(k, end='-')
10-20-30-
```

Example with a generator

```python
>>> for k in ivalues(range(10, 40, 10)): print(k, end='-')
10-20-30-
```

Return type `Iterable[Any]`

Returns a generator/iterator with tuples of key-value pairs.

### 12.14 `DEFAULT_DUMP_EXTENSION`

`negmas.helpers.DEFAULT_DUMP_EXTENSION = 'json'`

```python
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to ‘strict’.

### 12.15 `PYTHON_CLASS_IDENTIFIER`

`negmas.java.PYTHON_CLASS_IDENTIFIER = '__python_class__'`

```python
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
```

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of `object.__str__()` (if defined) or `repr(object)`. encoding defaults to `sys.getdefaultencoding()`. errors defaults to ‘strict’.
12.16 Outcome

\[
\text{negmas.outcomes.Outcome} = \text{typing.Union[negmas.outcomes.OutcomeType, typing.Tuple[typing.Union[int, float, str, list], typing.Dict[typing.Union[int, str], typing.Union[int, float, str, list]]]}
\]

An outcome is either a tuple of values or a dict with name/value pairs.

12.17 INVALID_UTILITY

\[
\text{negmas.utilities.INVALID_UTILITY} = -\infty
\]

float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

12.18 UtilityValue

\[
\text{negmas.utilities.UtilityValue} = \text{typing.Union[negmas.helpers.Distribution, float]}
\]

Either a utility_function distribution or an exact offerable_outcomes utility_function value.

UtilityFunction's always return a `UtilityValue which makes it easier to implement algorithms relying on probabilistic modeling of utility_function values.
CHAPTER 13

Indices and tables

• genindex
• modindex
• search
Symbols

_ShadowAgentMechanismInterface (class in negmas.common), 192
__call__() (negmas.tournaments.WorldGenerator method), 574
__call__() (negmas.utilities.ComplexNonlinearUtilityFunction method), 146
__call__() (negmas.utilities.ComplexWeightedUtilityFunction method), 138
__call__() (negmas.utilities(ConstUFun method), 68
__call__() (negmas.utilities.ExpDiscountedUFun method), 84
__call__() (negmas.utilities.HyperRectangleUtilityFunction method), 121
__call__() (negmas.utilities.IPUtilityFunction method), 155
__call__() (negmas.utilities.JavaUtilityFunction method), 165
__call__() (negmas.utilities.LinDiscountedUFun method), 76
__call__() (negmas.utilities.LinearUtilityAggregationFunction method), 102
__call__() (negmas.utilities.MappingUtilityFunction method), 93
__call__() (negmas.utilities.NonLinearUtilityAggregationFunction method), 112
__call__() (negmas.utilities.NonlinearHyperRectangleUtilityFunction method), 130
__call__() (negmas.utilities.RandomUtilityFunction method), 173
__call__() (negmas.utilities.UtilityFunction method), 60
_safe_max() (in module negmas.apps.scml), 335

A

ACCEPT_OFFER (negmas.outcomes.ResponseType attribute), 45
Action (class in negmas.situated), 296
action (negmas.apps.scml.Job attribute), 344
action (negmas.apps.scml.RunningCommandInfo attribute), 339
add() (negmas.mechanisms.Mechanism method), 183
add_capabilities() (negmas.genius.GeniusNegotiator method), 289
add_capabilities() (negmas.negotiators.Negotiator method), 195
add_capabilities() (negmas.negotiators.PassThroughNegotiator method), 203
add_capabilities() (negmas.sao.AspirationNegotiator method), 240
add_capabilities() (negmas.sao.JavaSAONegotiator method), 274
add_capabilities() (negmas.sao.LimitedOutcomesAcceptor method), 233
add_capabilities() (negmas.sao.LimitedOutcomesNegotiator method), 226
add_capabilities() (negmas.sao.NaiveTitForTatNegotiator method), 258
add_capabilities() (negmas.sao.NiceNegotiator method), 265
add_capabilities() (negmas.sao.OnlyBestNegotiator method), 252
add_capabilities() (negmas.sao.PassThroughSAONegotiator method), 281
add_capabilities() (negmas.sao.RandomNegotiator method), 221
add_capabilities() (negmas.sao.SAONegotiator method), 215
add_capabilities() (negmas.sao.ToughNegotiator method), 246
add_loan() (negmas.apps.scml.Factory method), 353
add_loan() (negmas.apps.scml.FactorySimulator method), 463
add_loan() (negmas.apps.scml.FastFactorySimulator method), 481
add_loan() (negmas.apps.scml.SlowFactorySimulator
method), 472
add_records() (in module negmas.helpers), 553
add_requirements() (negmas.mechanisms.Mechanism
method), 184
add_requirements() (negmas.sao.SAOMechanism
method), 211
add_section() (negmas.situated.BulletinBoard
method), 304
adjust_params() (negmas.utilities.HyperRectangleUtilityFunction
method), 121
Agent (class in negmas.situated), 298
AgentMechanismInterface (class in negmas.common), 189
AgentWorldInterface (class in negmas.situated), 314
agg_stats (negmas.tournaments.TournamentResults
attribute), 575
agreement (negmas.common.MechanismState
attribute), 191
agreement (negmas.mechanisms.Mechanism
attribute), 182
agreement (negmas.mechanisms.MechanismRoundResult
attribute), 187
agreement (negmas.sao.SAOMechanism
attribute), 210
agreement (negmas.sao.SAOState
attribute), 207
agreement (negmas.situated.Contract
attribute), 297
agreement_rate (negmas.apps.scml.SCMLWorld
attribute), 505
all (negmas.mechanisms.Mechanism
attribute), 182
all (negmas.outcomes.Issue
attribute), 47
all (negmas.outcomes.Issues
attribute), 55
all (negmas.sao.SAOMechanism
attribute), 210
all() (negmas.outcomes.Issue
method), 47
alpha_q (negmas.apps.scml.ConsumptionProfile
attribute), 523
alpha_q (negmas.apps.scml.MiningProfile
attribute), 539
alpha_t (negmas.apps.scml.MiningProfile
attribute), 539
alpha_u (negmas.apps.scml.ConsumptionProfile
attribute), 523
alpha_u (negmas.apps.scml.MiningProfile
attribute), 539
anac2019_collusion() (in module negmas.apps.scml), 331
anac2019_sabotage() (in module negmas.apps.scml), 334
anac2019_std() (in module negmas.apps.scml), 332
anac2019_tournament() (in module negmas.apps.scml), 329
anac2019_world() (in module negmas.apps.scml), 328
announce() (negmas.apps.scml.SCMLWorld
method), 506
announce() (negmas.events.EventSource
method), 321
announce() (negmas.mechanisms.Mechanism
method), 184
announce() (negmas.sao.SAOMechanism
method), 211
announce() (negmas.situated.BulletinBoard
method), 304
announce() (negmas.situated.World
method), 309
append_stats() (negmas.situated.World
method), 309
append_stats() (negmas.apps.scml.SCMLWorld
method), 506
approximate() (negmas.utilities.ComplexNonlinearUtilityFunction
class method), 146
approximate() (negmas.utilities.ComplexWeightedUtilityFunction
class method), 138
approximate() (negmas.utilities_CONSTUFun
class method), 68
approximate() (negmas.utilities.ExpDiscountedUFun
class method), 84
approximate() (negmas.utilities.HyperRectangleUtilityFunction
class method), 121
approximate() (negmas.utilities.IPUtilityFunction
class method), 155
approximate() (negmas.utilities.JavaUtilityFunction
class method), 165
approximate() (negmas.utilities.LinearUtilityAggregationFunction
class method), 102
approximate() (negmas.utilities.MappingUtilityFunction
class method), 93
approximate() (negmas.utilities.NonlinearHyperRectangleUtilityFunction
class method), 130
approximate() (negmas.utilities.NonLinearUtilityAggregationFunction
class method), 112
approximate() (negmas.utilities.RandomUtilityFunction
class method), 173
approximate() (negmas.utilities.UtilityFunction
class method), 61
around() (negmas.helpers.Distribution
method), 555
asdict() (negmas.situated.Breach
method), 298
asdict() (negmas.apps.scml.CFP
method), 349
asdict() (negmas.apps.scml.SCMLAgreement
method), 506
asdict() (negmas.common.AgentMechanismInterface method), 190
asdict() (negmas.common.MechanismState method), 192
asdict() (negmas.outcomes.OutcomeType method), 44
asdict() (negmas.sao.SAOSState method), 208
asdict() (negmas.situated.Contract method), 297
aspiration() (negmas.negotiators.AspirationMixin method), 198
aspiration_init() (negmas.negotiators.AspirationMixin method), 198
aspiration_init() (negmas.sao.AspirationNegotiator method), 240
AspirationMixin (class in negmas.negotiators), 198
AspirationNegotiator (class in negmas.sao), 238
assign_managers() (negmas.apps.scml.SCMLWorld method), 506
astuple() (negmas.apps.scml.CFP method), 349
astuple() (negmas.apps.scml.SCMLAgreement method), 347
astuple() (negmas.outcomes.OutcomeType method), 44
attach_to_world() (negmas.apps.scml.Factory method), 353
available_storage_at() (negmas.apps.scml.Factory method), 352
available_storage_at() (negmas.apps.scml.FactorySimulator method), 343
available_storage_at() (negmas.apps.scml.SlowFactorySimulator method), 346
available_storage_to() (negmas.apps.scml.Factory method), 333
available_storage_to() (negmas.apps.scml.FactorySimulator method), 336
available_storage_to() (negmas.apps.scml.SlowFactorySimulator method), 342
balance() (negmas.apps.scml.Factory attribute), 352
balance() (negmas.apps.scml.FactoryStatusUpdate attribute), 343
balance() (negmas.apps.scml.Agent attribute), 300
Bank (class in negmas.apps.scml), 438
base_type (negmas.utilities.ComplexNonlinearUtilityFunction attribute), 145
base_type (negmas.utilities.ComplexWeightedUtilityFunction attribute), 137
base_type (negmas.utilities.ConstUFun attribute), 67
awi (negmas.apps.scml.Bank attribute), 440
awi (negmas.apps.scml.Consumer attribute), 517
awi (negmas.apps.scml.DefaultBank attribute), 435
awi (negmas.apps.scml.DefaultGreedyManager attribute), 490
awi (negmas.apps.scml.DefaultInsuranceCompany attribute), 445
awi (negmas.apps.scml.DoNothingFactoryManager attribute), 376
awi (negmas.apps.scml.FactoryManager attribute), 368
awi (negmas.apps.scml.GreedyFactoryManager attribute), 384
awi (negmas.apps.scml.InsuranceCompany attribute), 450
awi (negmas.apps.scml.JavaDoNothingFactoryManager attribute), 404
awi (negmas.apps.scml.JavaDummyMiddleMan attribute), 426
awi (negmas.apps.scml.JavaFactoryManager attribute), 393
awi (negmas.apps.scml.JavaGreedyFactoryManager attribute), 415
awi (negmas.apps.scml.Miner attribute), 533
awi (negmas.apps.scml.ReactiveMiner attribute), 541
awi (negmas.apps.scml.ScheduleDrivenConsumer attribute), 525
awi (negmas.apps.scml.SCMLAgent attribute), 456
awi (negmas.situated.Agent attribute), 300
Index

base_type (negmas.utilities.ExpDiscountedUFun attribute), 83
base_type (negmas.utilities.HyperRectangleUtilityFunction attribute), 120
base_type (negmas.utilities.IPUtilityFunction attribute), 154
base_type (negmas.utilities.JavaUtilityFunction attribute), 164
base_type (negmas.utilities.LinDiscountedUFun attribute), 75
base_type (negmas.utilities.LinearUtilityAggregationFunction attribute), 101
base_type (negmas.utilities.MappingUtilityFunction attribute), 92
base_type (negmas.utilities.NonlinearUtilityAggregationFunction attribute), 129
base_type (negmas.utilities.RandomUtilityFunction attribute), 172
base_type (negmas.utilities.UtilityFunction attribute), 60
bb_query() (negmas.apps.scml.SCMLAWI method), 358
bb_query() (negmas.situated.AgentWorldInterface method), 315
bb_read() (negmas.apps.scml.SCMLAWI method), 359
bb_read() (negmas.situated.AgentWorldInterface method), 315
bb_record() (negmas.apps.scml.SCMLAWI method), 359
bb_record() (negmas.situated.AgentWorldInterface method), 315
bb_remove() (negmas.apps.scml.SCMLAWI method), 359
bb_remove() (negmas.situated.AgentWorldInterface method), 315
before_death() (negmas.genius.GeniusNegotiator method), 289
before_death() (negmas.negotiators.Negotiator method), 195
before_death() (negmas.negotiators.PassThroughNegotiator method), 203
before_death() (negmas.sao.AspirationNegotiator method), 240
before_death() (negmas.sao.JavaSAONegotiator method), 275
before_death() (negmas.sao.LimitedOutcomesAcceptor method), 233
before_death() (negmas.sao.LimitedOutcomesNegotiator method), 227
before_death() (negmas.sao.NaiveTitForTatNegotiator method), 259
before_death() (negmas.sao.NiceNegotiator method), 265
before_death() (negmas.sao.OnlyBestNegotiator method), 252
before_death() (negmas.sao.PassThroughSAONegotiator method), 281
before_death() (negmas.sao.RandomNegotiator method), 221
before_death() (negmas.sao.SAONegotiator method), 216
before_death() (negmas.sao.ToughNegotiator method), 246
bb (negmas.apps.scml.RunningCommandInfo attribute), 339
beta_q (negmas.apps.scml.ConsumptionProfile attribute), 539
beta_q (negmas.apps.scml.MiningProfile attribute), 539
beta_t (negmas.apps.scml.MiningProfile attribute), 539
beta_u (negmas.apps.scml.ConsumptionProfile attribute), 539
beta_u (negmas.apps.scml.MiningProfile attribute), 539
bookmark() (negmas.apps.scml.FactorySimulator method), 465
bookmark() (negmas.apps.scml.FastFactorySimulator method), 482
bookmark() (negmas.apps.scml.GreedyScheduler method), 499
bookmark() (negmas.apps.scml.Scheduler method), 407
bookmark() (negmas.apps.scml.SlowFactorySimulator method), 473
Breach (class in negmas.situated), 297
breach_rate (negmas.apps.scml.SCAImWorld attribute), 505
BreachProcessing (class in negmas.situated), 298
broken (negmas.common.MechanismState attribute), 191
broken (negmas.mechanisms.MechanismRoundResult attribute), 187
broken (negmas.sao.SAOState attribute), 207
BulletinBoard (class in negmas.situated), 303
business_size (negmas.apps.scml.SCMLWorld attribute), 505
buy() (negmas.apps.scml.FactorySimulator method), 353
buy() (negmas.apps.scml.FactorySimulator method), 465
buy() (negmas.apps.scml.FastFactorySimulator method), 482
buy() (negmas.apps.scml.SlowFactorySimulator method), 474
buy_insurance() (negmas.apps.scml.DefaultInsuranceCompany

