Ocarina Documentation

Release 2017.x)

Julien Delange, Jerome Hugues, Bechir Zalila

Apr 08, 2019

Contents

CHAPTER 1

About This Guide

1.1 About this Guide

This guide describes the use of Ocarina, a compiler for the AADL.

It presents the features of the compiler, related APIs and tools; and details how to use them to build and exploit AADL models.

It also details model transformations of AADL models onto Petri Net models

Companion documents describe other add-ons for Ocarina:

- PolyORB-HI/Ada, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets Ada targets: Native or bare board runtimes;
- PolyORB-HI/C, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets C targets: POSIX and RT-POSIX systems, RTEMS.

1.2 Document Conventions

This document uses the following conventions:

Note: This is just a note, for your information.

Warning: This is a warning, something you should take care of.

A filename or a path to a filename is displayed like this: `/path/to/filename.ext`

A command to type in the shell is displayed like this: **`command --arguments`**

A sample of code is illustrated like this:

```
First Line of Code
Second Line of Code
...
```

1.3 Copyright Information

CHAPTER 2

Introduction

2.1 About Ocarina

Ocarina is an application that can be used to analyze and build applications from AADL descriptions. Because of its modular architecture, Ocarina can also be used to add AADL functions to existing applications. Ocarina supports the AADL 1.0 and AADLv2 standards and proposes the following features :

- Parsing and pretty printing of AADL models
- Semantics checks
- Code generation, with the following code generators
  - PolyORB-HI/Ada, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets Ada targets: Native or bare board runtimes;
  - PolyORB-HI/C, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets C targets: POSIX systems, RTEMS;
  - POK, a partioned operating system compliant with the ARINC653 standard.
- Model checking using Petri nets;
- Computation of Worst-Case Execution Time using the Bound-T tool from Tidorum Ltd.;
- REAL, Requirement Enforcement and Analysis Language, an AADLv2 annex language to evaluate properties and metrics of AADLv2 architectural models;
- Scheduling analysis of AADL models, with a gateway to the Cheddar scheduling analysis tool from the Université de Bretagne Occidentale, and MAST from the University of Cantabria

2.2 Licence

Ocarina is distributed under the GPLv3 plus runtime exception.

The GPLv3 plus runtime exception guarantees that Ocarina, but also the code it generates can be distributed under customer-specific terms and conditions. Specifically, the licence ensures that you can generate proprietary, classified, or otherwise restricted executables.

2.3 About AADL

The "Architecture Analysis and Design Language" AADL is a textual and graphical language for model-based engineering of embedded real-time systems. It has been published as SAE Standard AS5506B (http://standards.sae.org/as5506b/). AADL is used to design and analyze the software and hardware architectures of embedded real-time systems.

AADL allows for the description of both software and hardware parts of a system. It focuses on the definition of clear block interfaces, and separates the implementations from these interfaces. It can be expressed using both a graphical and a textual syntax. From the description of these blocks, one can build an assembly of blocks that represent the full system. To take into account the multiple ways to connect components, the AADL defines different connection patterns: subcomponent, connection, and binding.

An AADL model can incorporate non-architectural elements: embedded or real-time characteristics of the components (such as execution time, memory footprint), behavioral descriptions. Hence it is possible to use AADL as a back- bone to describe all the aspects of a system. Let us review all these elements:

An AADL description is made of components. The AADL standard defines software components (data, thread, thread group, subprogram, process) and execution plat- form components (memory, bus, processor, device, virtual processor, virtual bus) and hybrid components (system). Each Component category describe well identified elements of the actual architecture, using the same vocabulary of system or software engineering:

- Subprograms model procedures like in C or Ada.
- Threads model the active part of an application (such as POSIX threads). AADL threads may have multiple operational modes. Each mode may describe a different behavior and property values for the thread.
- Processes are memory spaces that contain the threads. Thread groups are used to create a hierarchy among threads.
- Processors model microprocessors and a minimal operating system (mainly a scheduler).
- Memories model hard disks, RAMs, buses model all kinds of networks, wires, devices model sensors, ...
- Virtual bus and Virtual processor models "virtual" hardware components. A virtual bus is a communication channel on top of a physical bus (e.g. TCP/IP over Ethernet); a virtual processor denotes a dedicated scheduling domain inside a processor (e.g. an ARINC653 partition running on a processor).

Unlike other components, Systems do not represent anything concrete; they combine building blocks to help structure the description as a set of nested components.

Packages add the notion of namespaces to help structuring the models. Abstracts model partially defined components, to be refined during the modeling process.

Component declarations have to be instantiated into subcomponents of other components in order to model system architecture. At the top-level, a system contains all the component instances. Most components can have subcomponents, so that an AADL description is hierarchical. A complete AADL description must provide a top-most level system that will contain certain kind of components (processor, process, bus, device, abstract and memory), thus providing the root of the architecture tree. The architecture in itself is the instantiation of this system, which is called the root system.

The interface of a component is called component type. It provides features (e.g. communication ports). Components communicate one with another by connecting their features. To a given component type correspond zero or several implementations. Each of them describes the internals of the components: subcomponents, connections between those subcomponents, etc.

An implementation of a thread or a subprogram can specify call sequences to other subprograms, thus describing the execution flows in the architecture. Since there can be different implementations of a given component type, it is possible to select the actual components to put into the architecture, without having to change the other components, thus providing a convenient approach to configure applications.

The AADL defines the notion of properties that can be attached to most elements (components, connections, features, etc.). Properties are typed attributes that specify constraints or characteristics that apply to the elements of the architecture: clock frequency of a processor, execution time of a thread, bandwidth of a bus, ... Some standard properties are defined, e.g. for timing aspects; but it is possible to define new properties for different analysis (e.g. to define particular security policies).

AADL is a language, with different representations. A textual representation provides a comprehensive view of all details of a system, and graphical if one want to hide some details, and allow for a quick navigation in multiple dimensions. In the following, we illustrate both notations. Let us note that AADL can also be expressed as a UML model following the MARTE profile.

The concepts behind AADL are those typical to the construction of embedded systems, following a component- based approach: blocks with clear interfaces and properties are defined, and compose to form the complete system. Besides, the language is defined by a companion standard document that documents legality rules for component assemblies, its static and execution semantics.

The following figure illustrates a complete space system, used as a demonstrator during the ASSERT project. It illustrates how software and hardware concerns can be separately developed and then combined in a complete model.

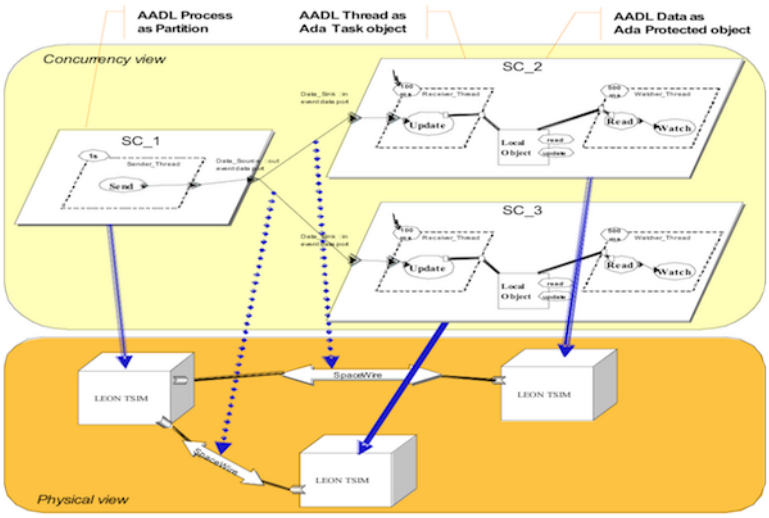

Fig. 1: ASSERT MPC Case study

2.4 Ocarina concepts

Ocarina uses the following set of definitions :

- A *scenario file* is a specific AADL system that controls the behavior of Ocarina through various properties, see *Scenario files*.
- A *root system* is the root of an AADL model; it is a system implementation without feature. As a closed system, it has definitions required for complete processing by Ocarina: processors, threads, processes, etc.

CHAPTER 3

Installation

3.1 Supported platforms

Ocarina has been compiled and successfully tested on the following platforms:

- GNU/Linux
- Mac OS X
- Windows

Note: Ocarina should compile and run on every target for which GNAT is available.

3.2 Build requirements

An Ada compiler:

- GNAT Pro, GNAT GPL or FSF/GCC with Ada back-end

Note: per construction, the macro configure used to find your GNAT compiler looks first to the executable gnatgcc, then adagcc and finally to gcc to find out which Ada compiler to use. You should be very careful with your path and binaries if you have multiple GNAT versions installed. See below explanations on the ADA environment variable if you need to override the default guess.

Note: Ocarina requires at least GCC/FSF 7 or GNAT GPL 2016 or more recent.

- autoconf, automake, GNU Make, python

Optional components:

- GNATColl for the Ocarina Python bindings
- Sphinx and the sphinx-bootstrap-theme to build the documentation, and a full valid LaTeX installation
- Bound-T for the WCET analysis (`bound_t` backend)
- Cheddar for scheduling analysis (`cheddar` backend)
- MAST for scheduling analysis (`mast` backend)
- RTOS supported by one of the Ocarina runtimes

3.3 Semi-automated build instructions

The ocarina-build repository proposes a script, *build_ocarina.sh*, to get source code, compile and test Ocarina.

It relies on bash constructs to coordinate various activities to:

- fetch Ocarina source, with its runtimes PolyORB-HI/Ada and PolyORB-HI/C
- compile Ocarina, and install it in a local directory
- run Ocarina testsuites, and eventually collect coverage metrics

To install this script, simply clone the repository and run the script. Use *build_ocarina.sh -h* to access its help.

- The following command gets a fresh copy of Ocarina source code:

```
% ./build_ocarina.sh -s -u
```

- The following command compiles and installs Ocarina:

```
% ./build_ocarina.sh -b
```

3.4 Manual build instructions

To compile and install Ocarina, execute in a shell:

```
% ./configure [some options]
% make               (or gmake if your make is not GNU make)
% make install       (ditto)
```

This will install files in standard locations. If you want to choose another prefix than */usr/local*, give configure use *–prefix* argument

Note: you MUST use GNU make to compile this software.

Note: If you modify source files, build Ocarina after a checkout or make distclean, or the directory hierarchy of the source files, you should re-generate autoconf and automake files (configure, Makefile.in. . .); to do this, from the main directory, run:

```
./support/reconfig
```

Note: To install the PolyORB/HI runtimes, you may use the script get_runtimes.sh. It will install required ressources in the Ocarina source tree:

```
./support/get_runtimes.sh po_hi_ada po_hi_c
```

3.5 Build options

Available options for the configure script include:

- *–enable-doc*: to build the documentation

Note: You must first install Sphinx and the sphinx-bootstrap-theme

- *–enable-shared*: to build shared libraries
- *–enable-debug*: enable debugging information generation and supplementary runtime checks. Note that this option has a significant space and time cost, and is not recommended for production use.
- *–enable-python*: to build the Python bindings.

Note: This option requires GNATColl to be installed, and Ocarina built with shared libraries support.

- *–with-ocarina-runtimes=x*: enable building Ocarina along with the requested runtimes. x is a set of valid runtimes located in the resources/runtimes directory. x is case insensitive. Examples of use:
- *–with-ocarina-runtimes=all*: compile Ocarina along with all the runtimes. All the Ocarina runtimes MUST be located in the resources/runtimes directory.
- *–with-ocarina-runtimes="polyorb-hi-c PolyORB-HI-Ada"*: compile Ocarina along with the PolyORB-HI-Ada and the PolyORB-HI-C runtimes.

Note: The runtime directories (e.g. `polyorb-hi-ada` or `polyorb-hi-c` MUST exist in the resources/runtimes directory.

No option: compile Ocarina along with all the runtimes found in the resources/runtimes directory.

For more details on available options, one may use the *–help* flag.

The following environment variables can be used to override configure's guess at what compilers to use:

- *CC*: the C compiler
- *ADA*: the Ada 95 compiler (e.g. gcc, gnatgcc or adagcc)

For example, if you have two versions of GNAT installed and available in your PATH, and configure picks the wrong one, you can indicate what compiler should be used with the following syntax:

```
% ADA=/path/to/good/compiler/gcc ./configure [options]
```

Ocarina will be compiled with GNAT build host's configuration, including run-time library. You may override this setting using *ADA_INCLUDE_PATH* and *ADA_OBJECTS_PATH* environment variables. See GNAT User's Guide for more details.

Note: Developers building Ocarina from the version control repository who need to rebuild the configure and Makefile.in files should use the script support/reconfig for this purpose. This should be done after each update from the repository. In addition to the requirements above, they will need autoconf 2.57 or newer, automake 1.6.3 or newer.

3.6 Windows-specific options

Ocarina relies on autotools script to compile, and then on Python for testing and running regression testing. Such setting is unusual for Windows and requires additional tools.

The recommended set of tools for compiling Ocarina under Windows (tested on Windows 7, as of 2018/02/27) is to

- install MSYS2, and use its terminal for running all compilation scripts
- install autoconf, automake, Python and GNU Make
- install *mingw-w64-x86_64-gcc-ada* package, it has GNAT front-end

---

Note: It is highly recommended to rely on the *build_ocarina.sh* script for the Windows platform.

CHAPTER 4

Usage

4.1 Ocarina command-line

Ocarina has a rich command-line interface, covering all required steps to parse, instantiate, analyze or generate code from AADL models.

-h, --help
Display help and exit

--version
Display version and exit

-v, --verbose
Output extra verbose information

-q
Quiet mode (default)

-d
Debug mode

-s
Output default search directory, then exit

-aadlv[ARG]
AADL version, ARG = 1 for AADL 1.0, 2 for AADL 2.x

-f
Parse predefined non-standard property sets

-disable-annexes=ARG
Deactivate annex ARG

-r ARG
Use ARG as root system

-o ARG
Specify output file/directory

-y
Automatically load AADL files

-I ARG
Add ARG to the directory search list

-p
Parse and instantiate the model

-i
Instantiate the model

-x
Parse AADL file as an AADL scenario file

-g ARG
Generate code using Ocarina backend 'ARG'

--list-backends
List available backends

--spark2014
Generate SPARK2014 annotations

-b
Compile generated code

-z
Clean code generated

-k ARG
Set POK flavor (arinc653/deos/pok/vxworks)

-t
Run Ocarina in terminal interactive mode

-real_theorem ARG
Name of the main REAL theorem to evaluate

-real_lib ARG
Add external library of REAL theorems

-real_continue_eval
Continue evaluation of REAL theorems after first failure (REAL backend)

-boundt_process ARG
Generate .tpo file for process ARG (Bound-T backend)

-ec
Compute coverage metrics

-er
Execute system

-asn1
Generate ASN1 deployment file (PolyORB-HI-C only)

-perf
Enable profiling with gprof (PolyORB-HI-C only)

Note: A man page is also installed in Ocarina installation path, in `$OCARINA_PATH/share/man/man1/`.

4.2 ocarina-config

ocarina-config returns path and library information on Ocarina installation. This script can be used to compile user program that uses Ocarina's API.

```
Usage: ocarina-config [OPTIONS]
Options:
        No option:
            Output all the flags (compiler and linker) required
            to compile your program.
        [--prefix[=DIR]]
            Output the directory in which Ocarina architecture-independent
           files are installed, or set this directory to DIR.
        [--exec-prefix[=DIR]]
            Output the directory in which Ocarina architecture-dependent
           files are installed, or set this directory to DIR.
        [--version|-v]
            Output the version of Ocarina.
        [--config]
            Output Ocarina's configuration parameters.
        [--runtime[=<Runtime_Name>]]
            Checks the validity and the presence of the given runtime and
            then, outputs its path. Only one runtime can be requested at
            a time. If no runtime name is given, outputs the root directory
            of all runtimes.
        [--libs]
            Output the linker flags to use for Ocarina.
        [--projects]
            Output the path to GNAT Project files for Ocarina
        [--properties]
            Output the location of the standard property file.
        [--resources]
            Output the location of resource files
            (typically the standard properties)
        [--cflags]
            Output the compiler flags to use for Ocarina.
        [--help]
            Output this message
```

CHAPTER 5

Scenario files

AADL scenario files are a very simple way to set build options when using Ocarina. AADL scenario may consist of more than one AADL file.

Scenario files rely on the system component category to configure all elements of the system to be processed. The following scenario file illustrates this feature. It extends an existing scenario file (see below) with project-specific configuration data:

```
package Scenario
public
  with Ocarina_Config;
  with Ocarina_Library;

  system producer_consumer extends Ocarina_Library::Default_PolyORB_HI_C_Config
  properties
    Ocarina_Config::Referencial_Files     =>
    ("pr_a", "pr_a.ref",
    "pr_b", "pr_b.ref");
    Ocarina_Config::AADL_Files            +=>
    ("producer_consumer.aadl", "software.aadl");
    Ocarina_Config::Root_System_Name => "PC_Simple.impl";
  end producer_consumer;

  system implementation producer_consumer.Impl
  end  producer_consumer.Impl;

end scenario;
```

Scenario files rely on specific properties:

- property `AADL_Files` lists all files that are part of the system;
- property `Root_System_Name` is the name of the Root System;
- property `Generator` is the name of the generator (or back-end) to use

Note: The definition of scenario-specific properties may be found in section *Ocarina_Config*.

ocarina -x <scenario_file.aadl> will cause the scenario file to be processed. In addition, the flag `-b` will compile generated source files.

5.1 ocarina_library.aadl

```
package Ocarina_Library

--  This package provides a default scenario files that can be
--  inherited by others.

public
  with Ocarina_Config;

  system Default_PolyORB_HI_C_Config
  properties
    Ocarina_Config::AADL_Version         => AADLv2;
    --  Default AADL version

    Ocarina_Config::Generator            => PolyORB_HI_C;
    --  Use the PolyORB-HI/C backend

    Ocarina_Config::Needed_Property_Sets =>
    (Ocarina_Config::Data_Model,
     Ocarina_Config::ARINC653_Properties,
     Ocarina_Config::Deployment,
     Ocarina_Config::Cheddar_Properties);
    --  Additional property sets

    Ocarina_Config::Timeout_Property     => 4000ms;

    Ocarina_Config::AADL_Files =>
    (Ocarina_Config::Ocarina_Driver_Library);

  end Default_PolyORB_HI_C_Config;

end Ocarina_Library;
```

CHAPTER 6

PolyORB-HI/C

6.1 About

PolyORB-HI/C is a middleware for High-Integrity Systems, it inherits most concepts of the schizophrenic middleware *PolyORB* while being based on a complete new source code base, compatible with the Ravenscar profile and the restrictions for High-Integrity systems.

PolyORB-HI/C acts as an execution runtime for the AADL language. In this context, Ocarina acts as a compiler, turning an AADL model into C code that uses low-level constructs provided by PolyORB-HI/C.

The generated code is in charge of allocating all required ressources (threads, buffers, message queue), configure communication stacks, marshallers and concurrency structures.

6.2 Supported Platforms

PolyORB-HI-C has been compiled and sucessfully tested on

Native platforms

- Linux
- Mac OS X
- FreeBSD
- Windows

Embedded platforms

- AIR Hypervisor
- FreeRTOS (alpha stage)
- RTEMS
- Xenomai

- XtratuM

Note: when using RTEMS operating system, you have to define the **`RTEMS_MAKEFILE_PATH`** environment variable. See RTEMS documentation for more details.

6.3 Tree structure

PolyORB-HI-C source directory has the following tree structure:

- `doc/`: documentation,
- `examples/`: set of examples to test PolyORB-HI-C
- `share/`: common files (aadl files used by Ocarina, makefiles, ...)
- `src/`: core of PolyORB-HI
- `src/drivers`: device drivers supported by PolyORB-HI-C
- `tools/`: some script to handle the packaging and a verification tool to check if the binaries are compliant with the POSIX restrictions
- `COPYING3` and `COPYING.RUNTIME`: licence information
- `README`: short description of the distribution

When installed with Ocarina, in `$OCARINA_PATH` directory:

- documentation is in `$OCARINA_PATH/share/doc/ocarina`
- examples are in `$OCARINA_PATH/examples/ocarina/polyorb-hi-c/`
- runtime files are in `$OCARINA_PATH/include/ocarina/runtime/polyorb-hi-c/`

6.4 Generating code from an AADL model

To build your own system, you have to select the PolyORB-HI/C backend, either using a scenario file or the command line.

- To use a scenario file, refer to *Scenario files*
- To use command line, you have to select the `polyorb_hi_c` backend, e.g. by using the command **`ocarina -g polyorb_hi_c <list-of-aadl-files>`**. Refer to *Ocarina command-line* for more details .

6.5 Code generation towards PolyORB-HI/C

6.5.1 The `ping` example

In the following, and for each component of the distributed application, we give the AADL entities that are used to model this component, and then the transformation rules used by the code generator to generate C code from these entities.

The mapping rules will be illustrated using the following simple example of a distributed application:

@image{fig/ping, 12cm}

The figure above shows the architecture of the `ping` example: a client, which is a process containing one single *periodic* thread, sends a message to the server which is a process containing one *sporadic* thread that handles incoming ping messages from the client. Each node of the `ping` application runs on a different machine.

In this chapter, we first present the AADL modeling patterns used to define a distributed application. Then, we give the rules applied to map AADL entities onto instances of PolyORB-HI/C elements.

In the following, we detail only the rules that are directly related to the distributed application as a whole system. The rules that are specific to the components of the distributed application are explained in the sections that deals with these respective components.

6.5.2 Mapping AADL `system`

A full system is captured using a root system. The system implementation shown on the following example models such system.

```
system PING
end PING;

system implementation PING.Impl
subcomponents
  -- Nodes
  Node_A : process A.Impl;
  Node_B : process B.Impl {ARAO::port_number => 12002;};

  -- Processors
  -- ...
connections
  -- ...
properties
  -- ...
end PING.Impl;
```

For each node (process) of the system, we instantiate a subcomponent in the system implementation, and bind it to a `processor` component. This processor will configure the build target.

We use the `properties` section of the AADL `system` (see @ref{Hosts‘ for more details) to map the different nodes on the different platforms of the distributed application. The `connections` section of the system implementation models the connections between the different nodes of the application.

An AADL `system` is mapped into a hierarchy of directories:

- the root directory of the distributed application has the same name as the root system implementation, in lower case, all dot being converted into underscores. This directory is the root of the directory hierarchy of the generated C code.
- for each node of the distributed application, a child directory having the same name as the corresponding process subcomponent (in lower case) is created inside the root directory. This child directory will contain all the code generated for the particular node it was created for (see @ref{Distributed application nodes} for more details).