NegMAS Documentation, Release 0.3.2
method), 445
buy_insurance() (negmas.apps.scml.SCMLAWI method), 359
buy_insurance() (negmas.apps.scml.SCMLWorld method), 506
buy_loan() (negmas.apps.scml.DefaultBank method), 435
call() (negmas.negotiators.Controller method), 199
call() (negmas.sao.SAOController method), 270
camel_case() (in module negmas.helpers), 550
can_accept_more_agents() (negmas.mechanisms.Mechanism method), 184
can_accept_more_agents() (negmas.sao.SAOMechanism method), 211
can_enter() (negmas.mechanisms.Mechanism method), 184
can_enter() (negmas.sao.SAOMechanism method), 211
can_expect_agreement() (negmas.apps.scml.Consumer method), 517
can_expect_agreement() (negmas.apps.scml.DefaultGreedyManager method), 491
can_expect_agreement() (negmas.apps.scml.DoNothingFactoryManager method), 376
can_expect_agreement() (negmas.apps.scml.FactoryManager method), 369
can_expect_agreement() (negmas.apps.scml.GreedyFactoryManager method), 384
can_expect_agreement() (negmas.apps.scml.JavaDoNothingFactoryManager method), 405
can_expect_agreement() (negmas.apps.scml.JavaDummyMiddleMan method), 426
can_expect_agreement() (negmas.apps.scml.JavaFactoryManager method), 394
can_expect_agreement() (negmas.apps.scml.JavaGreedyFactoryManager method), 415
can_expect_agreement() (negmas.apps.scml.Miner method), 534
can_expect_agreement() (negmas.apps.scml.ReactiveMiner method), 542
can_expect_agreement() (negmas.apps.scml.ScheduleDrivenConsumer method), 526
can_expect_agreement() (negmas.apps.scml.SCMLAgent method), 456
can_leave() (negmas.mechanisms.Mechanism method), 184
can_leave() (negmas.sao.SAOMechanism method), 211
can_participate() (negmas.mechanisms.Mechanism method), 184
can_participate() (negmas.sao.SAOMechanism method), 211
can_produce() (negmas.apps.scml.DefaultGreedyManager method), 491
can_produce() (negmas.apps.scml.GreedyFactoryManager method), 385
can_secure_needs() (negmas.apps.scml.DefaultGreedyManager method), 491
can_secure_needs() (negmas.apps.scml.GreedyFactoryManager method), 385
cancel_production() (negmas.apps.scml.SCMLAWI method), 359
cancellation_cost (negmas.apps.scml.ManufacturingProfile attribute), 340
cancellation_cost (negmas.apps.scml.ManufacturingProfileCompiled attribute), 341
cancellation_rate (negmas.apps.scml.SCMLWorld attribute), 505
cancelled_contracts (negmas.situated.World attribute), 308
capabilities (negmas.genius.GeniusNegotiator attribute), 289
capabilities (negmas.negotiators.Negotiator attribute), 195
capabilities (negmas.negotiators.PassThroughNegotiator attribute), 202
capabilities (negmas.sao.AspirationNegotiator attribute), 239
capabilities (negmas.sao.JavaSAONegotiator attribute), 274
capabilities (negmas.sao.LimitedOutcomesAcceptor attribute), 232
capabilities (negmas.sao.LimitedOutcomesNegotiator attribute), 226
capabilities (negmas.sao.NaiveTitForTatNegotiator attribute), 258
capabilities (negmas.sao.NiceNegotiator at-
confirmPartialExecution() (negmas.apps.scml.JavaDoNothingFactoryManager method), 405

confirmPartialExecution() (negmas.apps.scml.JavaDummyMiddleMan method), 426

confirmPartialExecution() (negmas.apps.scml.JavaFactoryManager method), 394

confirmPartialExecution() (negmas.apps.scml.JavaGreedyFactoryManager method), 416

conflict_level() (negmas.utilities.ComplexNonlinearUtilityFunction class method), 146

conflict_level() (negmas.utilities.ComplexWeightedUtilityFunction class method), 138

conflict_level() (negmas.utilities.ConstUFun class method), 69

conflict_level() (negmas.utilities.ExpDiscountedUFun class method), 85

conflict_level() (negmas.utilities.HyperRectangleUtilityFunction class method), 122

conflict_level() (negmas.utilities.IPUtilityFunction class method), 155

conflict_level() (negmas.utilities.JavaUtilityFunction class method), 166

conflict_level() (negmas.utilities.LinDiscountedUFun class method), 77

conflict_level() (negmas.utilities.LinearUtilityAggregationFunction class method), 103

conflict_level() (negmas.utilities.MappingUtilityFunction class method), 93

conflict_level() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 130

conflict_level() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 112

conflict_level() (negmas.utilities.RandomUtilityFunction class method), 174

conflict_level() (negmas.utilities.UtilityFunction class method), 61

create() (negmas.java.INegmasGateway class method), 566

ConstUFun (class in negmas.utilities), 67
Consumer (class in negmas.apps.scml), 515
create() (negmas.apps.scml.JavaDummyMiddleMan class method), 427
create() (negmas.apps.scml.JavaFactoryManager class method), 394
create() (negmas.apps.scml.JavaGreedyFactoryManager class method), 416
create() (negmas.apps.scml.Minor class method), 534
create() (negmas.apps.scml.ReactiveMiner class method), 543
create() (negmas.apps.scml.ScheduleDrivenConsumer class method), 527
create() (negmas.common.NamedObject class method), 189
create() (negmas.events.Notifier class method), 322
create() (negmas.genius.GeniusNegotiator class method), 290
create() (negmas.mechanisms.Mechanism class method), 184
create() (negmas.negotiators.Controller class method), 199
create() (negmas.negotiators.Negotiator class method), 195
create() (negmas.negotiators.PassThroughNegotiator class method), 203
create() (negmas.outcomes.Issue class method), 47
create() (negmas.sao.AspirationNegotiator class method), 240
create() (negmas.sao.JavaSAONegotiator class method), 275
create() (negmas.sao.LimitedOutcomesAcceptor class method), 234
create() (negmas.sao.LimitedOutcomesNegotiator class method), 227
create() (negmas.sao.NaiveTitForTatNegotiator class method), 259
create() (negmas.sao.NiceNegotiator class method), 265
create() (negmas.sao.OnlyBestNegotiator class method), 253
create() (negmas.sao.PassThroughSAONegotiator class method), 282
create() (negmas.sao.RandomNegotiator class method), 222
create() (negmas.sao.SAOController class method), 271
create() (negmas.sao.SAOMechanism class method), 212
create() (negmas.sao.SAONegotiator class method), 216
create() (negmas.sao.ToughNegotiator class method), 247
create() (negmas.situated.Agent class method), 300
create() (negmas.situated.BulletinBoard class method), 304
create() (negmas.situated.Entity class method), 314
create() (negmas.situated.StatsMonitor class method), 318
create() (negmas.situated.WordMonitor class method), 319
create() (negmas.utilities.ComplexNonlinearUtilityFunction class method), 147
create() (negmas.utilities.ComplexWeightedUtilityFunction class method), 139
create() (negmas.utilities.ConcatenateUtilityFunction class method), 199
create() (negmas.utilities.ConstUFun class method), 50
create() (negmas.utilities.ExpDiscountedUFun class method), 85
create() (negmas.utilities.HyperDiscountedUFun class method), 122
create() (negmas.utilities.IPUtilityFunction class method), 156
create() (negmas.utilities.JavaUtilityFunction class method), 166
create() (negmas.utilities.LinDiscountedUFun class method), 77
create() (negmas.utilities.LinearUtilityAggregationFunction class method), 103
create() (negmas.utilities.MappingUtilityFunction class method), 94
create() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 131
create() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 113
create() (negmas.utilities.RandomUtilityFunction class method), 174
create() (negmas.utilities.UtilityFunction class method), 62
create_loggers() (in module negmas.helpers), 550
create_negotiator() (negmas.negotiators.Controller method), 199
create_negotiator() (negmas.sao.SAOController method), 271
credit_rating() (negmas.apps.scml.DefaultBank method), 435
current_offer (negmas.sao.SAOSState attribute), 207
current_proposer (negmas.sao.SAOSState attribute), 207
current_step (negmas.apps.scml.SCMLAI attribute), 358
current_step (negmas.situated.AgentWorldInterface attribute), 314
cv (negmas.apps.scml.ConsumptionProfile attribute), 523
cv (negmas.apps.scml.MiningProfile attribute), 539
data (negmas.events.Event attribute), 321
data (negmas.events.Notification attribute), 322
data (negmas.situated.BulletinBoard attribute), 304
default_signing_delay (negmas.situated.AgentWorldInterface attribute), 314
default_signing_delay (negmas.situated.BulletinBoard attribute), 304
DEFAULT_DUMP_EXTENSION (in module negmas.helpers), 604
DEFAULT_JAVA_PORT (negmas.java.JNegmasGateway attribute), 566
DEFAULT_NEGOTIATOR (in module negmas.apps.scml), 599
DEFAULT_PYTHON_PORT (negmas.java.JNegmasGateway attribute), 566
default_signing_delay (negmas.apps.scml.SCMLAWI attribute), 358
default_signing_delay (negmas.situated.AgentWorldInterface attribute), 314
DefaultBank (class in negmas.apps.scml), 433
DefaultGreedyManager (class in negmas.apps.scml), 488
DefaultInsuranceCompany (class in negmas.apps.scml), 443
delete_bookmark() (negmas.apps.scml.FactorySimulator method), 465
delete_bookmark() (negmas.apps.scml.FastFactorySimulator method), 499
delete_bookmark() (negmas.apps.scml.SlowFactorySimulator method), 474
discrete_outcomes() (negmas.common.AgentMechanismInterface method), 190
discrete_outcomes() (negmas.mechanisms.Mechanism method), 184
discrete_outcomes() (negmas.sao.SAOMechanism method), 212
discrete_outcomes() (negmas.common._ShadowAgentMechanismInterface method), 192
Distribution (class in negmas.helpers), 554
distribution() (negmas.utilities.IPUtilityFunction method), 156
do_nothing() (negmas.apps.scml.RunningCommandInfo class method), 427
do_nothing_manager() (negmas.apps.scml.JavaFactoryManager class method), 394
do_nothing_manager() (negmas.apps.scml.JavaGreedyFactoryManager class method), 416
DoNothingFactoryManager (class in negmas.apps.scml), 374
dump() (in module negmas.helpers), 553
dynamic_entry (negmas.mechanisms.Mechanism attribute), 182
dynamic_entry (negmas.sao.SAOMechanism attribute), 210
dynamicity (negmas.apps.scml.ConsumptionProfile attribute), 523
Entity (class in negmas.situated), 313
enumerate() (negmas.outcomes.Issue class method), 47
enumerate_outcomes() (in module negmas.outcomes), 42
error (negmas.mechanisms.MechanismRoundResult attribute), 187
error_details (negmas.common.MechanismState attribute), 191
error_details (negmas.mechanisms.MechanismRoundResult attribute), 187
error_details (negmas.sao.SAOSState attribute), 207
eu (negmas.genius.GeniusNegotiator attribute), 289
eu (negmas.sao.AspirationNegotiator attribute), 239
eu (negmas.sao.JavaSAONegotiator attribute), 274
eu (negmas.sao.LimitedOutcomesAccepter attribute), 233
eu (negmas.sao.LimitedOutcomesNegotiator attribute), 226
eu (negmas.sao.NaiveTitForTatNegotiator attribute), 258
eu (negmas.sao.NiceNegotiator attribute), 264
class method), 491
from_config() (negmas.apps.scml.DefaultInsuranceCompany class method), 446
from_config() (negmas.apps.scml.DoNothingFactoryManager class method), 377
from_config() (negmas.apps.scml.FactoryManager class method), 369
from_config() (negmas.apps.scml.GreedyFactoryManager class method), 385
from_config() (negmas.apps.scml.InsuranceCompany class method), 451
from_config() (negmas.apps.scml.JavaDoNothingFactoryManager class method), 405
from_config() (negmas.apps.scml.JavaDummyMiddleMan class method), 427
from_config() (negmas.apps.scml.JavaFactoryManager class method), 394
from_config() (negmas.apps.scml.JavaGreedyFactoryManager class method), 416
from_config() (negmas.apps.scml.JavaMiner class method), 534
from_config() (negmas.apps.scml.JavaReactiveMiner class method), 543
from_config() (negmas.apps.scml.ScheduleDrivenConsumer class method), 527
from_config() (negmas.apps.scml.SCMLAgent class method), 457
from_config() (negmas.apps.scml.SCMLWorld class method), 508
from_config() (negmas.helpers.ConfigReader class method), 300
from_config() (negmas.situated.Agent class method), 304
from_config() (negmas.situated.BulletinBoard class method), 309
from_config() (negmas.situated.World class method), 508
from_dict() (negmas.apps.scml.JavaDoNothingFactoryManager from_dict method), 406
from_dict() (negmas.apps.scml.JavaDummyMiddleMan from_dict method), 427
from_dict() (negmas.apps.scml.JavaFactoryManager from_dict method), 395
from_dict() (negmas.apps.scml.JavaGreedyFactoryManager from_dict method), 417
from_dict() (negmas.java.JavaCallerMixin class method), 565
from_dict() (negmas.sao.JavaSAONegotiator class method), 275
from_dict() (negmas.utilities.JavaUtilityFunction class method), 166
from_genius() (negmas.outcomes.Issue from_genius method), 48
from_genius() (negmas.utilities.ComplexNonlinearUtilityFunction from_genius method), 147
from_genius() (negmas.utilities.ComplexWeightedUtilityFunction from_genius method), 139
from_genius() (negmas.utilities.ConstUFun from_genius method), 70
from_genius() (negmas.utilities.ExpDiscountedUFun from_genius method), 85
from_genius() (negmas.utilities.HyperRectangleUtilityFunction from_genius method), 123
from_genius() (negmas.utilities.IPUtilityFunction from_genius method), 156
from_genius() (negmas.utilities.JavaUtilityFunction from_genius method), 166
from_genius() (negmas.utilities.LinearUtilityAggregationFunction from_genius method), 103
from_genius() (negmas.utilities.MappingUtilityFunction from_genius method), 94
from_genius() (negmas.utilities.NonlinearHyperRectangleUtilityFunction from_genius method), 131
from_genius() (negmas.utilities.NonLinearUtilityAggregationFunction from_genius method), 113
from_genius() (negmas.utilities.RandomUtilityFunction from_genius method), 175
from_genius() (negmas.utilities.UtilityFunction from_genius method), 62
from_issue_collection() (negmas.outcomes.Issues from_issue_collection method), 55
from_java() (negmas.apps.scml.CFP from_java method), 349
from_java() (negmas.apps.scml.CFP class method), 349
from_java() (negmas.outcomes.Issue from java method), 48
from_manufacturing_profile() (negmas.apps.scml.ManufacturingProfileCompiled from_manufacturing_profile method), 342