6.5.3 Mapping AADL `process`

We use the `process` component category to model an application node (e.g. an element that would ultimately become a Unix process, an embedded application or an ARINC653 partition). The process implementation shown in the listing below shows such system.

```
process A
features
  Out_Port : out event data port Simple_Type;
end A;

process implementation A.Impl
subcomponents
  Pinger : thread P.Impl;
connections
  C1 : event data port Pinger.Data_Source -> Out_Port;
end A.Impl;
```

For each thread that belongs to a node of the distributed application, we instantiate a subcomponent in the process implementation. For each connection between a node and another, a `port` feature has to be added to both nodes with the direction `out` for the source and `in` for the destination (see @ref{Connections} for more details on connections mapping).

Elements associated to this process (threads, subprograms, data types, etc) are generated in a child directory of the root system directory. This directory has the same name as the process *subcomponent* instance relative to the handled node in the system implementation that model the distributed application, in lower case.

For example, all the entities relative to the process `A.Impl` of the `Ping` example are generated in the directory `ping_impl/node_a`.

The following paragraphs list the C compilation units that are created for each node of the distributed application.

Marshallers functions

The marshallers functions are used to put all request and types values in a message in order to send them through a network connections. All marshalling functions are declared in the file `marshallers.c`.

However, PolyORB-HI-C can also use third-party marshallers. It can rely on the marshallers generated for ASN1 encoding. Details about ASN1 marshallers are provided in the next section.

@subsubsection Using ASN1 marshallers

With the ASN1 tools from Semantix (see. @url{http://www.semantix.gr/assert/}), you can convert ASN1 declarations into AADL models. Then, these models can be used with AADL components and PolyORB-HI-C relies on Semantix tools to automatically generates C code that implements the ASN1 types.

For that purpose, you need to install the program `asn2aadlPlus` and `asn1cc`. These programs are freely available on @url{http://www.semantix.gr/assert/}. Then, when you use ASN1 types with your AADL model (with the AADL files generated with `asn2aadlPlus`), PolyORB-HI-C uses the generated code from ASN1 descriptions and integrate it to marshall data.

Node activity

We denote *activity* the set of the actions performed by one particular node which are not triggered by other nodes. All the periodic threads of a node are part of the node activity.

The code related to the node activity is generated in an C file with the name `activity.c`. An example is shown below :

```
#include <po_hi_types.h>
#include <po_hi_gqueue.h>
#include <request.h>
#include <deployment.h>
```

(continues on next page)

(continued from previous page)

```
#include <types.h>
#include <subprograms.h>
#include <po_hi_task.h>
#include <po_hi_main.h>
#include <marshallers.h>
extern __po_hi_entity_t __po_hi_port_global_to_entity[__PO_HI_NB_PORTS];
extern __po_hi_port_t __po_hi_port_global_to_local[__PO_HI_NB_PORTS];
__po_hi_int8_t __po_hi_data_source_local_destinations[1] = {ping_me_global_data_sink};
__po_hi_uint8_t __po_hi_pinger_woffsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_offsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_used_size[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_empties[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_first[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_recent[__po_hi_pinger_nb_ports * sizeof(__po_hi_
↪request_t)];
__po_hi_uint8_t __po_hi_pinger_queue[0 * sizeof(__po_hi_request_t)];
__po_hi_uint16_t __po_hi_pinger_total_fifo_size = 0;
__po_hi_port_t __po_hi_pinger_history[0];
__po_hi_uint8_t __po_hi_pinger_n_dest[__po_hi_pinger_nb_ports] = {1};
__po_hi_int8_t __po_hi_pinger_fifo_size[__po_hi_pinger_nb_ports] = {__PO_HI_GQUEUE_
↪FIFO_OUT};
__po_hi_uint8_t* __po_hi_pinger_destinations[__po_hi_pinger_nb_ports] = {__po_hi_data_
↪source_local_destinations};
/*  Periodic task : Pinger*/

/****************/
/* pinger_job   */
/****************/

void* pinger_job ()
{
   simple_type data_source_request_var;
   __po_hi_request_t data_source_request;

   __po_hi_gqueue_init(node_a_pinger_k,__po_hi_pinger_nb_ports,__po_hi_pinger_queue,__
↪po_hi_pinger_fifo_size,__po_hi_pinger_first,__po_hi_pinger_offsets,__po_hi_pinger_
↪woffsets,__po_hi_pinger_n_dest,__po_hi_pinger_destinations,__po_hi_pinger_used_size,
↪__po_hi_pinger_history,__po_hi_pinger_recent,__po_hi_pinger_empties,__po_hi_pinger_
↪total_fifo_size);
   __po_hi_wait_initialization();
   while (1)
   {
      /*  Call implementation*/
      do_ping_spg(&(data_source_request_var));
      /*  Set the OUT port values*/
      data_source_request.vars.pinger_global_data_source.pinger_global_data_source =␣
↪data_source_request_var;
      data_source_request.port = data_source_request_var;
      __po_hi_gqueue_store_out(node_a_pinger_k,pinger_local_data_source,&(data_source_
↪request));
      /*  Send the OUT ports*/
      __po_hi_gqueue_send_output(node_a_pinger_k,pinger_global_data_source);
      __po_hi_wait_for_next_period(node_a_pinger_k);
   }
}
```

(continues on next page)

(continued from previous page)

```
/*************************/
/* __po_hi_main_deliver   */
/*************************/

void __po_hi_main_deliver
      (__po_hi_msg_t* message)
{
   __po_hi_request_t request;
   __po_hi_entity_t entity;

   __po_hi_unmarshall_request(&(request),message);
   entity = __po_hi_port_global_to_entity[request.port];
   switch (entity)
   {
      default:
      {
         break;
      }
   }
}
```

All the naming rules explained in @ref{Whole distributed application} are also applied to map the package name. This file contains all the routines mapped from the periodic threads that belong to the handled node (see @ref{Threads} for more details on thread mapping). This package contains also the instances of shared objects used in this node (see @ref{Data} for more details). If the node does not contain any *periodic* thread nor shared objects, there is no `activity.c` file generated for this node. Thus, the node `B` in the `Ping` example does not have a `activity.c` package.

Data types

All the data types mapped from AADL data components and used by a particular node of a distributed application are gathered in a separate C file called `types.h`.

For more detail on the mapping of data components, see @ref{Data}.

Subprograms

The mapping of all AADL subprogram components used by a particular node is generated in a separate file called `subprograms.c`. The content of the file is shown in the following example:

For more detail on the mapping of subprogram components, see @ref{Subprograms}.

Deployment information

The deployment information is the information each node has on the other nodes in the distributed applications. This information is used, to send a request to another node or to receive a request from another node. The deployment information is generated for each node in two C files : `deployment.h` and `deployment.c`.

The file `deployment.h` contains the following types

- a first type called `__po_hi_node_t`. For each node in the application we create an enum whose name is mapped from the node *instance* declared in the system implementation to which we concatenate the string *_k*. All the naming rules listed in @ref{Whole distributed application} have to be respected.

- a second type called __po_hi_entity_t. For each thread in the the application, we declare an enum.
- a third type called __po_hi_task_id. For each thread that run on the current node.
- a fourth type called __po_hi_entity_server_t. For each node that may communicate with the current node, we add a value in this enum. It will be used by the transport layer. Please note that at least one server is declared : the value invalid_server.
- a fifth type called __po_hi_port_t that contains all global port identifier.

More, this file contains the following maccros :

- __PO_HI_NB_ENTITIES is the number of entities in the whole distributed system.
- __PO_HI_NB_TASKS is the number of the tasks that will be started on the current node
- __PO_HI_NB_NODES is the number of nodes in the distributed system.
- __PO_HI_PROTECTED is the number of protected objects use on the current node.
- __PO_HI_NB_PORTS that represent the total number of ports in the whole distributed system.
- __PO_HI_NB_DEVICES that represent the total number of devices in the whole distributed system.

The file deployment.c contains the following variables :

- mynode variable which has the value of the handled node.
- __po_hi_entity_table variable is used to know on which node an entity runs.
- __po_hi_port_global_to_local variable is used to convert a global port identifier to a local port identifier
- __po_hi_port_global_to_entity variable is used to know on which entity a given port is. This table is used convert a global port identifier to an entity identifier.
- __po_hi_uint8_t __po_hi_deployment_endiannesses variable details which the endianess of each node. It is an array which size is __PO_HI_NB_NODES.
- __po_hi_port_to_device is an array which size is __PO_HI_NB_PORTS. For each port, it indicates the value of the device identifier that handles it.
- __po_hi_port_global_model_names is an array which size is __PO_HI_NB_PORTS. For each port, it contains the name of the port.
- __po_hi_port_global_names is an array which size is __PO_HI_NB_PORTS. For each port, it contains the name generated by the code generator.
- __po_hi_devices_naming is an array which size is __PO_HI_NB_DEVICES. For each deivce, it contains all relevant information for their configuration. The configuration string is deduced from the Configuration property associated with the device.

The following example shows the Deployment package relative to the node A of the Ping example:

```
#ifndef __DEPLOYMENT_H_
#define __DEPLOYMENT_H_
#include <po_hi_protected.h>
typedef enum
{
   pinger_local_data_source = 0
} __po_hi_pinger_t;

#define __po_hi_pinger_nb_ports 1

typedef enum
```

(continues on next page)

(continued from previous page)

```
{
   ping_me_local_data_sink = 0
} __po_hi_ping_me_t;

#define __po_hi_ping_me_nb_ports 1

/*  For each node in the distributed application add an enumerator*/

typedef enum
{
   node_a_k = 0,
   node_b_k = 1
} __po_hi_node_t;

/*  For each thread in the distributed application nodes, add an enumerator*/

typedef enum
{
   node_a_pinger_k_entity = 0,
   node_b_ping_me_k_entity = 1
} __po_hi_entity_t;

typedef enum
{
   node_a_pinger_k = 0
} __po_hi_task_id;

#define __PO_HI_NB_TASKS 1

/*  For each thread in the distributed application nodes THAT MAY COMMUNICATE*/
/*   with the current node, add an enumerator*/

typedef enum
{
   invalid_server = -1
} __po_hi_entity_server_t;

#define __PO_HI_NB_SERVERS 0

#define __PO_HI_NB_PROTECTED 0

#define __PO_HI_NB_NODES 2

#define __PO_HI_NB_ENTITIES 2

#define __PO_HI_NB_PORTS 2

typedef enum
{
   pinger_global_data_source = 0,
   ping_me_global_data_sink = 1
} __po_hi_port_t;

#endif
```

OS Configuration

A host is the set formed by a processor and an operating system (or real-time kernel).

To model both the processor and the OS, we use the `processor` AADL component. The characteristics of the processor are defined using AADL properties. For example, if our distributed application uses an IP based network to make its node communicate, then each host must have an IP address. Each host must also precise its platform (native, LEON...). The listing following example shows how to express this using a custom property set.

```
processor the_processor
properties
  ARAO::location           => "127.0.0.1";
  ARAO::Execution_Platform => Native;
end the_processor;
```

To map an application node (processor) to a particular host, we use the `Actual_Processor_Binding` property. The following example shows how the node `Node_A` is mapped to the processor `Proc_A` in the `Ping` example.

```
system PING
end PING;

system implementation PING.Impl
subcomponents
  -- Nodes
  Node_A : process A.Impl;
  Node_B : process B.Impl {ARAO::port_number => 12002;};

  -- Processors
  CPU_A  : processor the_processor;
  CPU_B  : processor the_processor;
connections
  -- ...
properties
  -- Processor bindings
  actual_processor_binding => reference CPU_A applies to Node_A;
  actual_processor_binding => reference CPU_B applies to Node_B;
end PING.Impl;
```

The C generated code concerning the code generation to model host mapping is located in the `naming.c` file. More precisely, the `node_addr` and `node_port` contains, for each node, the information related to its host. These information are dependant on the transport mechanism used in the distributed application.

6.5.4 Mapping AADL `threads`

The threads are the active part of the distributed application. A node must contain at least one thread and may contain more than one thread. In this section, we give the AADL entities used to model threads. Then, we give the mapping rule to generate C code corresponding to the periodic and aperiodic threads.

The rules are listed relatively to the packages generated for the nodes and for the distributed application (see @ref{Distributed application nodes} and @ref{Whole distributed application}). Only rules that are related directly to a thread as a whole subsystem are listed here.

AADL entities

AADL `thread` components are used to model execution flows.

The `features` subclause of the thread component declaration describes the thread interface. The ports that may be connected to the ports of other threads, enclosing process, etc.

The `properties` subclause of the thread implementation lists the properties of the thread such as its priority, its nature (periodic, sporadic) and many other properties ares expressed using AADL properties.

The `calls` subclause of the thread implementation contains the sequences of subprograms the thread may call during its job (see @ref{Subprograms} for more details on the subprogram mapping). If the thread job consist of calling more than one subprogram, it is **mandatory** to encapsulate these calls inside a single subprogram which will consist the thread job.

The `connections` section of a thread implementation connects the parameters of the subprograms called by the thread to the ports of the threads or to the parameters of other called subprograms in the same thread.

```
thread P
features
  Data_Source : event out data port Simple_Type;
end P;

thread implementation P.Impl
calls {
  --  ...
};
connections
  --  ...
properties
  Dispatch_Protocol => Periodic;
  Period            => 1000 Ms;
end P.Impl;
```

The listing above shows the thread `P` which belongs to the process `A.impl` in the `Ping` example. We can see that `P` is a periodic thread with a period of 1000ms, that this thread has a unique `out event data port` and that at each period, the thread performs a call to the `Do_Ping_Spg` subprogram whose `out parameter` is connected to the thread port.

Mapping rules for periodic threads

Periodic threads are cyclic threads that are triggered by and only by a periodic time event. between two time events the periodic threads do a non blocking job and then they sleep waiting for the next time event.

The majority of the code generated for the periodic threads is put in the `activity.c` file generated for the application node containing the handled thread. Each periodic thread is created in the main function (`main.c` file) with the `__po_hi_create_periodic_task` function-call.

The generated code in the `activity.c` file is a parameterless function that represents the thread job. The defining identifier of the function is mapped from the thread instance name in the process that models the node, to which we append the string `_job`. All the naming rules listed in @ref{Whole distributed application} have to be respected. The body of this subprogram calls the subprograms mapped from the subprogram calls the thread performs. Then, it sends the request to the remote threads it may be connected to. Finally, at the end of the function, we make a call to the `__po_hi_wait_next_period()` with the task identifier as parameter. This call ensure that we wait the next period before we start the function again.

The generated code in `main.c` file is a function call that creates a periodic task. The task is created with the function `__po_hi_create_periodic_task`. This creates a periodic task with the wanted properties at the elaboration time of the node. The package instantiation name is mapped from the thread instance name in the process that model the node, to which we append the string `_k`. All the naming rules listed in @ref{Whole distributed application} have to be respected. The function-call takes the following parameters:

- the enumerator corresponding to the thread
- the task period,
- the task priority. If the user did not specify a priority, then __PO_HI_DEFAULT_PRIORITY is used,
- the task job which corresponds to the subprogram <Thread_Name>_job.

The following example shows the generated code for the periodic thread Pinger from the node Node_A of the Ping example:

```
#include <po_hi_types.h>
#include <po_hi_gqueue.h>
#include <request.h>
#include <deployment.h>
#include <types.h>
#include <subprograms.h>
#include <po_hi_task.h>
#include <po_hi_main.h>
#include <marshallers.h>
extern __po_hi_entity_t __po_hi_port_global_to_entity[__PO_HI_NB_PORTS];
extern __po_hi_port_t __po_hi_port_global_to_local[__PO_HI_NB_PORTS];
__po_hi_int8_t __po_hi_data_source_local_destinations[1] = {ping_me_global_data_sink};
__po_hi_uint8_t __po_hi_pinger_woffsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_offsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_used_size[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_empties[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_first[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_recent[__po_hi_pinger_nb_ports * sizeof(__po_hi_
↪request_t)];
__po_hi_uint8_t __po_hi_pinger_queue[0 * sizeof(__po_hi_request_t)];
__po_hi_uint16_t __po_hi_pinger_total_fifo_size = 0;
__po_hi_port_t __po_hi_pinger_history[0];
__po_hi_uint8_t __po_hi_pinger_n_dest[__po_hi_pinger_nb_ports] = {1};
__po_hi_int8_t __po_hi_pinger_fifo_size[__po_hi_pinger_nb_ports] = {__PO_HI_GQUEUE_
↪FIFO_OUT};
__po_hi_uint8_t* __po_hi_pinger_destinations[__po_hi_pinger_nb_ports] = {__po_hi_data_
↪source_local_destinations};
/*  Periodic task : Pinger*/

/****************/
/* pinger_job   */
/****************/

void* pinger_job ()
{
   simple_type data_source_request_var;
   __po_hi_request_t data_source_request;

   __po_hi_gqueue_init(node_a_pinger_k,__po_hi_pinger_nb_ports,__po_hi_pinger_queue,__
↪po_hi_pinger_fifo_size,__po_hi_pinger_first,__po_hi_pinger_offsets,__po_hi_pinger_
↪woffsets,__po_hi_pinger_n_dest,__po_hi_pinger_destinations,__po_hi_pinger_used_size,
↪__po_hi_pinger_history,__po_hi_pinger_recent,__po_hi_pinger_empties,__po_hi_pinger_
↪total_fifo_size);
   __po_hi_wait_initialization();
   while (1)
   {
      /*  Call implementation*/
      do_ping_spg(&(data_source_request_var));
```

(continues on next page)

(continued from previous page)

```
      /*  Set the OUT port values*/
      data_source_request.vars.pinger_global_data_source.pinger_global_data_source =
↪data_source_request_var;
      data_source_request.port = data_source_request_var;
      __po_hi_gqueue_store_out(node_a_pinger_k,pinger_local_data_source,&(data_source_
↪request));
      /*  Send the OUT ports*/
      __po_hi_gqueue_send_output(node_a_pinger_k,pinger_global_data_source);
      __po_hi_wait_for_next_period(node_a_pinger_k);
   }
}


/*************************/
/* __po_hi_main_deliver   */
/*************************/

void __po_hi_main_deliver
      (__po_hi_msg_t* message)
{
   __po_hi_request_t request;
   __po_hi_entity_t entity;

   __po_hi_unmarshall_request(&(request),message);
   entity = __po_hi_port_global_to_entity[request.port];
   switch (entity)
   {
      default:
      {
         break;
      }
   }
}
```

Mapping rules for sporadic threads

Sporadic threads are cyclic threads that are triggered by the arrival of a sporadic event. The minimum inter-arrival time between two sporadic events is called the period of the sporadic thread.

The majority of the code generated for the sporadic threads is put in the `activity.c` file generated for the application node containing the handled thread. Each periodic thread is created in the main function (`main.c` file) with the `__po_hi_create_sporadic_task` function-call.

The generated code in the `activity.c` file is a parameterless function that represents the thread job. The defining identifier of the function is mapped from the thread instance name in the process that models the node, to which we append the string `_job`. All the naming rules listed in @ref{Whole distributed application} have to be respected. In the body of the function, the thread will wait for an event (most of the time : a message from another entity).

The generated code in `main.c` file is a function-call that creates the sporadic task. The task is created with the function `__po_hi_create_sporadic_task`. This creates a sporadic task with the wanted properties at the elaboration time of the node. The package instantiation name is mapped from the thread instance name in the process that model the node, to which we append the string `_k`. All the naming rules listed in @ref{Whole distributed application} have to be respected. The function-call takes the following parameters:

- the enumerator corresponding to the thread
- the task priority. If the user did not specify a priority, then __PO_HI_DEFAULT_PRIORITY is used,
- the task job which corresponds to the subprogram <Thread_Name>_job.

The following example shows the generated code for the sporadic thread Ping_Me from the node Node_B of the Ping example.

```
#include <po_hi_gqueue.h>
#include <po_hi_types.h>
#include <request.h>
#include <deployment.h>
#include <po_hi_task.h>
#include <subprograms.h>
#include <po_hi_main.h>
#include <marshallers.h>
extern __po_hi_entity_t __po_hi_port_global_to_entity[__PO_HI_NB_PORTS];
extern __po_hi_port_t __po_hi_port_global_to_local[__PO_HI_NB_PORTS];
__po_hi_uint8_t __po_hi_ping_me_woffsets[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_offsets[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_used_size[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_empties[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_first[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_recent[__po_hi_ping_me_nb_ports * sizeof(__po_hi_
↪request_t)];
__po_hi_uint8_t __po_hi_ping_me_queue[16 * sizeof(__po_hi_request_t)];
__po_hi_uint16_t __po_hi_ping_me_total_fifo_size = 16;
__po_hi_port_t __po_hi_ping_me_history[16];
__po_hi_uint8_t __po_hi_ping_me_n_dest[__po_hi_ping_me_nb_ports] = {0};
__po_hi_int8_t __po_hi_ping_me_fifo_size[__po_hi_ping_me_nb_ports] = {16};
__po_hi_uint8_t* __po_hi_ping_me_destinations[__po_hi_ping_me_nb_ports] = {NULL};

/*********************/
/* ping_me_deliver   */
/*********************/

void ping_me_deliver
      (__po_hi_request_t* request)
{

   switch (request->port)
   {
      case ping_me_global_data_sink:
      {
         __po_hi_gqueue_store_in(node_b_ping_me_k,ping_me_local_data_sink,request);

         break;
      }
      default:
      {
         break;
      }
   }
}

/*  Sporadic task : Ping_Me*/
/*  Get the IN ports values*/
```

(continues on next page)

(continued from previous page)

```
/*****************/
/* ping_me_job   */
/*****************/

void* ping_me_job ()
{
   __po_hi_port_t port;
   __po_hi_request_t data_sink_request;

   __po_hi_gqueue_init(node_b_ping_me_k,__po_hi_ping_me_nb_ports,__po_hi_ping_me_
↪queue,__po_hi_ping_me_fifo_size,__po_hi_ping_me_first,__po_hi_ping_me_offsets,__po_
↪hi_ping_me_woffsets,__po_hi_ping_me_n_dest,__po_hi_ping_me_destinations,__po_hi_
↪ping_me_used_size,__po_hi_ping_me_history,__po_hi_ping_me_recent,__po_hi_ping_me_
↪empties,__po_hi_ping_me_total_fifo_size);
   __po_hi_wait_initialization();
   while (1)
   {
      __po_hi_gqueue_wait_for_incoming_event(node_b_ping_me_k,&(port));
      __po_hi_compute_next_period(node_b_ping_me_k);
      if (__po_hi_gqueue_get_count(node_b_ping_me_k,ping_me_local_data_sink))
      {
               __po_hi_gqueue_get_value(node_b_ping_me_k,ping_me_local_data_sink,&
↪(data_sink_request));
         __po_hi_gqueue_next_value(node_b_ping_me_k,ping_me_local_data_sink);

      }
      /*  Call implementation*/
      ping_spg(data_sink_request.vars.ping_me_global_data_sink.ping_me_global_data_
↪sink);
      __po_hi_wait_for_next_period(node_b_ping_me_k);
   }
}


/*************************/
/* __po_hi_main_deliver   */
/*************************/

void __po_hi_main_deliver
      (__po_hi_msg_t* message)
{
   __po_hi_request_t request;
   __po_hi_entity_t entity;

   __po_hi_unmarshall_request(&(request),message);
   entity = __po_hi_port_global_to_entity[request.port];
   switch (entity)
   {
      case node_b_ping_me_k_entity:
      {
         ping_me_deliver(&(request));

         break;
      }
      default:
      {
         break;
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

6.5.5 Deployment information

As said in @ref{Distributed application nodes}, the files `deployment.h` and `deployment.c` are generated for each node in the distributed application. For each thread port in the whole distributed application, we declare an enumerator in this type. The defining identifier of the enumerator is mapped from the process subcomponent name and the thread subcomponent name as follows: `<Node_Name>_<Thread_Name>_K`.

For each that that may communicate, we generate the following elements

- A variable called `__po_hi_<thread_name>_local_to_global` (in `deployment.c`) that is used to convert a local port identifier of the thread to a global one.
- A type `__po_hi_<thread_name>_t` that will contain on local port identifier.
- A macro `__po_hi_<thread_name>_nb_ports` that will contain the number of ports for the thread.

For these elements, all the naming rules listed in @ref{Whole distributed application} must be respected.