620 Index
from_mapping() (negmas.utilities.IPUtilityFunction method), 157
from_single_issue() (negmas.outcomes.Issues class method), 55
from_ufun() (negmas.utilities.IPUtilityFunction method), 157
from_xml_str() (negmas.outcomes.Issue class method), 48
from_xml_str() (negmas.utilities.ComplexNonlinearUtilityFunction class method), 148
from_xml_str() (negmas.utilities.ComplexWeightedUtilityFunction class method), 140
from_xml_str() (negmas.utilities.ConstUFun class method), 70
from_xml_str() (negmas.utilities.ExpDiscountedUFun class method), 86
from_xml_str() (negmas.utilities.HyperRectangleUtilityFunction class method), 123
from_xml_str() (negmas.utilities.IPUtilityFunction class method), 158
from_xml_str() (negmas.utilities.JavaUtilityFunction class method), 167
from_xml_str() (negmas.utilities.LinDiscountedUFun class method), 78
from_xml_str() (negmas.utilities.LinearUtilityAggregationFunction class method), 104
from_xml_str() (negmas.utilities.MappingUtilityFunction class method), 95
from_xml_str() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 132
from_xml_str() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 114
from_xml_str() (negmas.utilities.RandomUtilityFunction class method), 175
from_xml_str() (negmas.utilities.UtilityFunction class method), 62

generate_bilateral() (negmas.utilities.ComplexWeightedUtilityFunction class method), 142
generate_bilateral() (negmas.utilities.ConstUFun class method), 72
generate_bilateral() (negmas.utilities.ExpDiscountedUFun class method), 88
generate_bilateral() (negmas.utilities.HyperRectangleUtilityFunction class method), 125
generate_bilateral() (negmas.utilities.IPUtilityFunction class method), 160
generate_bilateral() (negmas.utilities.JavaUtilityFunction class method), 169
generate_bilateral() (negmas.utilities.LinDiscountedUFun class method), 80
generate_bilateral() (negmas.utilities.LinearUtilityAggregationFunction class method), 106
generate_bilateral() (negmas.utilities.MappingUtilityFunction class method), 97
generate_bilateral() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 134
generate_bilateral() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 116
generate_bilateral() (negmas.utilities.RandomUtilityFunction class method), 177
generate_bilateral() (negmas.utilities.UtilityFunction class method), 64
generate_random() (negmas.utilities.ComplexNonlinearUtilityFunction class method), 150
generate_random() (negmas.utilities.ComplexWeightedUtilityFunction class method), 142
generate_random() (negmas.utilities.ConstUFun class method), 72
generate_random() (negmas.utilities.ExpDiscountedUFun class method), 88
generate_random() (negmas.utilities.HyperRectangleUtilityFunction class method), 125

gateway (negmas.java.JNegmasGateway attribute), 566
generate() (negmas.outcomes.Issue static method), 50
generate_bilateral() (negmas.utilities.ComplexNonlinearUtilityFunction class method), 150

generate() (negmas.utilities.IPUtilityFunction class method), 158

generate() (negmas.utilities.JavaUtilityFunction class method), 167
generate() (negmas.utilities.LinDiscountedUFun class method), 78
generate() (negmas.utilities.LinearUtilityAggregationFunction class method), 104
generate() (negmas.utilities.MappingUtilityFunction class method), 95
generate() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 132
generate() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 114
generate() (negmas.utilities.RandomUtilityFunction class method), 175
generate() (negmas.utilities.UtilityFunction class method), 62

generate() (negmas.outcomes.Issues class method), 55
generate() (negmas.utilities.IPUtilityFunction class method), 158
generate() (negmas.utilities.JavaUtilityFunction class method), 167
generate() (negmas.utilities.LinDiscountedUFun class method), 78
generate() (negmas.utilities.LinearUtilityAggregationFunction class method), 104
generate() (negmas.utilities.MappingUtilityFunction class method), 95
generate() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 132
generate() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 114
generate() (negmas.utilities.RandomUtilityFunction class method), 175
generate() (negmas.utilities.UtilityFunction class method), 62

G

Index 621
method), 169

generate_random() (negmas.utilities.LinearDiscountedUFun
class method), 80

generate_random() (negmas.utilities.LinearUtilityAggregationFunction
class method), 106

generate_random() (negmas.utilities.MappingUtilityFunction
class method), 97

generate_random() (negmas.utilities.NonlinearHyperRectangleUtilityFunction
class method), 134

generate_random() (negmas.utilities.NonLinearUtilityAggregationFunction
class method), 116

generate_random() (negmas.utilities.RandomUtilityFunction
class method), 178

generate_random_bilateral() (negmas.utilities.ComplexNonlinearUtilityFunction
class method), 150

generate_random_bilateral() (negmas.utilities.ComplexWeightedUtilityFunction
class method), 142

generate_random_bilateral() (negmas.utilities.ConstUFun class method), 73

generate_random_bilateral() (negmas.utilities.ExpDiscountedUFun class method), 89

generate_random_bilateral() (negmas.utilities.HyperRectangleUtilityFunction
class method), 126

generate_random_bilateral() (negmas.utilities.IPUtilityFunction
class method), 161

generate_random_bilateral() (negmas.utilities.JavaUtilityFunction
class method), 170

generate_random_bilateral() (negmas.utilities.LinearDiscountedUFun
class method), 81

generate_random_bilateral() (negmas.utilities.LinearUtilityAggregationFunction
class method), 106

generate_random_bilateral() (negmas.utilities.MappingUtilityFunction
class method), 97

generate_random_bilateral() (negmas.utilities.NonlinearHyperRectangleUtilityFunction
class method), 134

generate_random_bilateral() (negmas.utilities.NonLinearUtilityAggregationFunction
class method), 116

genius_bridge_is_running() (in module negmas.genius), 287

get() (negmas.outcomes.OutcomeType method), 44

gerivisdom() (in module negmas.situated.Contract method), 297

genericMapping (in module negmas.generics), 600

genius_negotiator (class in negmas.genius), 287

generates_random() (negmas.apps.scml.CFP method), 349

generate_random() (negmas.apps.scml.SCMLAgreement method), 347

generate_random() (negmas.helpers.get_info() method), 552

genius_negotiator (class in negmas.genius), 287

genericMapping (in module negmas.generics), 600

genius_negotiator (class in negmas.genius), 287

generate_random_bilateral() (negmas.utilities.UtilityFunction class method), 65

genius_negotiator (class in negmas.genius), 287

generate_random_bilateral() (negmas.utilities.UtilityFunction class method), 65

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287

genius_negotiator (class in negmas.genius), 287
negmas.apps.scml.JavaFactoryManager

negmas.apps.scml.JavaGreedyFactoryManager

negmas.apps.scml.JavaDoNothingFactoryManager

negmas.apps.scml.JavaFactoryManager

negmas.apps.scml.JavaGreedyFactoryManager

negmas.common._ShadowAgentMechanismInterface

negmas.common._ShadowAgentMechanismInterface

negmas.apps.scml.JavaDoNothingFactoryManager

negmas.apps.scml.JavaFactoryManager

negmas.apps.scml.JavaGreedyFactoryManager

negmas.common._ShadowAgentMechanismInterface
NegMAS Documentation, Release 0.3.2

mas.apps.scml.JavaGreedyFactoryManager (method), 417
getState() (negmas.common._ShadowAgentMechanism method), 193
get() (in module negmas.generics), 600
gmap() (in module negmas.generics), 601
goto() (negmas.apps.scml.SlowFactorySimulator method), 475
greedy_manager() (negmas.apps.scml.JavaDoNothingFactoryManager class method), 406
greedy_manager() (negmas.apps.scml.JavaDummyMiddleMan class method), 428
greedy_manager() (negmas.apps.scml.JavaFactoryManager class method), 395
greedy_manager() (negmas.apps.scml.JavaGreedyFactoryManager class method), 417
GreedyFactoryManager (class in negmas.apps.scml), 382
GreedyScheduler (class in negmas.apps.scml), 498

H

has_error (negmas.common.MechanismState attribute), 191
has_error (negmas.sao.SAOState attribute), 207
hidden_money (negmas.apps.scml.Factory attribute), 352
hidden_storage (negmas.apps.scml.Factory attribute), 352
hide_funds() (negmas.apps.scml.Factory method), 353
hide_funds() (negmas.apps.scml.SCMLAWI method), 360
hide_inventory() (negmas.apps.scml.SCMLAWI method), 360
hide_product() (negmas.apps.scml.Factory method), 353
historical_cost (negmas.apps.scml.Process attribute), 338
history (negmas.mechanisms.Mechanism attribute), 182
history (negmas.sao.SAOMechanism attribute), 210
humanize_time() (in module negmas.helpers), 552
HyperRectangleUtilityFunction (class in negmas.utilities), 118

I

id (negmas.apps.scml.FactoryManager attribute), 376
id (negmas.apps.scml.FactoryManager class attribute), 368
id (negmas.apps.scml.FactoryManager attribute), 384
id (negmas.apps.scml.FactoryManager attribute), 440
id (negmas.apps.scml.FactoryManager attribute), 450
id (negmas.apps.scml.FactoryManager attribute), 404
id (negmas.apps.scml.FactoryManager attribute), 393
id (negmas.apps.scml.FactoryManager attribute), 415
id (negmas.apps.scml.FactoryManager attribute), 533
id (negmas.apps.scml.FactoryManager attribute), 338
id (negmas.apps.scml.FactoryManager attribute), 337
id (negmas.apps.scml.FactoryManager attribute), 542
id (negmas.apps.scml.FactoryManager attribute), 526
id (negmas.apps.scml.FactoryManager attribute), 456
id (negmas.common.NamedObject attribute), 188
id (negmas.events.Notifier attribute), 322
id (negmas.genius.GeniusNegotiator attribute), 289
id (negmas.mechanisms.Mechanism attribute), 182
id (negmas.negotiators.Controller attribute), 199
id (negmas.negotiators.Negotiator attribute), 195
id (negmas.negotiators.PassThroughNegotiator attribute), 202
id (negmas.outcomes.Issue attribute), 47
id (negmas.sao.AspirationNegotiator attribute), 239
id (negmas.sao.JavaSAONegotiator attribute), 274
id (negmas.sao.LimitedOutcomesAcceptor attribute), 233
id (negmas.sao.LimitedOutcomesNegotiator attribute), 226
id (negmas.sao.NaiveTitForTatNegotiator attribute), 258
id (negmas.sao.NiceNegotiator attribute), 264
id (negmas.sao.OnlyBestNegotiator attribute), 252
id (negmas.sao.PassThroughSAONegotiator attribute), 281
id (negmas.sao.RandomNegotiator attribute), 221
id (negmas.sao.SAONegotiator attribute), 270
id (negmas.sao.SAONegotiator attribute), 210
id (negmas.sao.SAONegotiator attribute), 215
id (negmas.sao.ToughNegotiator attribute), 246
id (negmas.situated.Agent attribute), 300
id (negmas.situated.BulletinBoard attribute), 304
id (negmas.situated.Entity attribute), 313
id (negmas.situated.StatsMonitor attribute), 318
id (negmas.situated.WorldMonitor attribute), 319
id (negmas.utilities.ComplexNonlinearUtilityFunction attribute), 145
id (negmas.utilities.ComplexWeightedUtilityFunction attribute), 137
id (negmas.utilities.ConstUFun attribute), 67

624

Index
init() (negmas.situated.Agent method), 301
init() (negmas.situated.StatsMonitor method), 318
init() (negmas.situated.WorldMonitor method), 319
init_() (negmas.apps.scml.Bank method), 441
init_() (negmas.apps.scml.Consumer method), 518
init_() (negmas.apps.scml.DefaultBank method), 436
init_() (negmas.apps.scml.DefaultGreedyManager method), 492
init_() (negmas.apps.scml.DefaultInsuranceCompany method), 447
init_() (negmas.apps.scml.DoNothingFactoryManager method), 378
init_() (negmas.apps.scml.FactoryManager method), 370
init_() (negmas.apps.scml.GreedyFactoryManager method), 386
init_() (negmas.apps.scml.InsuranceCompany method), 452
init_() (negmas.apps.scml.JavaDoNothingFactoryManager method), 406
init_() (negmas.apps.scml.JavaDummyMiddleMan method), 428
init_() (negmas.apps.scml.JavaFactoryManager method), 396
init_() (negmas.apps.scml.JavaGreedyFactoryManager method), 417
init_() (negmas.apps.scml.Miner method), 535
init_() (negmas.apps.scml.ReactiveMiner method), 543
init_() (negmas.apps.scml.ScheduleDrivenConsumer method), 527
init_() (negmas.apps.scml.SCMLAgent method), 458
init_java_bridge() (negmas.apps.scml.JavaDoNothingFactoryManager method), 406
init_java_bridge() (negmas.apps.scml.JavaFactoryManager method), 428
negmas.java.JavaCallerMixin method, 565
negmas.sao.JavaSAONegotiator method, 275
negmas.utilities.JavaUtilityFunction method, 170
keys() (negmas.common.AgentMechanismInterface method), 190
keys() (negmas.common.MechanismState method), 192
keys() (negmas.outcomes.OutcomeType method), 44
keys() (negmas.sao.SAOState method), 208
keys() (negmas.situated.Contract method), 297
kill_negotiator() (negmas.negotiators.Controller method), 200
kill_negotiator() (negmas.sao.SAOController method), 271
kstest (negmas.tournaments.TournamentResults attribute), 575

L

level (negmas.situated.Breach attribute), 298
LimitedOutcomesAcceptor (class in negmas.sao), 231
LimitedOutcomesNegotiator (class in negmas.sao), 224
LinDiscountedUFun (class in negmas.utilities), 74
line (negmas.apps.scml.Job attribute), 344
line (negmas.apps.scml.ManufacturingProfile attribute), 340
line (negmas.apps.scml.ManufacturingProfileCompiled attribute), 341
line (negmas.apps.scml.ProductionFailure attribute), 346
line_schedules (negmas.apps.scml.Factory attribute), 352
line_schedules_at() (negmas.apps.scml.FactorySimulator method), 466
line_schedules_at() (negmas.apps.scml.FastFactorySimulator method), 483
line_schedules_at() (negmas.apps.scml.SlowFactorySimulator method), 475
line_schedules_to() (negmas.apps.scml.FactorySimulator method), 466
line_schedules_to() (negmas.apps.scml.FastFactorySimulator method), 484
line_schedules_to() (negmas.apps.scml.SlowFactorySimulator method), 475
LinearUtilityAggregationFunction (class in negmas.utilities), 99
load_genius_domain() (class in negmas.inout), 558
load_genius_domain_from_folder() (module negmas.inout), 559
Loan (class in negmas.apps.scml), 350
loans (negmas.apps.scml.Factory attribute), 352
loans_at() (negmas.apps.scml.FactorySimulator method), 466
loans_at() (negmas.apps.scml.FastFactorySimulator method), 484
loans_at() (negmas.apps.scml.SlowFactorySimulator method), 475
loans_to() (negmas.apps.scml.FactorySimulator method), 466
loans_to() (negmas.apps.scml.FastFactorySimulator method), 484
loans_to() (negmas.apps.scml.SlowFactorySimulator method), 475
loc (negmas.helpers.Distribution attribute), 555
logdebug() (negmas.apps.scml.SCMLAWI method), 360
logdebug() (negmas.apps.scml.SCMLWorld method), 509
logdebug() (negmas.situated.AgentWorldInterface method), 316
logerror() (negmas.situated.World method), 310
logerror() (negmas.apps.scml.SCMLWorld method), 509
logerror() (negmas.situated.AgentWorldInterface method), 316
loginfo() (negmas.situated.AgentWorldInterface method), 310
loginfo() (negmas.apps.scml.SCMLWorld method), 509
loginfo() (negmas.situated.AgentWorldInterface method), 316
logwarning() (negmas.apps.scml.SCMLWorld method), 509
logwarning() (negmas.situated.AgentWorldInterface method), 316
logwarning() (negmas.situated.World method), 310