```
#ifndef __DEPLOYMENT_H_
#define __DEPLOYMENT_H_
#include <po_hi_protected.h>
typedef enum
{
   pinger_local_data_source = 0
} __po_hi_pinger_t;

#define __po_hi_pinger_nb_ports 1

typedef enum
{
   ping_me_local_data_sink = 0
} __po_hi_ping_me_t;

#define __po_hi_ping_me_nb_ports 1

/*  For each node in the distributed application add an enumerator*/

typedef enum
{
   node_a_k = 0,
   node_b_k = 1
} __po_hi_node_t;

/*  For each thread in the distributed application nodes, add an enumerator*/

typedef enum
{
   node_a_pinger_k_entity = 0,
   node_b_ping_me_k_entity = 1
} __po_hi_entity_t;
```

(continues on next page)

(continued from previous page)

```
typedef enum
{
   node_a_pinger_k = 0
} __po_hi_task_id;

#define __PO_HI_NB_TASKS 1

/*  For each thread in the distributed application nodes THAT MAY COMMUNICATE*/
/*   with the current node, add an enumerator*/

typedef enum
{
   invalid_server = -1
} __po_hi_entity_server_t;

#define __PO_HI_NB_SERVERS 0

#define __PO_HI_NB_PROTECTED 0

#define __PO_HI_NB_NODES 2

#define __PO_HI_NB_ENTITIES 2

#define __PO_HI_NB_PORTS 2

typedef enum
{
   pinger_global_data_source = 0,
   ping_me_global_data_sink = 1
} __po_hi_port_t;

#endif
```

```
#include <deployment.h>
__po_hi_entity_server_t server_entity_table[__PO_HI_NB_ENTITIES] = {invalid_server,
↪invalid_server};
__po_hi_node_t entity_table[__PO_HI_NB_ENTITIES] = {node_a_k,node_b_k};
__po_hi_node_t mynode = node_a_k;
```

The listing above shows the generated `__po_hi_entity_server_t` and `entity_table` for the nodes `B` from the `Ping` example.

6.5.6 Mapping of AADL `ports`

Threads can contain one or several ports. To handle them, we declared several arrays in the `activity.c`

- **`__po_hi_<port_name>_destinations`** [array for each port of] the thread which contains all destinations of the port.
- **`__po_hi_<thread_name>_woffsets`** [array (size = number of] ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_offsets` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.

- `__po_hi_<thread_name>_used_size` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_empties` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_first` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_recent` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_queue` : array (size = size of the global queue for the thread) used by pohic to handle the global queue.
- `__po_hi_<thread_name>_total_fifo_size` : variable that contains the size of the global queue. It is the sum of all port size for the thread.
- `__po_hi_<thread_name>_history` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_n_dest` : array (size = number of ports in the thread) used by pohic for the global queue of the thread. It contains the number of destinations for each port of the thread.
- `__po_hi_<thread_name>_fifo_size` : array (size = number of ports in the thread) used by pohic for the global queue of the thread.
- `__po_hi_<thread_name>_destinations` : array (size = number of ports in the thread) that contains all destinations for each port.

6.5.7 Mapping of AADL `Connections`

The connections are entities that support communication between the application nodes. In this section, we present the AADL entities used to model connection between nodes.

A connection between two nodes of the system is modeled by:

- The `ports` features that exist on each one of the nodes. Ports can be declared inside processes or threads. The direction of the port (`in`, `out` or `in out`) indicates the direction of the information flow.
- The `connections` section in the system implementation relative to the distributed application and in the process and thread implementations.

```
system PING
end PING;

system implementation PING.Impl
subcomponents
  -- Nodes
  Node_A : process A.Impl;
  Node_B : process B.Impl {ARAO::port_number => 12002;};

  -- Processors
  CPU_A  : processor the_processor;
  CPU_B  : processor the_processor;
connections
  -- Port connections
  event data port Node_A.Out_Port -> Node_B.In_Port;
properties
  -- Processor bindings
```

(continues on next page)

(continued from previous page)

```
  actual_processor_binding => reference CPU_A applies to Node_A;
  actual_processor_binding => reference CPU_B applies to Node_B;
end PING.Impl;
```

The listing above shows the connection between the node `A` and `B` in the system implementation.

The nature of the `port` (*event port*, *data port* or *event data port*) depends on the nature of the connection between the two nodes:

- if the message sent from one node to another node is only a triggering event and contains no data, we create an *event* port.
- if the message sent from one node to another node is a data message but it does not trigger the receiver thread, we create a *data* port.
- if the message sent from one node to another node is a data message that triggers the receiver thread, we create an @i{event data} port.

In a distributed system, when we send any data to a node, we need to put them in a stream. We call that the marshall operation. On the other hand, find data in a stream is called the unmarshall operation. In each distributed application, we generate marshallers for each types and request. These functions will marshall/unmarshall data in/from a message.

All marshallers functions are generated in a file called `marshallers.c`. The marshall (or unmarshall) functions for request are prefixed by the string `__po_hi_marshall_request_` (or `__po_hi_unmarshall_request_`). Marshall (or unmarshall) functions for types are prefixed by the string `__po_hi_marshall_type_` (or `__po_hi_unmarshall_type_`). Each function has the name of the type or the request it marshalls.

Finally, a function `__po_hi_marshall_request` and `__po_hi_unmarshall_request` is generated to handle all requests. Then, is called the appropriate function to call to marshall or unmarshall the data.

```
#include <types.h>
#include <po_hi_types.h>
#include <po_hi_marshallers.h>

/**************************************/
/* __po_hi_marshall_type_simple_type   */
/**************************************/

void __po_hi_marshall_type_simple_type
      (simple_type value,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_marshall_int(value,message,offset);
}


/**************************************/
/* __po_hi_unmarshall_type_simple_type   */
/**************************************/

void __po_hi_unmarshall_type_simple_type
      (simple_type* value,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{
```

(continues on next page)

(continued from previous page)

```
   __po_hi_unmarshall_int(value,message,offset);
}


/*********************************************/
/* __po_hi_marshall_request_ping_me_data_sink   */
/*********************************************/

void __po_hi_marshall_request_ping_me_data_sink
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_marshall_type_simple_type(request->vars.ping_me_global_data_sink.ping_me_
↪global_data_sink,message,offset);
}


/*************************************************/
/* __po_hi_unmarshall_request_ping_me_data_sink   */
/*************************************************/

void __po_hi_unmarshall_request_ping_me_data_sink
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_unmarshall_type_simple_type(&(request->vars.ping_me_global_data_sink.ping_
↪me_global_data_sink),message,offset);
}


/*************************************************/
/* __po_hi_marshall_request_pinger_data_source   */
/*************************************************/

void __po_hi_marshall_request_pinger_data_source
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_marshall_type_simple_type(request->vars.pinger_global_data_source.pinger_
↪global_data_source,message,offset);
}


/*************************************************/
/* __po_hi_unmarshall_request_pinger_data_source   */
/*************************************************/

void __po_hi_unmarshall_request_pinger_data_source
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
```

(continues on next page)

(continued from previous page)

```
{

   __po_hi_unmarshall_type_simple_type(&(request->vars.pinger_global_data_source.
↪pinger_global_data_source),message,offset);
}


/*****************************/
/* __po_hi_marshall_request   */
/*****************************/

void __po_hi_marshall_request
      (__po_hi_request_t* request,
      __po_hi_msg_t* message)
{
   __po_hi_uint16_t offset;

   offset = 0;
   __po_hi_marshall_port(request->port,message);
   switch (request->port)
   {
      case ping_me_global_data_sink:
      {
         __po_hi_marshall_request_ping_me_data_sink(request,message,&(offset));

         break;
      }
      case pinger_global_data_source:
      {
         __po_hi_marshall_request_pinger_data_source(request,message,&(offset));

         break;
      }
      default:
      {
         break;
      }
   }
}


/*******************************/
/* __po_hi_unmarshall_request   */
/*******************************/

void __po_hi_unmarshall_request
      (__po_hi_request_t* request,
      __po_hi_msg_t* message)
{
   __po_hi_uint16_t offset;

   offset = 0;
   __po_hi_unmarshall_port(&(request->port),message);
   switch (request->port)
   {
      case ping_me_global_data_sink:
      {
```

(continues on next page)

(continued from previous page)

```
         __po_hi_unmarshall_request_ping_me_data_sink(request,message,&(offset));

         break;
      }
      case pinger_global_data_source:
      {
         __po_hi_unmarshall_request_pinger_data_source(request,message,&(offset));

         break;
      }
      default:
      {
         break;
      }
   }
}
```

6.5.8 Mapping of AADL `Subprograms`

Subprograms are used to encapsulate behavioural aspects of the application.

To model a subprogram, we use the `subprogram` AADL component category. The parameters of the subprogram are specified in the `features` subclausen of the component declaration. If the subprogram does only the job of calling other declared subprograms, then the `calls` subclause of the subprogram implementation has to contain such calls. To point to the actual implementation of the subprogram, we use the AADL properties.

The following example shows the AADL model for the `Do_Ping_Spg` from the `Ping` example. It precises that the C implementation of the subprogram is located in the function `user_ping`. The file which contains this function must be stored with the aadl model.

Subprograms are generally called by threads or by other subprograms. To express this, we use the `calls` subclause of a component implementation. Then we perform all the connections between the called subprograms parameters and the caller components ports (or parameters if the caller is a subprogram).

The following listing shows the calls and connections sections of the periodic thread `P` in the `Ping` example.

```
subprogram Do_Ping_Spg
features
  Data_Source : out parameter Simple_Type;
properties
  source_language => C;
  source_name     => "user_ping";
end Do_Ping_Spg;
```

Each subprogram instance model a hand-written function. In the `subprograms.c` file, we declare the definition of this function and we generate a new one that will call the one provided by the user.

The following listing shows the calls and connections sections of the subprogram `ping_spg` in the `Ping` example.

```
#include <types.h>
#include <subprograms.h>
void user_do_ping_spg
      (simple_type* data_source);
/*****************/
```

(continues on next page)

(continued from previous page)

```
/* do_ping_spg */
/*****************/

void do_ping_spg
      (simple_type* data_source)
{

   user_do_ping_spg(data_source);
}
```

For each subprogram call in a thread, we generate an C subprogram call to the subprogram implementing the thread and given by mean of the AADL properties.

On the client side, the function `sth_Job` begins by calling the subprogram in its call sequence. then it calls the stubs of all the subprogram it is connected to.

On the server side, and in the function of the `process_request`, the subprogram implementation corresponding to the operation (coded in the message) is called.

6.5.9 Mapping of AADL `data`

The data are the messages exchanged amongst the nodes of the application.

AADL `data` components are used to model data exchanged in the distributed application. Properties are used to precise the nature of the data. To model a data structure (which contains fields of others data types) we use data component implementation and we add a subcomponent for each field of the structure.

The simple data types that can be modeled using AADL are

- boolean,
- integer,
- fixed point types,
- characters,
- wide characters

```
--  Boolean type

data Boolean_Data
properties
  ARAO::Data_Type  => Boolean;
end Boolean_Data;

--  Integer type

data Integer_Data
properties
  ARAO::Data_Type  => Integer;
end Integer_Data;

--  Fixed point type

data Fixed_Point_Type
```

(continues on next page)

(continued from previous page)

```
properties
  ARAO::Data_Type     => Fixed;

  ARAO::Data_Digits   => 10;
  --  The total number of digits is 10

  ARAO::Data_Scale    => 4;
  --  The precision is 10**(-4)
end Fixed_Point_Type;

--  Character type

data Character_Data
properties
  ARAO::Data_Type   => Character;
end Character_Data;

--  Wide character type

data W_Character_Data
properties
  ARAO::Data_Type   => Wide_Character;
end W_Character_Data;
```

The complex data types that can be modeled using AADL are

- Bounded strings
- Bounded wide strings
- Bounded arrays of a type that can be modeled
- Structure where the fields types are types that can be modeled

```
-- Bounded string type

data String_Data
properties
  ARAO::Data_Type   => String;
  ARAO::Max_Length => <User_Defined_Length>;
end String_Data;

-- Bounded wide string type

data W_String_Data
properties
  ARAO::Data_Type   => Wide_String;
  ARAO::Max_Length => <User_Defined_Length>;
end W_String_Data;

-- Bounded array type: Only the component implementation should be
-- used in the ports or parameters!

data Data_Array
properties
  ARAO::Length => <User_Defined_Length>;
end Data_Array;
```

(continues on next page)

(continued from previous page)

```
data implementation Data_Array.i;
subcomponents
  -- Only one subcomponent
  Element : data String_Data;
end Data_Array.i;

-- Data structure type: Only the component implementation should be
-- used in the ports or parameters!

data Data_Structure
end Data_Structure;

data implementation Data_Structure.i;
subcomponents
  Component_1 : data String_Data;
  Component_2 : data W_String_Data;
  Component_3 : data Data_Array.i;
end Data_Structure.i;
```

Data components may also contain subprogram features. Depending on the AADL properties given by the user. These component may denote a protected object or a non protected object. In either case, they are used to model a data structure that can be handled only by the subprograms it exports (which are the feature of the data structure).

@include protected_object_types.texi

The example above shows an example of a protected data component (@code{Protected_Object.Impl}). The object has a single field (subcomponent) which is a simple data component. Note that the description of the feature subprograms of these data component is a little bit different from the description of classic subprograms: each feature subprogram must have a full access to the internal structure of the object type. To achieve this, we use the @code{require data access} facility of AADL. To model a non protected data component, user should simply change the @code{ARAO::Object_Kind => Protected;} into @code{ARAO::Object_Kind => Non_Protected;} in the implementation of data component.

@subsection C mapping rules

Data component declaration are mapped into C type declaration in the file @code{types.h}. In the following we give the C type corresponding to each data component type that could be modeled.

@subsubsection Simple types

Simple data components are mapped into an C type definition whose defining identifier is mapped from the component declaration identifier (with respect to the naming rules listed in @ref{Whole distributed application}) and whose parent subtypes is: @itemize @bullet

> @item @code{int} for boolean data types @item @code{int} for integer data types @item @code{float} for fixed point types @item @code{chat} for character data types

@end itemize

@subsubsection Bounded strings and wide strings

Bounded strings and wide strings are not supported in the C generator at this time.

@subsubsection Bounded arrays

Bounded arrays and wide strings are not supported in the C generator at this time.

@subsubsection Data structures

Data structures are mapped into a C structure defined in the file @file{types.h}. The identifier of the record type is mapped from the data component name with respect to the naming rules given in @ref{Whole distributed application}.

Each field defining identifier is mapped from the subcomponent name given in the data component implementation with the same naming rules. The type of the field is the C type mapped from the data corresponding component. The following example shows the C mapping of the data structure defined given earlier in this part.

@include data_struct.h.texi

@subsubsection Object types

Protected object types are mapped into an a C structure. We add automatically a member in the structure with the type @code{__po_hi_protected_id} and the name @code{protected_id}. This member will identify the protected type in the distributed system. All other members of the object are declared as in Data Structures (see previous subsection). The features subprograms of the object types are declared in the @file{types.h} file, whereas the body of these functions are defined in the @file{types.c} file. Moreover, the value of the @code{protected_id} must be initialized. This is done in the main function (@file{main.c}), before the initialization. All the naming conventions given in @ref{Whole distributed application} have to be respected. The following example shows the specification of the protected type mapped from the @code{Protected_Object.Impl} shown earlier in this part. We show the files @file{types.h}, @file{types.c} and @file{main.c} (that initialize the @code{protected_id} member of the structure.

@include toy_types.h.texi @include toy_types.c.texi @include toy_main.c.texi

Non protected object types are mapped similarly to protected object types. The only difference, is that instead of creating a protected type, we create a generic parameterless nested package.

CHAPTER 7

OSATE2-Ocarina plug-in

The OSATE2-Ocarina plugin brings all Ocarina's features to OSATE2: code generation, generation of Petri nets, mapping of AADL models to scheduling analysis tools, and constraint analysis using REAL.

7.1 Installation

An eclipse update site is available at: https://raw.github.com/yoogx/osate2-ocarina/master/org.osate.ocarina.update/

To install the plug-in, select "Install New Software", add the install site and then select the OSATE-OCARINA plug-in in the droplist.

7.2 Configuration

The plugin can be configured from the OSATE2 Preferences panel. The plug-in preferences are located under OSATE Preferences/Ocarina.

7.3 Usage

Right-click a system implementation in the Outline, then select Ocarina, and then the command to execute.

The output of the command (generated source code, etc.) is stored in the *ocarina_out* folder in your project.

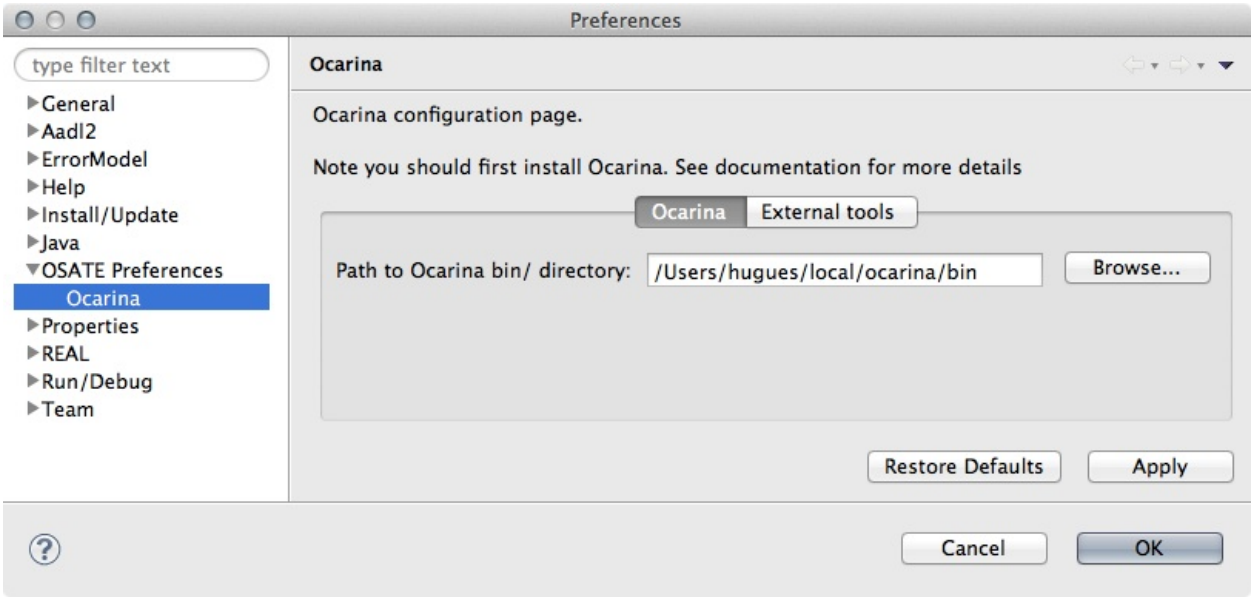

Fig. 1: OSATE Ocarina preferences

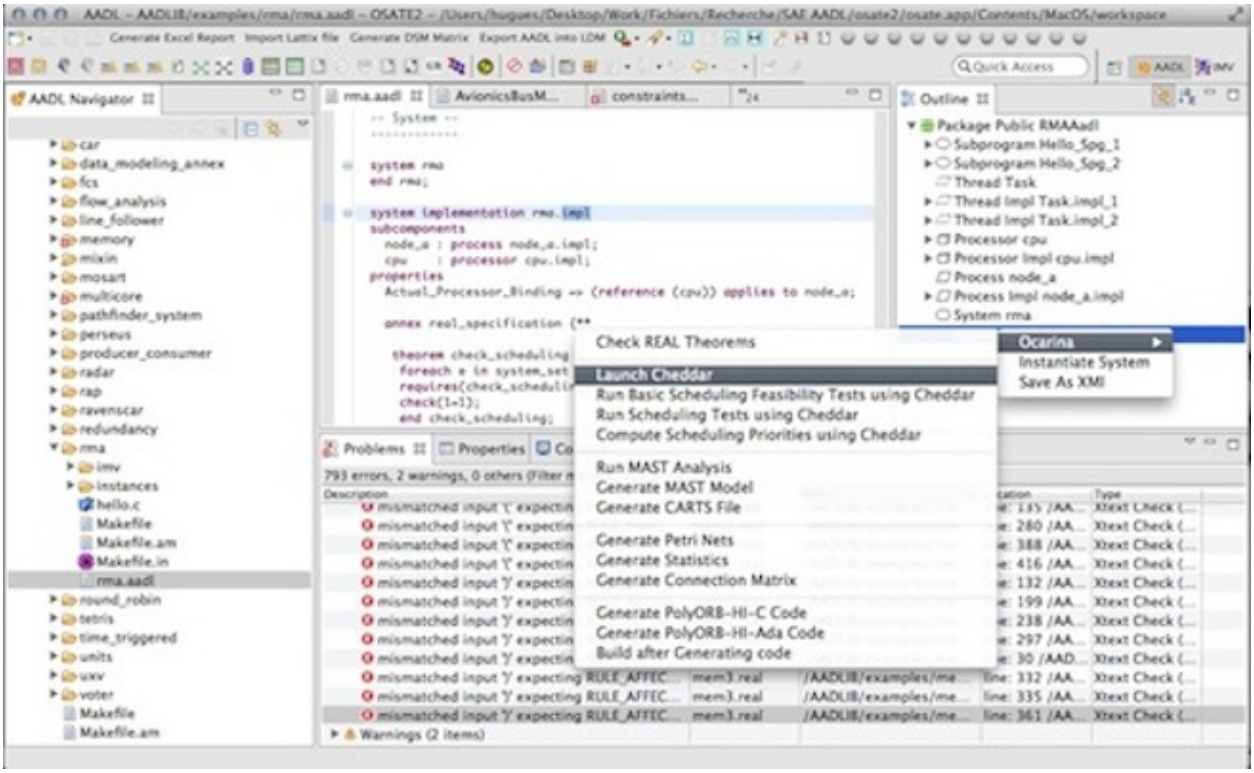

Fig. 2: OSATE Ocarina usage

CHAPTER 8

Python bindings for Ocarina

8.1 Ocarina Python bindings

Ocarina proposes Python bindings to its internal APIs. This binding is available if configured properly, first at compile-time, then at run-time.

At compile time, Ocarina must be configured with shared libraries support. Refer to the *Installation*;

At run-time, the following environment variables must be set up:

```
% export PATH=`ocarina-config --prefix`/bin:$PATH
% export OCARINA_PATH=`ocarina-config --prefix`
% export LD_LIBRARY_PATH=$OCARINA_PATH/lib:$LD_LIBRARY_PATH
% export PYTHONPATH=$OCARINA_PATH/include/ocarina/runtime/python:$OCARINA_PATH/lib:
↪$PYTHONPATH
```

8.2 Example

The following example illustrates the capabilities of the Python API, it implements a visitor that iterates of the AADL model elements:

```
#!/usr/bin/env python

# visitor.py: A simple script to visit all nodes of an AADL instance tree
#
# Note: this scripts require the docopt package

"""visitor.py: A simple script to visit all nodes of an AADL instance tree

Usage: visitor.py FILE
       visitor.py ( -h | --help )
       visitor.py --version
```

(continues on next page)

(continued from previous page)

```
Arguments:
    FILE  AADL file to process

Options:
    -h, --help  Help information
    --version   Version information

"""

from docopt import docopt
import sys

###################################
args = sys.argv[1:]
# XXX: we have to do a back-up of the command-line arguments prior to
# importing Ocarina, as its initialization phase will erase sys.argv
# To be investigated, see github issue #45 for details
###################################

import ocarina
import lmp
from ocarina_common_tools import *
import libocarina_python

def visitor (component, level):
    """
    This function visits an AADL component, and prints information
    about its features, subcomponents and properties.

    Args:
    component (str):  the NodeId of the component
    level (int): indentation level

    """

    print ' ' * level,'Visiting ',lmp.getInstanceName(component)[0]

    features=ocarina.AIN.Features(component)[0];
    if features is not None :
        print ' ' * level,' -> features:',features
        for feature in features :
            print ' ' * level,'   -> feature:',feature,', ',lmp.
↪getInstanceName(feature)[0]
            print ' ' * level,'      source feature: ', lmp.getInstanceName(ocarina.
↪getSourcePorts(feature)[0])[0]
            print ' ' * level,'      destination feature: ', lmp.
↪getInstanceName(ocarina.getDestinationPorts(feature)[0])[0]

    properties = ocarina.AIN.Properties(component)[0];
    if properties is not None :
        print ' ' * level,' -> properties:'
        for property in properties:
            print ' ' * level,'   ',ocarina.getPropertyValue(component,property)[0]

    subcomponents=ocarina.AIN.Subcomponents(component)[0];
    if subcomponents is not None :
```

(continues on next page)

(continued from previous page)

```
        print ' ' * level,' -> subcomponents:',subcomponents
        for subcomponent in subcomponents :
            print ' ' * level,'   -> ',subcomponent,",",lmp.
↪getInstanceName(subcomponent)[0]
            visitor(str(ocarina.AIN.Corresponding_Instance(subcomponent)[0]),level+3)

    print ' ' * level,'end of visit of ',component

def main ():
    # read  command line arguments

    arguments = docopt(__doc__, args, version="visitor.py 0.1")

    # build the repository path
    repo = arguments['FILE']

    err = ocarina.load(repo);        # load a file
    err = ocarina.analyze();         # analyze models
    err = ocarina.instantiate("");   # instantiate system

    print '-----------------------------------------------------'
    print 'Visit AADL Instance tree'
    print '-----------------------------------------------------'

    root=lmp.getRoot()[0]
    visitor(root,0)

if __name__ == "__main__":
    main ()
```

8.3 Python API description

The following lists all functions defined in the *ocarina* module

8.3.1 `ocarina` – Python binding to the Ocarina AADL processor

This module provides direct access to top-level functions of Ocarina to load, parse, instantiate AADL models, and to invoke backends.

ocarina.ocarina.**Backends = ('polyorb_hi_ada', 'polyorb_hi_c', 'real_theorem')**
List of supported backends, used by *generate*

class ocarina.ocarina.**Enum**

ocarina.ocarina.**add_real_library**(*libraryname*)

Parameters **libraryname** (*string*) – name of the REAL library file to include

ocarina.ocarina.**analyze**()
Analyze models

ocarina.ocarina.**generate**(*generator*)
Generate code

Parameters **generator** – one supported backends, from *Backends*

For instance, to use the PolyORB-HI/Ada backend, you may use the following

```
>>> generate (Backends.polyorb_hi_ada)
```

ocarina.ocarina.**getDestinationPorts**(*nodeId*)
Get the destination port associated to the feature_nodeId passed as parameter, in the case feature_nodeId participates in a connection.

ocarina.ocarina.**getPropertyValue**(*nodeId*, *propertyId*)
Get the value of the property

ocarina.ocarina.**getPropertyValueByName**(*nodeId*, *propertyString*)
Get the value of the property propertyString applied to model element nodeId.

ocarina.ocarina.**getSourcePorts**(*feature_nodeId*)
Get the source port associated to the feature_nodeId passed as parameter, in the case feature_nodeId participates in a connection.

ocarina.ocarina.**instantiate**(*root_system*)
Instantiate model, starting from root_system

Parameters **root_system** (*string*) – name of the root system to instantiate

ocarina.ocarina.**load**(*filename*)
Load a file

Parameters **filename** (*string*) – name of the file to be loaded, using Ocarina search path

E.g. to load "foo.aadl":

```
>>> load("foo.aadl")
```

ocarina.ocarina.**reset**()
Reset Ocarina internal state

Note: this function must be called before processing a new set of models.

ocarina.ocarina.**set_real_theorem**(*theorem_name*)
Set main REAL theorem

Parameters **theorem_name** (*string*) – name of the theorem

ocarina.ocarina.**status**()
Print Ocarina status

ocarina.ocarina.**version**()
Print Ocarina version

8.3.2 lmp – Port of Ellidiss LMP to Ocarina Python API

This module is an adaptation of Ellidiss LMP "Logical Model Processing" to Python.

ocarina.lmp.**getAliasDeclarations**()
Return the list of all the alias declaration defined in the current AADL project

ocarina.lmp.**getAnnexes**()
Return the list of all the annexes defined in the current AADL project

ocarina.lmp.**getComponentFullname**(*nodeId*)
Get the full qualified name of an AADL component

Parameters **nodeId** – the id of the component whose full qualified name is searched

For instance, to retrieve the full qualified name of MyComponent, retrieve its id (nodeId) and use the following

```
>>> getComponentFullname (nodeId)
```

ocarina.lmp.**getComponentImplementations**(*category*)
Return a list of component implementations defined in the current AADL project

Parameters **category** – one of the AADL category defined in the standard

For instance, to retrieve all the system implementations from the current project, you may use the following

```
>>> getComponentImplementations (System)
```

ocarina.lmp.**getComponentName**(*nodeId*)
Get the name of an AADL component

Parameters **nodeId** – the id of the component whose name is searched

For instance, to retrieve the name of MyComponent, retrieve its id (nodeId) and use the following

```
>>> getComponentName (nodeId)
```

ocarina.lmp.**getComponentTypes**(*category*)
Return a list of component types defined in the current AADL project

Parameters **category** – one of the AADL category defined in the standard

For instance, to retrieve all the system types from the current project, you may use the following

```
>>> getComponentTypes (System)
```

ocarina.lmp.**getFlowImplementations**()
Return the list of all the flow implementation defined in the current AADL project

ocarina.lmp.**getFlowSpecifications**()
Return the list of all the flow specification defined in the current AADL project

ocarina.lmp.**getImportDeclarations**()
Return the list of all the import declarations used in the current AADL project

ocarina.lmp.**getInModes**()
Return the list of all the in mode used in the current AADL project

ocarina.lmp.**getInstanceName**(*nodeId*)
Get the name of an AADL instance

Parameters **nodeId** – the id of the instance whose name is searched

For instance, to retrieve the name of MyInstance, retrieve its id (nodeId) and use the following

```
>>> getInstanceName (nodeId)
```

ocarina.lmp.**getInstances**(*category*)
Return a list of instances defined in the current AADL project

Parameters **category** – one of the AADL category defined in the standard

For instance, to retrieve all the system instances from the current project, you may use the following

```
>>> getInstances (System)
```

ocarina.lmp.**getModeTransitions**()
Return the list of all the mode transition defined in the current AADL project

ocarina.lmp.**getModes**()
Return the list of all the modes defined in the current AADL project

ocarina.lmp.**getNodeId**(*name*)
Get the Id of a component from its name

Parameters **name** – the AADL name of the node whose id is queried

For instance, to retrieve the id of MyHome, you may use the following

```
>>> getNodeId (MyHome)
```

ocarina.lmp.**getPackages**()
Return the list of all the packages defined in the current AADL project

ocarina.lmp.**getPropertyConstants**(*propertySetId*)
Return the list of all the constant property defined in the provided property set

Parameters **propertySetId** – the nodeId of the property set in the current AADL project to search in

For instance, to retrieve all the constant property from property set propertySet, retrieve its id (propertySetId) and use the following

```
>>> getPropertyConstants (propertySetId)
```

ocarina.lmp.**getPropertyDefinitions**(*propertySetId*)
Return the list of all the property declaration defined in the provided property set

Parameters **propertySetId** – the nodeId of the property set in the current AADL project to search in

For instance, to retrieve all the property declaration from property set propertySet, retrieve its id (propertySetId) and use the following

```
>>> getPropertyDefinitions (propertySetId)
```

ocarina.lmp.**getPropertySets**()
Return the list of all the property set defined in the current AADL project

ocarina.lmp.**getPropertyTypes**(*propertySetId*)
Return the list of all the property types defined in the provided property set

Parameters **propertySetId** – the nodeId of the property set in the current AADL project to search in

For instance, to retrieve all the property types from property set propertySet, retrieve its id (propertySetId) and use the following