M

make_bankrupt() (negmas.apps.scml.SCMLWorld method), 510
make_discounted_ufun() (module negmas.utilities), 58
make_empty() (negmas.apps.scml.FactoryStatusUpdate method), 343
ManufacturingProfile (class in negmas.apps.scml), 340
ManufacturingProfileCompiled (class in negmas.apps.scml), 341
MappingUtilityFunction (class in negmas.utilities), 90
max_n_agents (negmas.mechanisms.Mechanism attribute), 182
max_n_agents (negmas.sao.SAOMechanism attribute), 210
max_penalty (negmas.apps.scml.CFP attribute), 349
max_quantity (negmas.apps.scml.CFP attribute), 349
max_signing_delay (negmas.apps.scml.CFP attribute), 349
max_storage (negmas.apps.scml.Factory attribute), 352
max_storage (negmas.apps.scml.FactorySimulator attribute), 463
max_storage (negmas.apps.scml.FastFactorySimulator attribute), 481
max_storage (negmas.apps.scml.SlowFactorySimulator attribute), 472
max_time (negmas.apps.scml.CFP attribute), 349
max_unit_price (negmas.apps.scml.CFP attribute), 349
MAX_UNIT_PRICE (negmas.apps.scml.ScheduleDrivenConsumer attribute), 525
mean() (negmas.helpers.Distribution method), 555
Mechanism (class in negmas.mechanisms), 181
mechanism_state (negmas.situated.Contract attribute), 297
MechanismRoundResult (class in negmas.mechanisms), 186
MechanismState (class in negmas.common), 190
META_NEGOTIATION (negmas.situated.BreachProcessing attribute), 298
min_balance (negmas.apps.scml.Factory attribute), 352
min_penalty (negmas.apps.scml.CFP attribute), 349
min_quantity (negmas.apps.scml.CFP attribute), 349
min_signing_delay (negmas.apps.scml.CFP attribute), 349
min_storage (negmas.apps.scml.Factory attribute), 353
min_time (negmas.apps.scml.CFP attribute), 349
min_unit_price (negmas.apps.scml.CFP attribute), 349
Miner (class in negmas.apps.scml), 531
MiningProfile (class in negmas.apps.scml), 539
missing_inputs (negmas.apps.scml.ProductionFailure attribute), 346
missing_money (negmas.apps.scml.ProductionFailure attribute), 346
missing_space (negmas.apps.scml.ProductionFailure attribute), 346
MissingInput (class in negmas.apps.scml), 345
money_resolution (negmas.apps.scml.CFP attribute), 349
N
n_acceptances (negmas.sao.SAOState attribute), 207
n_lines (negmas.apps.scml.Factory attribute), 353
n_lines (negmas.apps.scml.FactorySimulator attribute), 463
n_lines (negmas.apps.scml.FastFactorySimulator attribute), 481
n_lines (negmas.apps.scml.SlowFactorySimulator attribute), 472
n_negotiation_rounds_failed (negmas.apps.scml.SCMLWorld attribute), 505
n_negotiation_rounds_successful (negmas.apps.scml.SCMLWorld attribute), 505
n_negotiators (negmas.common.AgentMechanismInterface attribute), 189
n_negotiators (negmas.common.MechanismState attribute), 191
n_negotiators (negmas.sao.SAOState attribute), 207
n_outcomes (negmas.mechanisms.Mechanism attribute), 182
n_outcomes (negmas.sao.SAOMechanism attribute), 210
n_outcomes() (negmas.outcomes.Issue static method), 50
n_outcomes() (negmas.outcomes.Issues method), 56
n_steps (negmas.apps.scml.FactorySimulator attribute), 463
n_steps (negmas.apps.scml.FastFactorySimulator attribute), 481
n_steps (negmas.apps.scml.ManufacturingProfile attribute), 340
n_steps (negmas.apps.scml.ManufacturingProfileCompiled attribute), 341
n_steps (negmas.apps.scml.RunningCommandCompiled attribute), 339
n_steps (negmas.apps.scml.SCMLWorld attribute), 358
n_steps (negmas.apps.scml.SlowFactorySimulator attribute), 472
n_steps (negmas.mechanisms.Mechanism attribute), 182
n_steps (negmas.sao.SAOMechanism attribute), 210
n_steps (negmas.situated.AgentWorldInterface attribute), 314
NaiveTitForTatNegotiator (class in negmas.sao), 257
name (negmas.apps.scml.Factory attribute), 440
name (negmas.apps.scml.Consumer attribute), 517
name (negmas.apps.scml.DefaultBank attribute), 435
nullified_at (negmas.situated.Agent attribute), 297
nullify_contract() (negmas.apps.scml.SCMLWorld method), 510
num_outcomes() (in module negmas.outcomes), 43

O
on_agent_bankrupt() (negmas.apps.scml.Consumer method), 519
on_cash_transfer() (negmas.apps.scml.Consumer method), 519
on_cash_transfer() (negmas.apps.scml.DefaultGreedyManager method), 493
on_cash_transfer() (negmas.apps.scml.JavaDoNothingFactoryManager method), 407

Index 631
on_contract_cancelled() (neg-mas.apps.scml.DefaultBank method), 436
on_contract_cancelled() (neg-mas.apps.scml.DefaultGreedyManager method), 493
on_contract_cancelled() (neg-mas.apps.scml.DefaultInsuranceCompany method), 447
on_contract_cancelled() (neg-mas.apps.scml.DoNothingFactoryManager method), 379
on_contract_cancelled() (neg-mas.apps.scml.FactoryManager method), 371
on_contract_cancelled() (neg-mas.apps.scml.GreedyFactoryManager method), 387
on_contract_cancelled() (neg-mas.apps.scml.InsuranceCompany method), 452
on_contract_cancelled() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 408
on_contract_cancelled() (neg-mas.apps.scml.JavaDummyMiddleMan method), 430
on_contract_cancelled() (neg-mas.apps.scml.JavaFactoryManager method), 398
on_contract_cancelled() (neg-mas.apps.scml.JavaGreedyFactoryManager method), 419
on_contract_cancelled() (neg-mas.apps.scml.Miner method), 536
on_contract_cancelled() (neg-mas.apps.scml.ReactiveMiner method), 544
on_contract_cancelled() (neg-mas.apps.scml.ScheduleDrivenConsumer method), 528
on_contract_cancelled() (neg-mas.apps.scml.SCMLAgent method), 459
on_contract_cancelled() (neg-mas.situated.Agent method), 493
on_contract_concluded() (neg-mas.apps.scml.SCMLWorld method), 510
on_contract_concluded() (neg-mas.situated.World method), 511
on_contract_executed() (neg-mas.apps.scml.Bank method), 441
on_contract_executed() (neg-mas.apps.scml.Consumer method), 520
on_contract_executed() (neg-mas.apps.scml.DefaultBank method), 436
on_contract_executed() (neg-mas.apps.scml.DefaultGreedyManager method), 493
on_contract_executed() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 408
on_contract_nullified() (neg-mas.apps.scml.Consumer method), 520
on_contract_nullified() (neg-mas.apps.scml.DefaultGreedyManager method), 493
on_contract_nullified() (neg-mas.apps.scml.DoNothingFactoryManager method), 379
on_contract_nullified() (neg-mas.apps.scml.FactoryManager method), 371
on_contract_nullified() (neg-mas.apps.scml.GreedyFactoryManager method), 387
on_contract_nullified() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 408
on_contract_nullified() (neg-mas.apps.scml.JavaDummyMiddleMan method), 430
on_contract_nullified() (neg-mas.apps.scml.JavaFactoryManager method), 398
on_contract_nullified() (neg-mas.apps.scml.JavaGreedyFactoryManager method), 419
on_contract_nullified() (neg-mas.situated.Agent method), 493
on_contract_nullified() (neg-mas.situated.World method), 511

Index 633
on_contract_nullified() (negmas.apps.scml.JavaFactoryManager method), 398
on_contract_nullified() (negmas.apps.scml.JavaGreedyFactoryManager method), 419
on_contract_nullified() (negmas.apps.scml.Miner method), 536
on_contract_nullified() (negmas.apps.scml.ReactiveMiner method), 545
on_contract_nullified() (negmas.apps.scml.ScheduleDrivenConsumer method), 529
on_contract_nullified() (negmas.apps.scml.SCMLAgent method), 459
on_contract_nullified() (negmas.apps.scml.SCMLWorld method), 510
on_contract_nullified() (negmas.situated.Agent method), 301
on_contract_nullified() (negmas.situated.World method), 311
on_contract_nullified_() (negmas.apps.scml.Bank method), 442
on_contract_nullified_() (negmas.apps.scml.Consumer method), 520
on_contract_nullified_() (negmas.apps.scml.DefaultBank method), 437
on_contract_nullified_() (negmas.apps.scml.DefaultGreedyManager method), 493
on_contract_nullified_() (negmas.apps.scml.DefaultInsuranceCompany method), 447
on_contract_nullified_() (negmas.apps.scml.FactoryManager method), 371
on_contract_nullified_() (negmas.apps.scml.DefaultInsuranceCompany method), 447
on_contract_nullified_() (negmas.apps.scml.DoNothingFactoryManager method), 379
on_contract_nullified_() (negmas.apps.scml.GreedyFactoryManager method), 437
on_contract_nullified_() (negmas.apps.scml.FactoryManager method), 371
on_contract_nullified_() (negmas.apps.scml.GreedyFactoryManager method), 437
on_contract_nullified_() (negmas.apps.scml.InsuranceCompany method), 452
on_contract_nullified_() (negmas.apps.scml.JavaDoNothingFactoryManager method), 409
on_contract_nullified_() (negmas.apps.scml.JavaFactoryManager method), 398
on_contract_nullified_() (negmas.apps.scml.JavaDummyMiddleMan method), 430
on_contract_nullified_() (negmas.apps.scml.JavaGreedyFactoryManager method), 419
on_contract_nullified_() (negmas.apps.scml.JavaFactoryManager method), 398
on_contract_nullified_() (negmas.apps.scml.JavaGreedyFactoryManager method), 419
on_contract_nullified_() (negmas.apps.scml.Miner method), 536
on_contract_nullified_() (negmas.apps.scml.ReactiveMiner method), 545
on_contract_nullified_() (negmas.apps.scml.ScheduleDrivenConsumer method), 529
on_contract_nullified_() (negmas.apps.scml.SCMLAgent method), 459
on_contract_nullified_() (negmas.apps.scml.SCMLWorld method), 510
on_contract_nullified_() (negmas.situated.Agent method), 301

634 Index
on_event() (negmas.apps.scml.Bank method), 442
on_event() (negmas.apps.scml.Consumer method), 520
on_event() (negmas.apps.scml.DefaultBank method), 437
on_event() (negmas.apps.scml.DefaultGreedyManager method), 493
on_event() (negmas.apps.scml.DefaultInsuranceCompany on_inventory_change() (negmas.apps.scml.JavaGreedyFactoryManager method), 420
on_event() (negmas.apps.scml.DefaultInsuranceCompany on_inventory_change() (negmas.apps.scml.JavaFactoryManager method), 398
on_event() (negmas.apps.scml.DefaultInsuranceCompany on_inventory_change() (negmas.apps.scml.JavaFactoryManager method), 398
on_event() (negmas.apps.scml.FactoryManager method), 387
on_event() (negmas.apps.scml.FactoryManager method), 371
on_event() (negmas.apps.scml_GREedyFactoryManager on_inventory_change() (negmas.apps.scml.SCMLAgent method), 459
on_event() (negmas.apps.scml.JavaDoNothingFactoryManager method), 545
on_event() (negmas.apps.scml.JavaDummyMiddleMan on_leave() (negmas.negotiators.Controller method), 200
on_event() (negmas.apps.scml.JavaFactoryManager method), 430
on_event() (negmas.apps.scml.JavaFactoryManager on_leave() (negmas.negotiators.PassThroughNegotiator method), 196
on_event() (negmas.apps.scml.JavaGreedyFactoryManager on_leave() (negmas.sao.AspirationNegotiator method), 241
on_event() (negmas.apps.scml.Miner method), 536
on_event() (negmas.apps.scml.ReactiveMiner method), 545
on_event() (negmas.apps.scml.ScheduleDrivenConsumer method), 529
on_event() (negmas.apps.scml.SCMLAgent method), 459
on_event() (negmas.apps.scml.SCMLWorld method), 510
on_event() (negmas.events.EventSink method), 322
on_event() (negmas.situated.Agent method), 302
on_event() (negmas.situated.World method), 311
on_inventory_change() (negmas.apps.scml.Consumer method), 520
on_inventory_change() (negmas.apps.scml.DefaultGreedyManager method), 493
on_inventory_change() (negmas.apps.scml.DoNothingFactoryManager method), 379
on_inventory_change() (negmas.apps.scml.FactoryManager method), 371
on_inventory_change() (negmas.apps.scml.GreedyFactoryManager method), 387
on_inventory_change() (negmas.apps.scml.JavaDoNothingFactoryManager method), 409
on_inventory_change() (negmas.apps.scml.JavaDummyMiddleMan method), 430
on_inventory_change() (negmas.apps.scml.JavaFactoryManager method), 398
on_inventory_change() (negmas.apps.scml.JavaGreedyFactoryManager method), 420
on_inventory_change() (negmas.apps.scml.JavaFactoryManager method), 398
on_inventory_change() (negmas.apps.scml.JavaGreedyFactoryManager method), 420
on_inventory_change() (negmas.apps.scml.JavaGreedyFactoryManager method), 420
on_leave() (negmas.sao.AspirationNegotiator method), 241
on_leave() (negmas.sao.JavaSAONegotiator method), 276
on_leave() (negmas.sao.LimitedOutcomesAccepter method), 234
on_leave() (negmas.sao.LimitedOutcomesNegotiator method), 228
on_leave() (negmas.sao.NaiveTitForTatNegotiator method), 260
on_leave() (negmas.sao.NiceNegotiator method), 266
on_leave() (negmas.sao.OnlyBestNegotiator method), 253
on_leave() (negmas.sao.PassThroughSAONegotiator method), 282
on_leave() (negmas.sao.RandomNegotiator method), 223
on_leave() (negmas.sao.SAOController method), 271
on_leave() (negmas.sao.SAOController method), 217
on_leave() (negmas.sao.ToughNegotiator method), 247
on_mechanism_error() (negmas.genius.GeniusNegotiator method), 291
on_mechanism_error() (negmas.mechanisms.Mechanism method), 185
on_mechanism_error() (negmas.negotiators.Controller method), 200
on_mechanism_error() (negmas.negotiators.Negotiator method), 196
on_mechanism_error() (negmas.negotiators.Negotiator method), 196
mas.negotiators.PassThroughNegotiator method), 204

on_mechanism_error() (negmas.sao.AspirationNegotiator method), 204

on_mechanism_error() (negmas.sao.JavaSAONegotiator method), 241

on_mechanism_error() (negmas.sao.LimitedOutcomesAcceptor method), 276

on_mechanism_error() (negmas.sao.LimitedOutcomesNegotiator method), 235

on_mechanism_error() (negmas.sao.NaiveTitForTatNegotiator method), 260

on_mechanism_error() (negmas.sao.NiceNegotiator method), 266

on_mechanism_error() (negmas.sao.OnlyBestNegotiator method), 254

on_mechanism_error() (negmas.sao.PassThroughSAONegotiator method), 283

on_mechanism_error() (negmas.sao.RandomNegotiator method), 223

on_mechanism_error() (negmas.sao.SAOController method), 272

on_mechanism_error() (negmas.sao.SAOMechanism method), 212

on_mechanism_error() (negmas.sao.SAONegotiator method), 217

on_mechanism_error() (negmas.sao.ToughNegotiator method), 247

on_neg_request_accepted() (negmas.apps.scml.Bank method), 442

on_neg_request_accepted() (negmas.apps.scml.Consumer method), 520

on_neg_request_accepted() (negmas.apps.scml.DefaultBank method), 437

on_neg_request_accepted() (negmas.apps.scml.DefaultGreedyManager method), 494

on_neg_request_accepted() (negmas.apps.scml.DefaultInsuranceCompany method), 447

on_neg_request_accepted() (negmas.apps.scml.DoNothingFactoryManager method), 379

on_neg_request_accepted() (negmas.apps.scml.FactoryManager method), 372

on_neg_request_accepted() (negmas.apps.scml.GreedyFactoryManager method), 388

on_neg_request_accepted() (negmas.apps.scml.InsuranceCompany method), 452

on_neg_request_accepted() (negmas.apps.scml.JavaDoNothingFactoryManager method), 409

on_neg_request_accepted() (negmas.apps.scml.JavaDummyMiddleMan method), 431

on_neg_request_accepted() (negmas.apps.scml.JavaFactoryManager method), 398

on_neg_request_accepted() (negmas.apps.scml.JavaGreedyFactoryManager method), 420

on_neg_request_accepted() (negmas.apps.scml.JavaMiner method), 537

on_neg_request_accepted() (negmas.apps.scml.ReactiveMiner method), 545

on_neg_request_accepted() (negmas.apps.scml.ScheduleDrivenConsumer method), 529

on_neg_request_accepted() (negmas.apps.scml.SCMALAgent method), 459

on_neg_request_accepted() (negmas.situated.Agent method), 302

on_neg_request_accepted() (negmas.situated.SituatedAgent method), 442

on_neg_request_accepted() (negmas.situated.Consumer method), 520

on_neg_request_accepted() (negmas.situated.DefaultBank method), 437

on_neg_request_accepted() (negmas.situated.DefaultGreedyManager method), 494

on_neg_request_accepted() (negmas.situated.DefaultInsuranceCompany method), 447

on_neg_request_accepted() (negmas.situated.DoNothingFactoryManager method), 379

on_neg_request_accepted() (negmas.situated.FactoryManager method), 372

on_neg_request_accepted() (negmas.situated.GreedyFactoryManager method), 388

on_neg_request_accepted() (negmas.situated.InsuranceCompany method), 452

on_neg_request_accepted() (negmas.situated.JavaDoNothingFactoryManager method), 409