```
>>> getPropertyTypes (propertySetId)
```

ocarina.lmp.**getPrototypeBindings**()
Return the list of all the prototype bindings defined in the current AADL project

ocarina.lmp.**getPrototypes**()
Return the list of all the prototypes defined in the current AADL project

ocarina.lmp.**getRoot**()
Get the Id of the current root instantiated model

CHAPTER 9

Editor support

The AADL modes for Emacs and vim provide syntax coloration and automatic indentation features when editing AADL files.

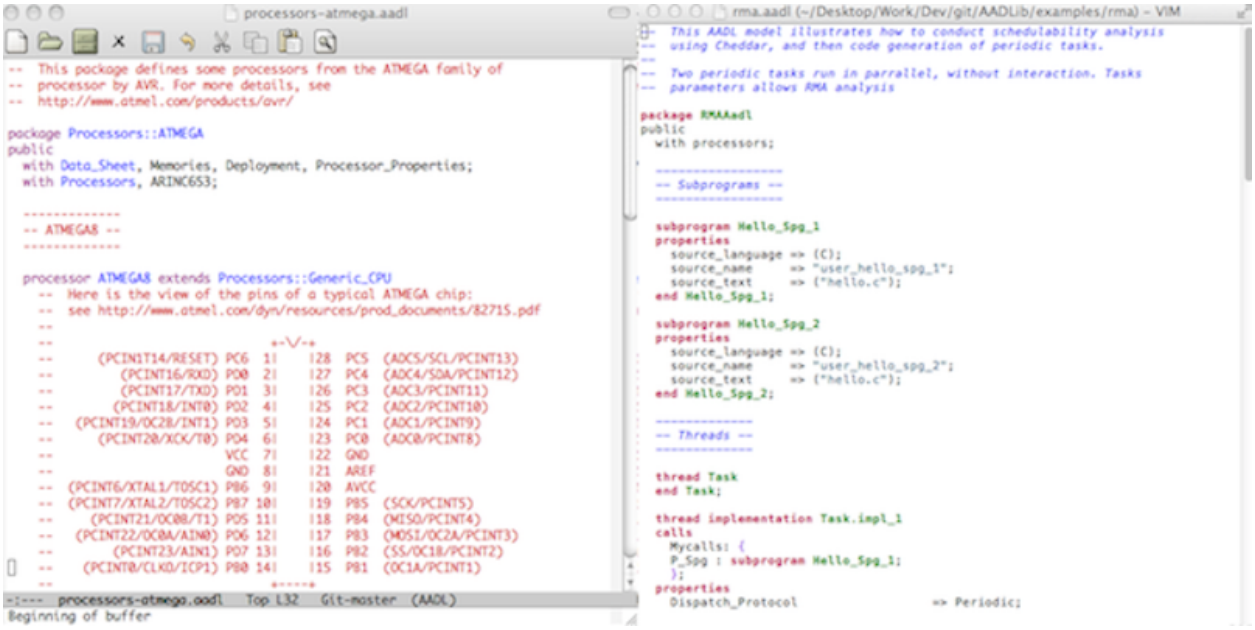

Fig. 1: AADL mode for emacs and vim

9.1 Emacs

To load the AADL mode for Emacs, you need to add the following line to your emacs configuration file (usually located in `~/.emacs`)

```
(load "/path/to/this/file.el")
```

For more details on this mode, please refer to the emacs contextual help.

9.2 vim

The AADL mode for vim is made of two files aadl.vim: one for syntactic coloration, and the other for indentation. The file for indentation must be placed into ~/.vim/indent/ while the one for syntactic coloration must be placed into `~/.vim/syntax/`

To load the AADL mode whenever you edit AADL files, create a file named `~/.vim/filetype.vim`, in which you write:

```
augroup filetypedetect
        au BufNewFile,BufRead *.aadl    setf aadl
augroup END
```

For more details, please read the documentation of vim.

CHAPTER 10

Ocarina property sets

10.1 Deployment

```
property set Deployment is

  Allowed_Transport_APIs : type enumeration
    (BSD_Sockets,
     SpaceWire);
  -- Supported transport API

  Transport_API : Deployment::Allowed_Transport_APIs applies to (bus);
  --  Transport API of a bus

  Location : aadlstring applies to (processor, device);
  --  Processor IP address (BSD_Sockets specific)

  Port_Number : aadlinteger applies to (process, device);
  --  IP port number of a process (BSD_Sockets specific)

  Protocol_Type : type enumeration (iiop, diop);
  --  Supported communication protocols
  Protocol : Deployment::Protocol_Type applies to (system);

  Allowed_Execution_Platform : type enumeration
   (Native,                    -- Native platforms (GNU/Linux, Solaris, Windows...)
    Native_Compcert,           -- Native platforms using the Compcert compiler
    bench,                     -- Benchmark platform (native with instrumentation).
    GNAT_Runtime,              -- Use a GNAT Runtime, e.g. from the Ada_Drivers_Library
    LEON_ORK,
    LEON_RTEMS,                -- LEON2 board or tsim-leon (RTEMS)
    LEON_RTEMS_POSIX,          -- LEON2 board or tsim-leon (RTEMS)
    LEON3_SCOC3,               -- LEON3 with RTEMS for SCOC3
    LEON3_XTRATUM,             -- LEON3 with Xtratum
    LEON3_XM3,                 -- RTEMS for XTRATUM/LEON3
```

(continues on next page)

(continued from previous page)

```
   LEON_GNAT,                  -- LEON2 board or qemu (GNATPRO/HI-E)
   LINUX32,                    -- Linux 32 bits
   LINUX_DLL,                  -- Linux Dynamic Library
   LINUX32_XENOMAI_NATIVE,     -- Linux 32 bits with native Xenomai
   LINUX32_XENOMAI_POSIX,      -- Linux 32 bits with Xenomai and POSIX skin
   LINUX64,                    -- Linux 64 bits
   ERC32_ORK,                  -- ERC32 board or tsim-erc32 (ORK)
   X86_RTEMS_POSIX,            -- x86 under RTEMS with POSIX layer
   X86_LINUXTASTE,             -- TASTE-specific linux distribution
   MARTE_OS,                   -- MaRTE OS
   WIN32,                      -- WIN32
   VXWORKS,                    -- VXWORKS
   FREERTOS,                   -- FREERTOS
   AIR                         -- AIR Hypervisor, by GMV
   );
--  Supported platforms

Execution_Platform : Deployment::Allowed_Execution_Platform
  applies to (all);
--  Execution platform of a processor

Ada_Runtime : aadlstring applies to (processor);
--  If Execution_Platform is set to GNAT_Runtime, this property
--  points to the name of the GNAT Project file that configures the
--  Ada_Runtime, e.g. from the Ada Device Drivers project.

USER_CFLAGS : aadlstring applies to (processor);
USER_LDFLAGS : aadlstring applies to (processor);
--  User defined CFLAGS and LDFLAGS

Supported_Execution_Platform : list of Deployment::Allowed_Execution_Platform
  applies to (device);
--  List execution platforms supported by a particular driver

USER_ENV: aadlstring applies to (processor);
--  Additional configuration parameters passed as environment
--  variables as part of the build phase. These env. variables are
--  passed in the generated makefiles.

Runtime : type enumeration
  (PolyORB_HI_C,
   PolyORB_HI_Ada,
   POK);
--  List of supported runtime

Supported_Runtime : Deployment::Runtime applies to (all);
--  List the runtime compatible with the component

Priority_Type : type aadlinteger 0 .. 255;

Priority : Deployment::Priority_Type applies to (data, thread);
--  Thread and data component priority

Driver_Name : aadlstring applies to (device);

Configuration : aadlstring applies to (device, thread);
```

(continues on next page)

(continued from previous page)

```
  Config : aadlstring applies to (device);

  ASN1_Module_Name : aadlstring applies to (all);

  Help : aadlstring applies to (all);

  Version : aadlstring applies to (all);

  Configuration_Type : classifier (data) applies to (all);

end Deployment;
```

10.2 Ocarina_Config

```
--  Property set containing the configuration properties of Ocarina.
--  This property set is not intended to be used by the AADL model of
--  an application, but, by the AADL model of its scenario.

property set Ocarina_Config is

  Generator_Type : type enumeration
   (PolyORB_QoS_Ada,
    PolyORB_HI_Ada,
    PolyORB_HI_C,
    PolyORB_HI_RTSJ,
    POK_C,
    Xtratum_Configuration,
    Petri_Nets);

  Generator : Ocarina_Config::Generator_Type applies to (system);
  --  The code generator that will be used for the current application

  Generator_Options_Type : type enumeration
   (gprof,
    ASN1);

  Generator_Options : list of Ocarina_Config::Generator_Options_Type
    applies to (system);
  --  Code generation options.

  AADL_Files : list of aadlstring applies to (system);
  --  List of the AADL source files used by the current application

  Cheddar_Properties  : constant aadlstring => "Cheddar_Properties";
  Data_Model          : constant aadlstring => "Data_Model";
  Deployment          : constant aadlstring => "Deployment";
  POK_Properties      : constant aadlstring => "pok_properties";
  ARINC653_Properties : constant aadlstring => "arinc653";
  ASSERT_Properties   : constant aadlstring => "ASSERT_Properties";
  TASTE_Properties    : constant aadlstring => "taste_properties";
  --  List of the predefined NON STANDARD property sets that can be used
  --  by an AADL application.

  Needed_Property_Sets : list of aadlstring applies to (system);
```

(continues on next page)

(continued from previous page)

```
  --  The actual property sets needed by one particular application.
  --  This avoid to parse systematically all the predefined non
  --  standard property sets. The user can also give the name of a
  --  custom property set (which may be used by many AADL models),
  --  provided that the value of the string matches exactly the base
  --  name (without the .aadl suffix and in a case-sensitive manner)
  --  the user property sey file name and provided that this property
  --  set file is located in the same directory as the Ocarina
  --  non-standard property sets.

  Ocarina_Driver_Library : constant list of aadlstring =>
  ("devices.aadl",
    "buses",
    "base_types",
    "exarm-ni-6071e-analog.aadl",
    "grspw.aadl",
    "rasta-serial.aadl",
    "sockets-rtems-ne2000.aadl",
    "exarm-ni-6071e-digital.aadl",
    "gruart.aadl",
    "rasta-spacewire.aadl",
    "tcp_protocol.aadl",
    "generic-keyboard.aadl",
    "leon-eth.aadl",
    "scoc3-spacewire.aadl",
    "udp-exarm.aadl",
    "generic_bus.aadl",
    "leon-serial.aadl",
    "sd-spw-usb.aadl",
    "generic_native.aadl",
    "native_uart.aadl",
    "serial-raw.aadl",
    "gr_cpci_x4cv.aadl",
    "rasta-1553.aadl",
    "sockets-raw.aadl",
    "grspw_packet.aadl",
    "apbuart.aadl",
    "greth.aadl",
    "stardundee.aadl");

  Root_System_Name : aadlstring applies to (system);
  --  If present, indicates the name of the root of the instance tree

  AADL_Version_Type : type enumeration (AADLv1, AADLv2);

  AADL_Version : Ocarina_Config::AADL_Version_Type applies to (system);
  --  AADL version of the model

  Use_Components_Library : aadlboolean applies to (system);

  Referencial_Files : list of aadlstring applies to (system);
  --  The list of referencial files used to compute the regression test

  Timeout_Property : Time applies to (system);
  --  The timeout used to stop an execution

  Annex_Type : type enumeration
```

(continues on next page)

(continued from previous page)

```
    (annex_all,
     annex_none,
     behavior_specification,
     real_specification,
     emv2));
     --  Designates the list of annexes supported in ocarina. annex_all
     --  and annex_none designate respectively all supported annexes
     --  and none of them for properties that accept this kind of
     --  designation.

  Enable_Annexes : list of Ocarina_Config::Annex_Type applies to (system);
  --  List of annexes to be enabled in the parsed model
end Ocarina_Config;
```

CHAPTER 11

GNU Free Documentation License

Version 1.1, March 2000

11.1 Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

11.2 Applicability and Definition

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

11.3 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

11.4 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

11.5 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
4. Preserve all the copyright notices of the Document.
5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
8. Include an unaltered copy of this License.
9. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

13. Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
14. Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

11.6 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

11.7 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

11.8 Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document,

provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

11.9 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

11.10 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

11.11 Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

11.12 How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

Symbols

A

B

E

G

I

L

O

R

S

V