on_neg_request_accepted() (negmas.situated.JavaDummyMiddleMan method), 431

on_neg_request_accepted() (negmas.situated.JavaFactoryManager method), 398
on_negotiation_end() (neg-mas.negotiators.Controller method), 201
on_negotiation_end() (neg-mas.negotiators.Negotiator method), 196
on_negotiation_end() (neg-mas.negotiators.PassThroughNegotiator method), 204
on_negotiation_end() (neg-mas.sao.AspirationNegotiator method), 241
on_negotiation_end() (neg-mas.sao.JavaSAONegotiator method), 277
on_negotiation_end() (neg-mas.sao.LimitedOutcomesAccepter method), 235
on_negotiation_end() (neg-mas.sao.LimitedOutcomesNegotiator method), 229
on_negotiation_end() (neg-mas.sao.NaiveTitForTatNegotiator method), 260
on_negotiation_end() (neg-mas.sao.NiceNegotiator method), 266
on_negotiation_end() (neg-mas.sao.OnlyBestNegotiator method), 254
on_negotiation_end() (neg-mas.sao.PassThroughSAONegotiator method), 283
on_negotiation_end() (neg-mas.sao.RandomNegotiator method), 223
on_negotiation_end() (neg-mas.sao.SAOController method), 272
on_negotiation_end() (neg-mas.sao.SAOMechanism method), 213
on_negotiation_end() (neg-mas.sao.SAONegotiator method), 217
on_negotiation_end() (neg-mas.sao.ToughNegotiator method), 248
on_negotiation_failure() (neg-mas.apps.scml.Bank method), 442
on_negotiation_failure() (neg-mas.apps.scml.Consumer method), 521
on_negotiation_failure() (neg-mas.apps.scml.DefaultBank method), 437
on_negotiation_failure() (neg-mas.apps.scml.DefaultGreedyManager method), 494
on_negotiation_failure() (neg-mas.apps.scml.DefaultInsuranceCompany method), 448
on_negotiation_failure() (neg-mas.apps.scml.DoNothingFactoryManager method), 521
on_negotiation_failure() (neg-mas.apps.scml.DoNothingFactoryManager method), 437
on_negotiation_failure() (neg-mas.apps.scml.FactoryManager method), 372
on_negotiation_failure() (neg-mas.apps.scml.GreedyFactoryManager method), 388
on_negotiation_failure() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 409
on_negotiation_failure() (neg-mas.apps.scml.JavaDummyMiddleMan method), 431
on_negotiation_failure() (neg-mas.apps.scml.JavaFactoryManager method), 399
on_negotiation_failure() (neg-mas.apps.scml.JavaGreedyFactoryManager method), 420
on_negotiation_failure() (neg-mas.apps.scml.Minor method), 537
on_negotiation_failure() (neg-mas.apps.scml.ReactiveMiner method), 545
on_negotiation_failure() (neg-mas.apps.scml.SCMLAgent method), 460
on_negotiation_failure() (neg-mas.situated.Agent method), 302
on_negotiation_failure() (neg-mas.apps.scml.Bank method), 442
on_negotiation_failure() (neg-mas.apps.scml.Consumer method), 521
on_negotiation_failure() (neg-mas.apps.scml.DefaultBank method), 437
on_negotiation_failure() (neg-mas.apps.scml.DefaultGreedyManager method), 494
on_negotiation_failure() (neg-mas.apps.scml.DefaultInsuranceCompany method), 448
on_negotiation_failure() (neg-mas.apps.scml.DoNothingFactoryManager method), 521
on_negotiation_failure() (neg-mas.apps.scml.FactoryManager method), 372
on_negotiation_failure() (neg-mas.apps.scml.GreedyFactoryManager method), 388
on_negotiation_failure() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 409

638 Index
<table>
<thead>
<tr>
<th>Event</th>
<th>Class</th>
<th>Method</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on_negotiation_failure()</code></td>
<td>neg-mas.apps.scml.JavaDummyMiddleMan method</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_failure()</code></td>
<td>neg-mas.apps.scml.JavaFactoryManager method</td>
<td>399</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_failure()</code></td>
<td>neg-mas.apps.scml.JavaGreedyFactoryManager method</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_failure()</code></td>
<td>neg-mas.apps.scml.JavaGreedyFactoryManager method</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_failure()</code></td>
<td>neg-mas.situated.Agent method</td>
<td>302</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.genius.GeniusNegotiator method</td>
<td>291</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.mechanisms.Mechanism method</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.negotiators.Controller method</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.negotiators.Negotiator method</td>
<td>197</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.negotiators.PassThroughNegotiator method</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.AspirationNegotiator method</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaSAONegotiator method</td>
<td>277</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.LimitedOutcomesAcceptor method</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.LimitedOutcomesNegotiator method</td>
<td>229</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.NaiveTitForTatNegotiator method</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.NiceNegotiator method</td>
<td>267</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.OnlyBestNegotiator method</td>
<td>254</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.PassThroughSAONegotiator method</td>
<td>283</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.RandomNegotiator method</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.SAOController method</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.SAOMechanism method</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.SAONegotiator method</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.ToughNegotiator method</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.SCMLAgent method</td>
<td>460</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.DefaultBank method</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.DefaultGreedyManager method</td>
<td>494</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.DefaultInsuranceCompany method</td>
<td>448</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.DoNothingFactoryManager method</td>
<td>380</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.GreedyFactoryManager method</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.InsuranceCompany method</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaDoNothingFactoryManager method</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaDummyMiddleMan method</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaFactoryManager method</td>
<td>399</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaGreedyFactoryManager method</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaGreedyFactoryManager method</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaMechanism method</td>
<td>537</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_start()</code></td>
<td>neg-mas.sao.JavaReactiveNegotiator method</td>
<td>546</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.Bank method</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.Consumer method</td>
<td>521</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.DefaultBank method</td>
<td>437</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.DefaultGreedyManager method</td>
<td>494</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.DefaultInsuranceCompany method</td>
<td>448</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.GreedyFactoryManager method</td>
<td>380</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.InsuranceCompany method</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.JavaDoNothingFactoryManager method</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.JavaDummyMiddleMan method</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.JavaFactoryManager method</td>
<td>399</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.ScheduleDrivenConsumer method</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.apps.scml.ScheduleDrivenConsumer method</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.sao.RandomNegotiator method</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td><code>on_negotiation_success()</code></td>
<td>neg-mas.sao.SCMLAgent method</td>
<td>460</td>
<td></td>
</tr>
</tbody>
</table>

Index 639
on_negotiation_success_(mas.apps.scml.Bank method), 442

on_negotiation_success_(negmas.apps.scml.Consumer method), 521

on_negotiation_success_(negmas.apps.scml.DefaultBank method), 437

on_negotiation_success_(negmas.apps.scml.DefaultGreedyManager method), 494

on_negotiation_success_(negmas.apps.scml.DefaultInsuranceCompany method), 448

on_negotiation_success_(negmas.apps.scml.DoNothingFactoryManager method), 380

on_negotiation_success_(negmas.apps.scml.FactoryManager method), 372

on_negotiation_success_(negmas.apps.scml.GreedyFactoryManager method), 388

on_negotiation_success_(negmas.apps.scml.JavaDoNothingFactoryManager method), 410

on_negotiation_success_(negmas.apps.scml.JavaDummyMiddleMan method), 431

on_negotiation_success_(negmas.apps.scml.JavaFactoryManager method), 399

on_negotiation_success_(negmas.apps.scml.JavaGreedyFactoryManager method), 420

on_negotiation_success_(negmas.apps.scml.Miner method), 537

on_negotiation_success_(negmas.apps.scml.ReactiveMiner method), 546

on_negotiation_success_(negmas.apps.scml.ScheduleDrivenConsumer method), 530

on_negotiation_success_(negmas.apps.scml.SCMLAgent method), 460

on_new_cfp_(negmas.apps.scml.Consumer method), 521

on_new_cfp_(negmas.apps.scml.JavaDoNothingFactoryManager method), 410

on_new_cfp_(negmas.apps.scml.JavaDummyMiddleMan method), 431

on_new_cfp_(negmas.apps.scml.JavaFactoryManager method), 399

on_new_cfp_(negmas.apps.scml.JavaGreedyFactoryManager method), 421

on_new_report_(negmas.apps.scml.Consumer method), 521

on_new_report_(negmas.apps.scml.DefaultGreedyManager method), 494

on_new_report_(negmas.apps.scml.DoNothingFactoryManager method), 410

on_new_report_(negmas.apps.scml.JavaDoNothingFactoryManager method), 431

on_new_report_(negmas.apps.scml.JavaDummyMiddleMan method), 431

on_new_report_(negmas.apps.scml.JavaFactoryManager method), 399

on_new_report_(negmas.apps.scml.JavaGreedyFactoryManager method), 421

on_new_report_(negmas.apps.scml.Miner method), 537

on_new_report_(negmas.apps.scml.ReactiveMiner method), 546

on_new_report_(negmas.apps.scml.ScheduleDrivenConsumer method), 530

on_new_report_(negmas.apps.scml.SCMLAgent method), 460
on_notification() (negmas.events.Notifiable method), 460
on_notification() (negmas.genius.GeniusNegotiator method), 291
on_notification() (negmas.negotiators.Controller method), 201
on_notification() (negmas.negotiators.Negotiator method), 197
on_notification() (negmas.negotiators.PassThroughNegotiator method), 205
on_notification() (negmas.sao.AspirationNegotiator method), 242
on_notification() (negmas.sao.JavaSAONegotiator method), 277
on_notification() (negmas.sao.LimitedOutcomesAcceptor method), 235
on_notification() (negmas.sao.LimitedOutcomesNegotiator method), 229
on_notification() (negmas.sao.NaiveTitForTatNegotiator method), 260
on_notification() (negmas.sao.NiceNegotiator method), 242
on_notification() (negmas.sao.OnlyBestNegotiator method), 254
on_notification() (negmas.sao.PassThroughSAONegotiator method), 283
on_notification() (negmas.sao.RandomNegotiator method), 223
on_notification() (negmas.sao.SAOController method), 272
on_notification() (negmas.sao.SAONegotiator method), 217
on_notification() (negmas.sao.ToughNegotiator method), 248
on_partner_proposal() (negmas.genius.GeniusNegotiator method), 291
on_partner_proposal() (negmas.sao.NaiveTitForTatNegotiator method), 261
on_partner_proposal() (negmas.sao.NiceNegotiator method), 267
on_partner_proposal() (negmas.sao.OnlyBestNegotiator method), 254
on_partner_proposal() (negmas.sao.PassThroughSAONegotiator method), 283
on_partner_proposal() (negmas.sao.SAONegotiator method), 218
on_partner_proposal() (negmas.sao.ToughNegotiator method), 248
on_partner_refused_to_propose() (negmas.genius.GeniusNegotiator method), 292
on_partner_refused_to_propose() (negmas.sao.AspirationNegotiator method), 242
on_partner_refused_to_propose() (negmas.sao.JavaSAONegotiator method), 277
on_partner_refused_to_propose() (negmas.sao.LimitedOutcomesAcceptor method), 236
on_partner_refused_to_propose() (negmas.sao.LimitedOutcomesNegotiator method), 230
on_partner_refused_to_propose() (negmas.sao.NaiveTitForTatNegotiator method), 261
on_partner_refused_to_propose() (negmas.sao.NiceNegotiator method), 267
on_partner_refused_to_propose() (negmas.sao.OnlyBestNegotiator method), 255
on_partner_refused_to_propose() (negmas.sao.PassThroughSAONegotiator method), 284
on_partner_refused_to_propose() (negmas.sao.SAOController method), 272
on_partner_refused_to_propose() (negmas.sao.SAONegotiator method), 217
on_partner_refused_to_propose() (negmas.sao.ToughNegotiator method), 248
on_partner_response() (negmas.genius.GeniusNegotiator method), 292
on_partner_response() (negmas.sao.AspirationNegotiator method), 242
on_partner_response() (negmas.sao.JavaSAONegotiator method), 277
on_partner_response() (negmas.sao.LimitedOutcomesAcceptor method), 236
on_partner_response() (negmas.sao.LimitedOutcomesNegotiator method), 229

Index 641
on_partner_response() (negmas.sao.NaiveTitForTatNegotiator method), 230

on_partner_response() (negmas.sao.NiceNegotiator method), 267

on_partner_response() (negmas.sao.OnlyBestNegotiator method), 255

on_partner_response() (negmas.sao.PassThroughSAONegotiator method), 284

on_partner_response() (negmas.sao.SAONegotiator method), 218

on_partner_response() (negmas.sao.ToughNegotiator method), 249

on_production_failure() (negmas.apps.scml.DefaultGreedyManager method), 495

on_production_failure() (negmas.apps.scml.DoNothingFactoryManager method), 380

on_production_failure() (negmas.apps.scml.FactoryManager method), 372

on_production_failure() (negmas.apps.scml.GreedyFactoryManager method), 388

on_production_failure() (negmas.apps.scml.JavaDoNothingFactoryManager method), 410

on_production_failure() (negmas.apps.scml.JavaDummyMiddleMan method), 431

on_production_failure() (negmas.apps.scml.JavaFactoryManager method), 399

on_production_failure() (negmas.apps.scml.JavaGreedyFactoryManager method), 421

on_production_success() (negmas.apps.scml.DefaultGreedyManager method), 495

on_production_success() (negmas.apps.scml.FactoryManager method), 372

on_production_success() (negmas.apps.scml.GreedyFactoryManager method), 389

on_production_success() (negmas.apps.scml.JavaDoNothingFactoryManager method), 410

on_production_success() (negmas.apps.scml.JavaDummyMiddleMan method), 432

on_production_success() (negmas.apps.scml.JavaFactoryManager method), 399

on_production_success() (negmas.apps.scml.JavaGreedyFactoryManager method), 421

on_production_success() (negmas.apps.scml.JavaSCMLAgent method), 460

on_production_success() (negmas.genius.GeniusNegotiator method), 292

on_production_success() (negmas.negotiators.Controller method), 201

on_production_success() (negmas.negotiators.PassThroughNegotiator method), 205

on_production_success() (negmas.sao.AspirationNegotiator method), 243

on_production_success() (negmas.sao.JavaSAONegotiator method), 278

on_round_end() (negmas.negotiators.Controller method), 197

on_round_end() (negmas.negotiators.Negotiator method), 197

on_round_end() (negmas.sao.AspirationNegotiator method), 243

on_round_end() (negmas.sao.JavaSAONegotiator method), 278
on_round_end() (negmas.sao.LimitedOutcomesNegotiator method), 230
on_round_end() (negmas.sao.NaiveTitForTatNegotiator method), 261
on_round_end() (negmas.sao.PassThroughSAONegotiator method), 284
on_round_end() (negmas.sao.RandomNegotiator method), 224
on_round_end() (negmas.sao.SAOController method), 272
on_round_end() (negmas.sao.SAONegotiator method), 218
on_round_end() (negmas.sao.SAONegotiator method), 219
on_round_end() (negmas.sao.ToughNegotiator method), 249
on_round_start() (negmas.sao.AspirationNegotiator method), 243
on_round_start() (negmas.sao.JavaSAONegotiator method), 278
on_round_start() (negmas.sao.LimitedOutcomesAcceptor method), 236
on_round_start() (negmas.sao.LimitedOutcomesNegotiator method), 230
on_round_start() (negmas.sao.NaiveTitForTatNegotiator method), 261
on_round_start() (negmas.sao.NiceNegotiator method), 268
on_round_start() (negmas.sao.OnlyBestNegotiator method), 255
on_round_start() (negmas.sao.PassThroughSAONegotiator method), 284
on_round_start() (negmas.sao.RandomNegotiator method), 224
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 292
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 237
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 292
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 237
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 237
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 237
on_ufun_changed() (negmas.genius.GeniusNegotiator method), 237
onCashTransfer() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onCashTransfer() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onCashTransfer() (neg- mas.apps.scml.JavaFactoryManager method), 396
onCashTransfer() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418
onContractBreached() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onContractBreached() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onContractBreached() (neg- mas.apps.scml.JavaFactoryManager method), 396
onContractBreached() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418
onContractCancelled() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onContractCancelled() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onContractCancelled() (neg- mas.apps.scml.JavaFactoryManager method), 396
onContractCancelled() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418
onContractExecuted() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onContractExecuted() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onContractExecuted() (neg- mas.apps.scml.JavaFactoryManager method), 396
onContractExecuted() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418
onContractNullified() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onContractNullified() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onContractNullified() (neg- mas.apps.scml.JavaFactoryManager method), 397
onContractNullified() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418
onNegRequestAccepted() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onNegRequestAccepted() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onNegRequestAccepted() (neg- mas.apps.scml.JavaFactoryManager method), 396
onNegRequestAccepted() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418

OnlyBestNegotiator (class in negmas.sao), 251

onInventoryChange() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onInventoryChange() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onInventoryChange() (neg- mas.apps.scml.JavaFactoryManager method), 396
onInventoryChange() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418

onNegotiationFailure() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onNegotiationFailure() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onNegotiationFailure() (neg- mas.apps.scml.JavaFactoryManager method), 397
onNegotiationFailure() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418

onNegotiationSuccess() (neg- mas.apps.scml.JavaDoNothingFactoryManager method), 407
onNegotiationSuccess() (neg- mas.apps.scml.JavaDummyMiddleMan method), 429
onNegotiationSuccess() (neg- mas.apps.scml.JavaFactoryManager method), 397
onNegotiationSuccess() (neg- mas.apps.scml.JavaGreedyFactoryManager method), 418

644 Index
onNegRequestAccepted() (neg-
mas.apps.scml.JavaFactoryManager
method), 397
onNegRequestAccepted() (neg-
mas.apps.scml.JavaGreedyFactoryManager
method), 418
onNegRequestRejected() (neg-
mas.apps.scml.JavaDoNothingFactoryManager
method), 407
onNegRequestRejected() (neg-
mas.apps.scml.JavaDummyMiddleMan
method), 429
onNegRequestRejected() (neg-
mas.apps.scml.JavaFactoryManager
method), 397
onNegRequestRejected() (neg-
mas.apps.scml.JavaGreedyFactoryManager
method), 418
onNewCFP() (negmas.apps.scml.JavaDoNothingFactoryManager
method), 407
onNewCFP() (negmas.apps.scml.JavaDummyMiddleMan
method), 429
onNewCFP() (negmas.apps.scml.JavaFactoryManager
method), 397
onNewCFP() (negmas.apps.scml.JavaGreedyFactoryManager
method), 418
onNewReport() (neg-
mas.apps.scml.JavaDoNothingFactoryManager
method), 407
onNewReport() (neg-
mas.apps.scml.JavaDummyMiddleMan
method), 429
onNewReport() (neg-
mas.apps.scml.JavaFactoryManager
method), 397
onNewReport() (neg-
mas.apps.scml.JavaGreedyFactoryManager
method), 418
onProductionFailure() (neg-
mas.apps.scml.JavaDoNothingFactoryManager
method), 407
onProductionFailure() (neg-
mas.apps.scml.JavaDummyMiddleMan
method), 429
onProductionFailure() (neg-
mas.apps.scml.JavaFactoryManager
method), 397
onProductionFailure() (neg-
mas.apps.scml.JavaGreedyFactoryManager
method), 418
onProductionSuccess() (neg-
mas.apps.scml.JavaDoNothingFactoryManager
method), 407
onProductionSuccess() (neg-
mas.apps.scml.JavaDummyMiddleMan
method), 429
onProductionSuccess() (neg-
mas.apps.scml.JavaFactoryManager
method), 397
onProductionSuccess() (neg-
mas.apps.scml.JavaGreedyFactoryManager
method), 418
onRemoveCFP() (negmas.apps.scml.JavaDoNothingFactoryManager
method), 407
onRemoveCFP() (negmas.apps.scml.JavaDummyMiddleMan
method), 429
onRemoveCFP() (negmas.apps.scml.JavaFactoryManager
method), 397
onRemoveCFP() (negmas.apps.scml.JavaGreedyFactoryManager
method), 418
Outcome (in module negmas.outcomes), 605
outcome_as_dict() (in module neg-
mas.outcomes), 43
outcome_as_tuple() (in module neg-
mas.outcomes), 38
outcome_in_range() (in module neg-
mas.outcomes), 40
outcome_index() (neg-
mas.common.AgentMechanismInterface
method), 190
outcome_is_complete() (in module neg-
mas.outcomes), 39
outcome_is_valid() (in module neg-
mas.outcomes), 38
outcome_range (negmas.outcomes.Issue attribute),
outcome_range (negmas.outcomes.Issues attribute),
outcome_range_is_complete() (in module neg-
mas.outcomes), 40
outcome_range_is_valid() (in module neg-
mas.outcomes), 39
outcomeIndex() (neg-
mas.common._ShadowAgentMechanismInterface
method), 193
outcomes (negmas.apps.scml.CFP attribute), 349
outcomes (negmas.mechanisms.Mechanism attribute), 183
outcomes (negmas.sao.SAOMechanism attribute), 210
OutcomeType (class in negmas.mechanisms), 43
outputs (negmas.apps.scml.Process attribute), 338
overconsumption (neg-
mas.apps.scml.ConsumptionProfile attribute), 523
override (negmas.apps.scml.Job attribute), 344
pareto_frontier() (in module negmas.utilities),
pareto_frontier() (neg-
mas.mechanisms.Mechanism method),
propose() (negmas.sao.PassThroughSAONegotiator method), 285
propose() (negmas.sao.SAONegotiator method), 219
propose() (negmas.sao.ToughNegotiator method), 250
Protocol (in module negmas.mechanisms), 186
PYTHON_CLASS_IDENTIFIER (in module negmas.java), 604
Q
quantity (negmas.apps.scml.InputOutput attribute), 338
quantity (negmas.apps.scml.MissingInput attribute), 345
quantity (negmas.apps.scml.ProductManufacturingInfo attribute), 342
quantity_in_storage (negmas.apps.scml.ProductionNeed attribute), 345
quantity_to_buy (negmas.apps.scml.ProductionNeed attribute), 345
query() (negmas.situated.BulletinBoard method), 305
R
rand() (negmas.outcomes.Issue method), 51
rand() (negmas.outcomes.Issues method), 56
rand_invalid() (negmas.outcomes.Issue method), 51
rand_invalid() (negmas.outcomes.Issues method), 56
rand_outcomes() (negmas.outcomes.Issues method), 51
rand_valid() (negmas.outcomes.Issue method), 51
rand_valid() (negmas.outcomes.Issues method), 56
random() (negmas.apps.scml.ConsumptionProfile class method), 523
random() (negmas.apps.scml.MiningProfile class method), 540
random() (negmas.apps.scml.SCMLWorld class method), 510
random_negotiator() (negmas.genius.GeniusNegotiator class method), 293
random_outcomes() (negmas.common.AgentMechanismInterface method), 190
random_outcomes() (negmas.mechanisms.Mechanism method), 185
random_outcomes() (negmas.sao.SAO Mechanism method), 213
random_small() (negmas.apps.scml.SCMLWorld class method), 513
RandomNegotiator (class in negmas.sao), 220
randomOutcomes() (negmas.common._ShadowAgentMechanismInterface method), 193
RandomUtilityFunction (class in negmas.utilities), 171
ReactiveMiner (class in negmas.apps.scml), 540
read() (negmas.situated.BulletinBoard method), 305
read_config() (negmas.apps.scml.Consumer class method), 521
read_config() (negmas.apps.scml.DefaultBank class method), 442
read_config() (negmas.apps.scml.DefaultGreedyManager class method), 495
read_config() (negmas.apps.scml.DefaultInsuranceCompany class method), 448
read_config() (negmas.apps.scml.DoNothingFactoryManager class method), 380
read_config() (negmas.apps.scml.FactoryManager class method), 373
random() (negmas.apps.scml.GreedyFactoryManager class method), 389
read_config() (negmas.apps.scml.InsuranceCompany class method), 453
read_config() (negmas.apps.scml.JavaDoNothingFactoryManager class method), 410
read_config() (negmas.apps.scml.JavaDummyMiddleMan class method), 432
random() (negmas.apps.scml.JavaFactoryManager class method), 399
read_config() (negmas.apps.scml.JavaGreedyFactoryManager class method), 421
read_config() (negmas.apps.scml.Miner class method), 538
read_config() (negmas.apps.scml.ReactiveMiner class method), 546
random() (negmas.apps.scml.ScheduleDrivenConsumer class method), 530
read_config() (negmas.apps.scml.SCMLAgent class method), 460
random() (negmas.apps.scml.SCMLWorld class method), 513
read_config() (negmas.helpers.ConfigReader class method), 556
read_config() (negmas.situated.Agent class method), 302

Index 647
read_config() (negmas.situated.BulletinBoard class method), 306
read_config() (negmas.situated.World class method), 311
receive() (negmas.apps.scml.Factory method), 353
receive() (negmas.apps.scml.FactorySimulator method), 467
receive() (negmas.apps.scml.FastFactorySimulator method), 484
receive() (negmas.apps.scml.SlowFactorySimulator method), 475
receive_financial_reports() (negmas.apps.scml.SCMLAWI method), 361
read_config() (negmas.common.MechanismState attribute), 525
relative_time (negmas.apps.scml.SCMLAWI attribute), 358
relative_time (negmas.apps.scml.SCMLWorld attribute), 505
relative_time (negmas.common.MechanismState attribute), 191
relative_time (negmas.mechanisms.Mechanism attribute), 183
relative_time (negmas.sao.SAOMechanism attribute), 210
relative_time (negmas.sao.SAOSAOMechanism attribute), 207
relative_time (negmas.sao.SAOSAOMechanism attribute), 315
remaining_steps (negmas.situated.World attribute), 308
remaining_steps (negmas.apps.scml.SCMLWorld attribute), 505
remaining_steps (negmas.mechanisms.Mechanism attribute), 183
remaining_steps (negmas.sao.SAOMechanism attribute), 210
remaining_steps (negmas.situated.World attribute), 308
remaining_time (negmas.apps.scml.SCMLWorld attribute), 505
remaining_time (negmas.mechanisms.Mechanism attribute), 183
remaining_time (negmas.sao.SAOMechanism attribute), 210
remaining_time (negmas.situated.World attribute), 309
remove() (negmas.mechanisms.Mechanism method), 186
remove() (negmas.sao.SAOMechanism method), 213
remove() (negmas.situated.BulletinBoard method), 306
remove() (negmas.situated.BulletinBoard method), 311
remove_cfp() (negmas.apps.scml.SCMLAWI method), 361
remove_requirements() (negmas.mechanisms.Mechanism method), 186
remove_requirements() (negmas.sao.SAOMechanism method), 213
NegotiationRequest (class in negmas.situated), 317
reports_at() (negmas.apps.scml.SCMLAWI method), 361
reports_for() (negmas.apps.scml.SCMLAWI method), 361
request_negotiation() (negmas.apps.scml.Consumer method), 521
request_negotiation() (negmas.apps.scml.DefaultGreedyManager method), 495
request_negotiation() (neg-mas.apps.scml.DoNothingFactoryManager method), 381
request_negotiation() (neg-mas.apps.scml.FactoryManager method), 373
request_negotiation() (neg-mas.apps.scml.GreedyFactoryManager method), 389
request_negotiation() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 410
request_negotiation() (neg-mas.apps.scml.JavaFactoryManager method), 400
request_negotiation() (neg-mas.apps.scml.JavaGreedyFactoryManager method), 421
request_negotiation() (neg-mas.apps.scml.Miner method), 538
request_negotiation() (neg-mas.apps.scml.ReactiveMiner method), 546
request_negotiation() (neg-mas.apps.scml.ScheduleDrivenConsumer method), 530
request_negotiation() (neg-mas.apps.scml.SCMLAgent method), 460
request_negotiation() (neg-mas.apps.scml.SCMLAWI method), 362
request_negotiation_about() (neg-mas.apps.scml.SCMLAWI method), 362
request_negotiation_about() (neg-mas.apps.scml.SCMLWorld method), 513
request_negotiation_about() (neg-mas.situated.AgentWorldInterface method), 316
request_negotiation_about() (neg-mas.situated.World method), 311
requested_negotiations (neg-mas.apps.scml.FactoryManager attribute), 368
requested_negotiations (neg-mas.apps.scml.GreedyFactoryManager attribute), 384
requested_negotiations (neg-mas.apps.scml.InsuranceCompany attribute), 451
requested_negotiations (neg-mas.apps.scml.JavaDoNothingFactoryManager attribute), 404
requested_negotiations (neg-mas.apps.scml.JavaDummyMiddleMan attribute), 426
requested_negotiations (neg-mas.apps.scml.JavaFactoryManager attribute), 393
requested_negotiations (neg-mas.apps.scml.JavaGreedyFactoryManager attribute), 415
requested_negotiations (neg-mas.apps.scml.Miner attribute), 533
requested_negotiations (neg-mas.apps.scml.ReactiveMiner attribute), 542
requested_negotiations (neg-mas.apps.scml.ScheduleDrivenConsumer attribute), 526
requested_negotiations (neg-mas.apps.scml.SCMLAgent attribute), 456
requested_negotiations (neg-mas.situated.Agent attribute), 300
requestNegotiation() (neg-mas.apps.scml.JavaDoNothingFactoryManager method), 410
requestNegotiation() (neg-mas.apps.scml.JavaDummyMiddleMan method), 432
requestNegotiation() (neg-mas.apps.scml.JavaFactoryManager method), 400
requestNegotiation() (neg-mas.apps.scml.JavaGreedyFactoryManager method), 421
requirements (neg-mas.common.AgentMechanismInterface attribute), 189
requirements (negmas.mechanisms.Mechanism attribute), 183
requirements (negmas.sao.SAOMechanism attribute), 210
reserve() (negmas.apps.scml.FactorySimulator method), 467
reserve() (negmas.apps.scml.FactorySimulator method), 484
reserve() (negmas.apps.scml.SlowFactorySimulator method), 490
reserve() (negmas.apps.scml.DefaultFactoryManager attribute), 400
reserve() (negmas.apps.scml.DefaultGreedyManager attribute), 490
reserve() (negmas.apps.scml.DefaultInsuranceCompany attribute), 445
reserve() (negmas.apps.scml.DoNothingFactoryManager attribute), 376
method), 476
reserved_storage_at() (negmas.apps.scml.FactorySimulator
method), 467
reserved_storage_at() (negmas.apps.scml.FastFactorySimulator
method), 485
reserved_storage_at() (negmas.apps.scml.SlowFactorySimulator
method), 476
reserved_storage_to() (negmas.apps.scml.FactorySimulator
method), 467
reserved_storage_to() (negmas.apps.scml.FastFactorySimulator
method), 485
reserved_storage_to() (negmas.apps.scml.SlowFactorySimulator
method), 476
reserved_value (negmas.genius.GeniusNegotiator attribute), 289
reserved_value (negmas.negotiators.Negotiator attribute), 195
reserved_value (negmas.negotiators.PassThroughNegotiator
attribute), 203
reserved_value (negmas.sao.AspirationNegotiator attribute), 239
reserved_value (negmas.sao.JavaSAONegotiator attribute), 274
reserved_value (negmas.sao.LimitedOutcomesAcceptor attribute), 233
reserved_value (negmas.sao.LimitedOutcomesNegotiator attribute), 226
reserved_value (negmas.sao.NaiveTitForTatNegotiator attribute), 258
reserved_value (negmas.sao.NiceNegotiator attribute), 264
reserved_value (negmas.sao.OnlyBestNegotiator attribute), 252
reserved_value (negmas.sao.PassThroughSAONegotiator attribute), 281
reserved_value (negmas.sao.RandomNegotiator attribute), 221
reserved_value (negmas.sao.SAOController attribute), 273
reserved_value (negmas.sao.SAONegotiator attribute), 219
reset_to() (negmas.apps.scml.SlowFactorySimulator
method), 477
resolved_breaches (negmas.situated.World attribute), 505
respond() (negmas.genius.GeniusNegotiator
method), 309
respond() (negmas.sao.AspirationNegotiator
method), 244
respond() (negmas.sao.JavaSAONegotiator
method), 279
respond() (negmas.sao.LimitedOutcomesAcceptor
method), 237
respond() (negmas.sao.LimitedOutcomesNegotiator
method), 231
respond() (negmas.sao.NaiveTitForTatNegotiator
method), 262
respond() (negmas.sao.NiceNegotiator method), 269
respond() (negmas.sao.OnlyBestNegotiator
method), 256
respond() (negmas.sao.PassThroughSAONegotiator
method), 285
respond() (negmas.sao.RandomNegotiator
method), 224
respond() (negmas.sao.SAOController method), 273
respond() (negmas.sao.SAONegotiator method), 219
respond_() (negmas.genius.GeniusNegotiator
method), 250
respond_() (negmas.sao.AspirationNegotiator
method), 244
respond_() (negmas.sao.JavaSAONegotiator
method), 279
respond_() (negmas.sao.LimitedOutcomesAcceptor
method), 237
respond_() (negmas.sao.LimitedOutcomesNegotiator
method), 231
respond_() (negmas.sao.NaiveTitForTatNegotiator
method), 262
respond_() (negmas.sao.NiceNegotiator method), 269
respond_() (negmas.sao.OnlyBestNegotiator
method), 256
respond_() (negmas.sao.PassThroughSAONegotiator
method), 285
respond_() (negmas.sao.SAONegotiator method), 219
respond_() (negmas.sao.ToughNegotiator method), 250
respond_to_negotiation_request() (negmas.apps.scml.Bank
method), 442
respond_to_negotiation_request() (negmas.apps.scml.Consumer
method), 521
respond_to_negotiation_request() (negmas.apps.scml.DefaultBank
method), 438
respond_to_negotiation_request() (negmas.
respond_to_renegotiation_request()
(negmas.apps.scml.JavaFactoryManager method), 400
respond_to_renegotiation_request()
(negmas.apps.scml.JavaGreedyFactoryManager method), 422
respond_to_renegotiation_request()
(negmas.apps.scml.Miner method), 538
respond_to_renegotiation_request()
(negmas.apps.scml.ReactiveMiner method), 547
respond_to_renegotiation_request()
(negmas.apps.scml.ScheduleDrivenConsumer method), 531
respond_to_renegotiation_request()
(negmas.situated.Agent method), 303
respondToNegotiationRequest()
(negmas.apps.scml.JavaDoNothingFactoryManager method), 411
respondToNegotiationRequest()
(negmas.apps.scml.JavaDummyMiddleMan method), 432
respondToNegotiationRequest()
(negmas.apps.scml.JavaFactoryManager method), 400
respondToNegotiationRequest()
(negmas.apps.scml.JavaGreedyFactoryManager method), 421
respondToRenegotiationRequest()
(negmas.apps.scml.JavaDoNothingFactoryManager method), 411
respondToRenegotiationRequest()
(negmas.apps.scml.JavaDummyMiddleMan method), 432
respondToRenegotiationRequest()
(negmas.apps.scml.JavaFactoryManager method), 400
respondToRenegotiationRequest()
(negmas.apps.scml.JavaGreedyFactoryManager method), 421
ResponseType (class in negmas.outcomes), 45
resumption_cost
(negmas.apps.scml.ManufacturingProfile attribute), 340
resumption_cost
(negmas.apps.scml.ManufacturingProfileCompiled attribute), 341
ReturnCause (class in negmas.helpers), 553
rollback ()
(negmas.apps.scml.FactorySimulator method), 468
rollback ()
(negmas.apps.scml.FastFactorySimulator method), 485
rollback ()
(negmas.apps.scml.GreedyScheduler method), 500
rollback ()
(negmas.apps.scml.Scheduler method), 652
schedule (negmas.apps.scml.ConsumptionProfile attribute), 523
schedule (negmas.apps.scml.Factory method), 353
schedule (negmas.apps.scml.FactorySimulator method), 468
schedule (negmas.apps.scml.FastFactorySimulator method), 486
schedule (negmas.apps.scml.GreedyScheduler method), 500
schedule (negmas.apps.scml.Scheduler method), 498
schedule (negmas.apps.scml.SlowFactorySimulator method), 477
schedule_at () (negmas.apps.scml.ConsumptionProfile method), 523
schedule_contract () (negmas.apps.scml.GreedyScheduler method), 500
schedule_contracts () (negmas.apps.scml.GreedyScheduler method), 501
schedule_job () (negmas.apps.scml.SCMLAWI method), 362
schedule_production () (negmas.apps.scml.SCMLAWI method), 362
schedule_within () (negmas.apps.scml.ConsumptionProfile method), 523
ScheduleDrivenConsumer (class in negmas.apps.scml), 524
ScheduleInfo (class in negmas.apps.scml), 496
Scheduler (class in negmas.apps.scml), 497
SCMLAction (class in negmas.apps.scml), 347
SCMLAgent (class in negmas.apps.scml), 454
SCMLAgreement (class in negmas.apps.scml), 347
SCMLAWI (class in negmas.apps.scml), 354
SCMLWorld (class in negmas.apps.scml), 502
score_stats (negmas.tournaments.TournamentResults attribute), 575
sell() (negmas.apps.scml.Factory method), 353
sell() (negmas.apps.scml.FactorySimulator method), 468
sell() (negmas.apps.scml.FastFactorySimulator method), 486
sell() (negmas.apps.scml.SlowFactorySimulator method), 477
set_bulleen_board() (negmas.apps.scml.SCMLWorld method), 514
set_bulleen_board() (negmas.apps.scml.Scheduler method), 313
set_consumers () (negmas.apps.scml.SCMLWorld method), 514
set_factory_managers () (negmas.apps.scml.SCMLWorld method), 514
set_miners () (negmas.apps.scml.SCMLWorld method), 514

Index 653
set_processes() (negmas.apps.scml.SCMLWorld method), 514
set_products() (negmas.apps.scml.SCMLWorld method), 514
set_profiles() (negmas.apps.scml.ReactiveMiner method), 547
set_renegotiation_agenda() (negmas.apps.scml.Bank method), 443
set_renegotiation_agenda() (negmas.apps.scml.Consumer method), 522
set_renegotiation_agenda() (negmas.apps.scml.DefaultBank method), 531
set_renegotiation_agenda() (negmas.apps.scml.DefaultGreedyManager method), 496
set_renegotiation_agenda() (negmas.apps.scml.DefaultInsuranceCompany method), 449
set_renegotiation_agenda() (negmas.apps.scml.FactoryManager method), 373
set_renegotiation_agenda() (negmas.apps.scml.GreedyFactoryManager method), 389
set_renegotiation_agenda() (negmas.apps.scml.JavaDoNothingFactoryManager method), 411
set_renegotiation_agenda() (negmas.apps.scml.JavaDummyMiddleMan method), 433
set_renegotiation_agenda() (negmas.apps.scml.JavaFactoryManager method), 400
set_renegotiation_agenda() (negmas.apps.scml.JavaGreedyFactoryManager method), 422
set_renegotiation_agenda() (negmas.apps.scml.JavaDoNothingFactoryManager method), 411
setName() (negmas.apps.scml.JavaDoNothingFactoryManager method), 411
setName() (negmas.apps.scml.JavaDummyMiddleMan method), 433
setName() (negmas.apps.scml.JavaFactoryManager method), 400
setName() (negmas.apps.scml.JavaGreedyFactoryManager method), 422
setRenegotiationAgenda() (negmas.apps.scml.JavaDoNothingFactoryManager method), 411
setRenegotiationAgenda() (negmas.apps.scml.JavaDummyMiddleMan method), 433
setRenegotiationAgenda() (negmas.apps.scml.JavaFactoryManager method), 400
setRenegotiationAgenda() (negmas.apps.scml.JavaGreedyFactoryManager method), 422
setRenegotiationAgenda() (negmas.apps.scml.JavaDoNothingFactoryManager method), 411
short_type_name (negmas.apps.scml.Bank attribute), 440
short_type_name (negmas.apps.scml.Consumer attribute), 517
short_type_name (negmas.apps.scml.DefaultBank attribute), 435
short_type_name (negmas.apps.scml.DefaultGreedyManager attribute), 490
short_type_name (negmas.apps.scml.DefaultInsuranceCompany attribute), 445
short_type_name (negmas.apps.scml.JavaDoNothingFactoryManager attribute), 376
short_type_name (negmas.apps.scml.FactoryManager attribute), 368
| short_type_name | (negmas.apps.scml.GreedyFactoryManager attribute), 384 | sign_contract() (negmas.apps.scml.GreedyFactoryManager method), 390 |
| short_type_name | (negmas.apps.scml.InsuranceCompany attribute), 451 | sign_contract() (negmas.apps.scml.InsuranceCompany method), 454 |
| short_type_name | (negmas.apps.scml.JavaDoNothingFactoryManager attribute), 404 | sign_contract() (negmas.apps.scml.JavaDoNothingFactoryManager method), 411 |
| short_type_name | (negmas.apps.scml.JavaDummyMiddleMan attribute), 426 | sign_contract() (negmas.apps.scml.JavaDummyMiddleMan method), 433 |
| short_type_name | (negmas.apps.scml.JavaFactoryManager attribute), 394 | sign_contract() (negmas.apps.scml.JavaFactoryManager method), 400 |
| short_type_name | (negmas.apps.scml.JavaGreedyFactoryManager attribute), 415 | sign_contract() (negmas.apps.scml.JavaGreedyFactoryManager method), 422 |
| short_type_name | (negmas.apps.scml.Miner attribute), 533 | sign_contract() (negmas.apps.scml.Miner method), 539 |
| short_type_name | (negmas.apps.scml.ReactiveMiner attribute), 542 | sign_contract() (negmas.apps.scml.ReactiveMiner method), 547 |
| short_type_name | (negmas.apps.scml.ScheduleDrivenConsumer attribute), 526 | sign_contract() (negmas.apps.scml.ScheduleDrivenConsumer method), 531 |
| short_type_name | (negmas.apps.scml.SCALMagent attribute), 456 | sign_contract() (negmas.apps.scml.SCALMagent method), 461 |
| short_type_name | (negmas.situated.Agent attribute), 300 | sign_contract() (negmas.situated.Agent method), 303 |
| short_type_name | (negmas.situated.Entity attribute), 313 | signed_contracts (negmas.situated.Contract attribute), 505 |
| short_type_name | (negmas.situated.StatsMonitor attribute), 318 | signed_contracts (negmas.situated.World attribute), 309 |
| short_type_name | (negmas.situated.WorldMonitor attribute), 319 | signing_delay (negmas.situated.Contract attribute), 349 |
| shutdown() | (negmas.java.JNegmasGateway class method), 566 | signedContract() (negmas.java.JNegmasGateway method), 422 |
| sign_contract() | (negmas.apps.scml.Bank method), 443 | signed_at (negmas.situated.Contract attribute), 297 |
| sign_contract() | (negmas.apps.scml.Consumer method), 522 | signed_contracts (negmas.situated.Contract attribute), 305 |
| sign_contract() | (negmas.apps.scml.DefaultBank method), 438 | signed_contracts (negmas.situated.World attribute), 309 |
| sign_contract() | (negmas.apps.scml.DefaultGreedyManager method), 496 | signing_delay (negmas.situated.Contract attribute), 349 |
| sign_contract() | (negmas.apps.scml.DoNothingFactoryManager method), 381 | SimpleTitForTatNegotiator (in module negmas.sao), 263 |
| sign_contract() | (negmas.apps.scml.FactoryManager method), 374 | SlowFactorySimulator (class in negmas.apps.scml), 470 |
| snake_case() | (in module negmas.helpers), 550 | |

Index 655
start (negmas.apps.scml.ScheduleInfo attribute), 496
start_java_side() (negmas.java.JNegmasGateway class method), 566
started (negmas.common.MechanismState attribute), 191
started (negmas.sao.SAOSState attribute), 208
started_on_or_after() (negmas.scml.RunningCommandInfo method), 340
state (negmas.scml.SCMLAWI attribute), 358
state (negmas.common.AgentMechanismInterface attribute), 190
state (negmas.mechanisms.Mechanism attribute), 183
state (negmas.sao.SAOMechanism attribute), 210
state (negmas.situated.AgentWorldInterface attribute), 315
stats (negmas.scml.SCMLWorld attribute), 506
stats (negmas.situated.World attribute), 309
StatsMonitor (class in negmas.situated), 317
step (negmas.scml.InputOutput attribute), 338
step (negmas.scml.ProductionNeed attribute), 345
step (negmas.scml.ProductManufacturingInfo attribute), 342
step (negmas.scml.RunningCommandInfo attribute), 339
step (negmas.common.MechanismState attribute), 191
step (negmas.sao.SAOSState attribute), 208
step (negmas.situated.Breach attribute), 298
step() (negmas.scml.FactoryMethod), 522
step() (negmas.scml.DefaultBankMethod), 438
step() (negmas.scml.DefaultGreedyManagerMethod), 496
step() (negmas.scml.DefaultInsuranceCompanyMethod), 449
step() (negmas.scml.DoNothingFactoryManagerMethod), 381
step() (negmas.scml.FactoryMethod), 353
step() (negmas.scml.InsuranceCompanyMethod), 374
step() (negmas.scml.GreedyFactoryManagerMethod), 390
step() (negmas.scml.InsuranceCompanyMethod), 454
step() (negmas.scml.JavaDoNothingFactoryManagerMethod), 411
step() (negmas.scml.JavaFactoryManagerMethod), 401
step() (negmas.scml.JavaGreedyFactoryManagerMethod), 422
step() (negmas.scml.MinorMethod), 539
step() (negmas.scml.ReactiveFactoryMethod), 547
step() (negmas.scml.ScheduleDrivenConsumerMethod), 547
step() (negmas.scml.SCMLAgentMethod), 461
step() (negmas.scml.SCMLWorldMethod), 515
step() (negmas.mechanisms.MechanismMethod), 186
step() (negmas.sao.SAOMechanismMethod), 213
step() (negmas.situated.AgentMethod), 303
step() (negmas.situated.StatsMonitorMethod), 318
step() (negmas.situated.WorldMethod), 313
step() (negmas.situated.WorldMonitorMethod), 319
step() (negmas.scml.BankMethod), 443
step() (negmas.scml.ConsumerMethod), 522
step() (negmas.scml.DefaultBankMethod), 438
step() (negmas.scml.DefaultGreedyManagerMethod), 496
step() (negmas.scml.DefaultInsuranceCompanyMethod), 449
step() (negmas.scml.DoNothingFactoryManagerMethod), 381
step() (negmas.scml.FactoryMethod), 374
step() (negmas.scml.GreedyFactoryManagerMethod), 390
step() (negmas.scml.InsuranceCompanyMethod), 454
step() (negmas.scml.JavaDoNothingFactoryManagerMethod), 411
step() (negmas.scml.JavaDummyMiddleManMethod), 433
step() (negmas.scml.JavaFactoryManagerMethod), 401
step() (negmas.scml.JavaGreedyFactoryManagerMethod), 422

Index
Stop_production() (negmas.apps.scml.SCMLAWI method), 363
class method

timedout (negmas.sao.SAOSState attribute), 208

TIMEOUT (negmas.helpers.ReturnCause attribute), 554
to_be_signed_at (negmas.situated.Contract attribute), 297
to_dict() (in module negmas.scml), 563
to_flat_dict() (in module negmas.java), 564
to_genius() (negmas.outcomes.Issue attribute class method), 52
to_genius() (negmas.utils.ComplexNonlinearUtilityFunction class method), 151
to_genius() (negmas.utils.ComplexWeightedUtilityFunction class method), 143
to_genius() (negmas.utils.ConstUFun class method), 73
to_genius() (negmas.utils.ExpDiscountedUFun class method), 89
to_genius() (negmas.utils.HyperRectangleUtilityFunction class method), 126
to_genius() (negmas.utils.IPUtilityFunction class method), 162
to_genius() (negmas.utils.JavaUtilityFunction class method), 170
to_genius() (negmas.utils.LinDiscountedUFun class method), 81
to_genius() (negmas.utils.LinearUtilityAggregationFunction class method), 107
to_genius() (negmas.utils.MappingUtilityFunction class method), 98
to_genius() (negmas.utils.NonlinearHyperRectangleUtilityFunction class method), 135
to_genius() (negmas.utils.NonLinearUtilityAggregationFunction class method), 117
to_genius() (negmas.utils.RandomUtilityFunction class method), 78
to_genius() (negmas.utils.UtilityFunction class method), 65
to_java() (in module negmas.java), 562
to_java() (negmas.apps.scml.CFP method), 350
to_java() (negmas.common._ShadowAgentMechanismInterface method), 193
to_java() (negmas.outcomes.Issue method), 53
to_xml_str() (negmas.outcomes.Issue class method), 53
to_xml_str() (negmas.utils.ComplexNonlinearUtilityFunction class method), 151
to_xml_str() (negmas.utils.ComplexWeightedUtilityFunction class method), 143

tau_q (negmas.apps.scml.ConsumptionProfile attribute), 523
tau_q (negmas.apps.scml.MiningProfile attribute), 539
tau_t (negmas.apps.scml.MiningProfile attribute), 539
tau_u (negmas.apps.scml.ConsumptionProfile attribute), 523
tau_u (negmas.apps.scml.MiningProfile attribute), 539

temporary_transaction() (in module negmas.scml), 327
test() (negmas.genius.GeniusNegotiator method), 294
time (negmas.apps.scml.Job attribute), 344
time (negmas.apps.scml.SCMALocation attribute), 348
time (negmas.apps.scml.SCMWorld attribute), 506
time (negmas.common.MechanismState attribute), 191
time (negmas.methods.Mechanism attribute), 183
time (negmas.sao.SAOMechanism attribute), 210
time (negmas.sao.SAOSState attribute), 208
time (negmas.situated.World attribute), 309
time_limit (negmas.methods.Mechanism attribute), 183
time_limit (negmas.sao.SAOMechanism attribute), 210
timedout (negmas.methods.MechanismState attribute), 191
timedout (negmas.methods.MechanismRoundResult attribute), 187
to_xml_str() (negmas.utilities.ConstUFun class method), 73

to_xml_str() (negmas.utilities.ExpDiscountedUFun class method), 89

to_xml_str() (negmas.utilities.HyperRectangleUtilityFunction class method), 126

to_xml_str() (negmas.utilities.IPUtilityFunction class method), 162

to_xml_str() (negmas.utilities.JavaUtilityFunction class method), 171

to_xml_str() (negmas.utilities.LinDiscountedUFun class method), 81

to_xml_str() (negmas.utilities.LinearUtilityAggregationFunction class method), 107

to_xml_str() (negmas.utilities.MappingUtilityFunction class method), 98

to_xml_str() (negmas.utilities.NonlinearHyperRectangleUtilityFunction class method), 135

to_xml_str() (negmas.utilities.NonLinearUtilityAggregationFunction class method), 117

to_xml_str() (negmas.utilities.RandomUtilityFunction class method), 178

total_balance (negmas.apps.scml.Factory class attribute), 353

total_cost() (negmas.apps.scml.GreedyScheduler method), 501

total_storage (negmas.apps.scml.FactorySimulator method), 469

total_storage() (negmas.apps.scml.FastFactorySimulator method), 487

total_storage() (negmas.apps.scml.SlowFactorySimulator method), 478

total_storage_to() (negmas.apps.scml.FastFactorySimulator method), 487

total_storage_to() (negmas.apps.scml.SlowFactorySimulator method), 479

total_unit_cost () (negmas.utilities.UtilityFunction attribute), 60

total_utility() (negmas.apps.scml.DefaultGreedyManager method), 496

total_utility() (negmas.apps.scml.GreedyFactoryManager method), 390

total_utility() (negmas.apps.scml.FastFactorySimulator method), 487

total_utility() (negmas.apps.scml.SlowFactorySimulator method), 479

total_utility() (negmas.apps.scml.FactorySimulator method), 470

tournament() (in module negmas.tournaments), 574

ToughNegotiator (class in negmas.sao), 244

tournament() (in module negmas.sao), 496

tournament() (in module negmas.sao), 390

transaction() (in module negmas.apps.scml), 327

transport_from() (negmas.apps.scml.Factory method), 354

transport_to() (negmas.apps.scml.Factory method), 354

transport_to() (negmas.apps.scml.FactorySimulator method), 470

transport_to() (negmas.apps.scml.FastFactorySimulator method), 487

transport_to() (negmas.apps.scml.SlowFactorySimulator method), 479

ttest (negmas.tournaments.TournamentResults attribute), 575

type (negmas.events.Event attribute), 321

type (negmas.events.Notification attribute), 322

type (negmas.outcomes.Issue attribute), 47

type (negmas.utilities.ComplexNonlinearUtilityFunction attribute), 145

type (negmas.utilities.ComplexWeightedUtilityFunction attribute), 137

type (negmas.utilities.ConstUFun attribute), 68

type (negmas.utilities.ExpDiscountedUFun attribute), 83

type (negmas.utilities.HyperRectangleUtilityFunction attribute), 121

type (negmas.utilities.IPUtilityFunction attribute), 154

type (negmas.utilities.JavaUtilityFunction attribute), 164

type (negmas.utilities.LinDiscountedUFun attribute), 75

type (negmas.utilities.LinearUtilityAggregationFunction attribute), 101

type (negmas.utilities.MappingUtilityFunction attribute), 92

type (negmas.utilities.NonlinearHyperRectangleUtilityFunction attribute), 129

type (negmas.utilities.NonLinearUtilityAggregationFunction attribute), 111

type (negmas.utilities.RandomUtilityFunction attribute), 173

Index
Index 659

U

underconsumption (negmas.apps.scml.ConsumptionProfile attribute), 523

unhide_funds() (negmas.apps.scml.Factory method), 354
unhide_funds() (negmas.apps.scml.SCMLAWI method), 363

unhide_inventory() (negmas.apps.scml.SCMLAWI method), 363
unhide_product() (negmas.apps.scml.Factory method), 354
unique_name() (in module negmas.helpers), 551
unit_time() (negmas.apps.scml.GreedyScheduler method), 501

unregister_interest() (negmas.apps.scml.SCMLAWI method), 363
unregister_interest() (negmas.apps.scml.SCMLWorld method), 515
unregister_stats_monitor() (negmas.apps.scml.SCMLWorld method), 515
unregister_stats_monitor() (negmas.situated.World method), 313
unregister_world_monitor() (negmas.apps.scml.SCMLWorld method), 515
unregister_world_monitor() (negmas.situated.World method), 313

unresolved_breaches (negmas.apps.scml.SCMLWorld attribute), 506

unsigned_contracts (negmas.apps.scml.Bank attribute), 440
unsigned_contracts (negmas.apps.scml.Consumer attribute), 517
unsigned_contracts (negmas.apps.scml.DefaultBank attribute), 435
unsigned_contracts (negmas.apps.scml.DefaultGreedyManager attribute), 490
unsigned_contracts (negmas.apps.scml.DefaultInsuranceCompany attribute), 445
unsigned_contracts (negmas.apps.scml.DoNothingFactoryManager attribute), 376
unsigned_contracts (negmas.apps.scml.FactoryManager attribute), 368
unsigned_contracts (negmas.apps.scml.GreedyFactoryManager attribute), 384
unsigned_contracts (negmas.apps.scml.JavaDoNothingFactoryManager attribute), 404
unsigned_contracts (negmas.apps.scml.JavaDummyMiddleMan attribute), 426
unsigned_contracts (negmas.apps.scml.JavaFactoryManager attribute), 394
unsigned_contracts (negmas.apps.scml.JavaGreedyFactoryManager attribute), 415
unsigned_contracts (negmas.apps.scml.Miner attribute), 534
unsigned_contracts (negmas.apps.scml.ReactiveMiner attribute), 490
unsigned_contracts (negmas.apps.scml.ScheduleDrivenConsumer attribute), 526
unsigned_contracts (negmas.apps.scml.SCMLAgent attribute), 456
unsigned_contracts (negmas.situated.Agent attribute), 300
unsigned_contracts (negmas.situated.BulletinBoard attribute), 304
unsigned_contracts (negmas.situated.Entity attribute), 313
unsigned_contracts (negmas.situated.StatsMonitor attribute), 318
unsigned_contracts (negmas.situated.WorldMonitor attribute), 319

types (negmas.outcomes.Issues attribute), 55
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Module/Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned_contracts</td>
<td>negmas.apps.scml.ScheduleDrivenConsumer attribute, 526</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>negmas.apps.scml.SCMLAgent attribute, 456</td>
</tr>
<tr>
<td>unsigned_contracts</td>
<td>negmas.situated.Agent attribute, 300</td>
</tr>
<tr>
<td>updates</td>
<td>negmas.apps.scml.RunningCommandInfo attribute, 339</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.genius.GenieNegotiator attribute, 289</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.negotiators.PassThroughNegotiator attribute, 203</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.AspirationNegotiator attribute, 239</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.JavaSAONegotiator attribute, 274</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.LimitedOutcomesAcceptor attribute, 233</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.LimitedOutcomesNegotiator attribute, 226</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.NaiveTitForTatNegotiator attribute, 258</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.NiceNegotiator attribute, 265</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.OnlyBestNegotiator attribute, 252</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.PassThroughSAONegotiator attribute, 281</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.RandomNegotiator attribute, 221</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.SAONegotiator attribute, 215</td>
</tr>
<tr>
<td>utility_function</td>
<td>negmas.sao.ToughNegotiator attribute, 246</td>
</tr>
<tr>
<td>UtilityFunction</td>
<td>class in negmas.utilities</td>
</tr>
<tr>
<td>UtilityValue</td>
<td>in module negmas.utilities</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.Bank attribute)</td>
<td>440</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.Consumer attribute)</td>
<td>517</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.DefaultBank attribute)</td>
<td>435</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.DefaultGreedyManager attribute)</td>
<td>490</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.DefaultInsuranceCompany attribute)</td>
<td>445</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.DoNothingFactoryManager attribute)</td>
<td>376</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.FactoryManager attribute)</td>
<td>369</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.GreedyFactoryManager attribute)</td>
<td>384</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.InsuranceCompany attribute)</td>
<td>451</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.JavaDoNothingFactoryManager attribute)</td>
<td>404</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.JavaDummyMiddleMan attribute)</td>
<td>426</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.JavaFactoryManager attribute)</td>
<td>394</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.JavaGreedyFactoryManager attribute)</td>
<td>415</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.Miner attribute)</td>
<td>534</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.ReactiveMiner attribute)</td>
<td>542</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.ScheduleDrivenConsumer attribute)</td>
<td>526</td>
</tr>
<tr>
<td>uuid (negmas.apps.scml.SCMLAgent attribute)</td>
<td>456</td>
</tr>
<tr>
<td>uuid (negmas.common.NamedObject attribute)</td>
<td>189</td>
</tr>
<tr>
<td>uuid (negmas.events.Notifier attribute)</td>
<td>322</td>
</tr>
<tr>
<td>uuid (negmas.genius.GenieNegotiator attribute)</td>
<td>289</td>
</tr>
<tr>
<td>uuid (negmas.mechanisms.Mechanism attribute)</td>
<td>183</td>
</tr>
<tr>
<td>uuid (negmas.negotiators.Controller attribute)</td>
<td>199</td>
</tr>
<tr>
<td>uuid (negmas.negotiators.Negotiator attribute)</td>
<td>195</td>
</tr>
<tr>
<td>uuid (negmas.negotiators.PassThroughNegotiator attribute)</td>
<td>203</td>
</tr>
<tr>
<td>uuid (negmas.outcomes.Issue attribute)</td>
<td>47</td>
</tr>
<tr>
<td>uuid (negmas.sao.AspirationNegotiator attribute)</td>
<td>239</td>
</tr>
<tr>
<td>uuid (negmas.sao.JavaSAONegotiator attribute)</td>
<td>274</td>
</tr>
<tr>
<td>uuid (negmas.sao.LimitedOutcomesAcceptor attribute)</td>
<td>233</td>
</tr>
<tr>
<td>uuid (negmas.sao.LimitedOutcomesNegotiator attribute)</td>
<td>226</td>
</tr>
<tr>
<td>uuid (negmas.sao.NaiveTitForTatNegotiator attribute)</td>
<td>258</td>
</tr>
<tr>
<td>uuid (negmas.sao.NiceNegotiator attribute)</td>
<td>265</td>
</tr>
<tr>
<td>uuid (negmas.sao.OnlyBestNegotiator attribute)</td>
<td>252</td>
</tr>
<tr>
<td>uuid (negmas.sao.PassThroughSAONegotiator attribute)</td>
<td>281</td>
</tr>
<tr>
<td>uuid (negmas.sao.RandomNegotiator attribute)</td>
<td>221</td>
</tr>
<tr>
<td>uuid (negmas.sao.SAONegotiator attribute)</td>
<td>215</td>
</tr>
<tr>
<td>uuid (negmas.sao.ToughNegotiator attribute)</td>
<td>246</td>
</tr>
<tr>
<td>uuid (negmas.situated.Agent attribute)</td>
<td>300</td>
</tr>
<tr>
<td>uuid (negmas.situated.BulletinBoard attribute)</td>
<td>304</td>
</tr>
<tr>
<td>uuid (negmas.situated.Entity attribute)</td>
<td>313</td>
</tr>
<tr>
<td>uuid (negmas.situated.StatsMonitor attribute)</td>
<td>318</td>
</tr>
<tr>
<td>uuid (negmas.situated.WorldMonitor attribute)</td>
<td>319</td>
</tr>
<tr>
<td>uuid (negmas.utilities.ComplexNonlinearUtilityFunction attribute)</td>
<td>146</td>
</tr>
<tr>
<td>uuid (negmas.utilities.ComplexWeightedUtilityFunction attribute)</td>
<td>146</td>
</tr>
</tbody>
</table>
xml() (negmas.utilities.HyperRectangleUtilityFunction method), 127
xml() (negmas.utilities.IPUtilityFunction method), 163
xml() (negmas.utilities.JavaUtilityFunction method), 171
xml() (negmas.utilities.LinDiscountedUFun method), 82
xml() (negmas.utilities.LinearUtilityAggregationFunction method), 108
xml() (negmas.utilities.MappingUtilityFunction method), 99
xml() (negmas.utilities.NonlinearHyperRectangleUtilityFunction method), 136
xml() (negmas.utilities.NonLinearUtilityAggregationFunction method), 118
xml() (negmas.utilities.RandomUtilityFunction method), 179
xml() (negmas.utilities.UtilityFunction method), 66

Z

zero_runs() (in module negmas.apps.scml), 335