<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Getting Started</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Why Comet?</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Dependencies</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Installation</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>5.1 Cloning the Github repository</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>5.2 Using a RPM package</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>5.3 Automatic deployment using Docker</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Running the STH server</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Registering historical context information</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Raw context information retrieval</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>Consuming aggregated time series context information</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>Removing raw and aggregated time series context information</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>Migrating raw and aggregated time series context information</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>Performance tests coverage</td>
<td>29</td>
</tr>
<tr>
<td>13</td>
<td>Additional resources</td>
<td>31</td>
</tr>
<tr>
<td>14</td>
<td>Contribution guidelines</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>14.1 Overview</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>14.2 How to contribute</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>14.3 Coding guidelines</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>14.4 Branching model</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>14.5 Changelog</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>14.6 Testing</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>14.7 Continuous testing</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>14.8 Code coverage</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>14.9 Code complexity</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>14.10 Source code documentation</td>
<td>35</td>
</tr>
</tbody>
</table>
Introduction

The Short Time Historic (STH, aka. Comet) is a component of the FIWARE ecosystem in charge of managing (storing and retrieving) historical raw and aggregated time series information about the evolution in time of context data (i.e., entity attribute values) registered in an Orion Context Broker instance.

All the communications between the STH and the Orion Context Broker as well as between the STH and any third party (typically for data retrieval) use standardized NGSI9 and NGSI10 interfaces.
Getting Started

Although the STH supports the storing and retrieval of raw context information, this is, the concrete entity attribute value changes which where registered in an Orion Context Broker instance over time, its main capability and responsibility is the generation of aggregated time series context information about the evolution in time of those entity attribute values.

Regarding the generation of aggregated time series context information, the STH manages 4 main concepts:

1. **Resolution or aggregation period**: The time period by which the aggregated time series information is grouped. Possible valid resolution values supported by the STH are: month, day, hour, minute and second.

2. **Origin**: For certain resolution, it is the origin of time for which the aggregated time series context information applies. For example, for a resolution of minutes, a valid origin value could be: 2015-03-01T13:00:00.0002, meaning the 13th hour of March, the 3rd, 2015. The origin is stored using UTC time to avoid locale issues.

3. **Offset**: For certain resolution, it is the offset from the origin for which the aggregated time series context information applies. For example, for a resolution of minutes and an origin 2015-03-01T13:00:00.0002, an offset of 10 refers to the 10th minute of the concrete hour pointed by the origin. In this example, there would be a maximum of 60 offsets from 0 to 59 corresponding to each one of the 60 minutes within the concrete hour.

4. **Samples**: For certain resolution, origin and offset, it is the number of samples, values, events or notifications available for that concrete offset from the origin.

All these concepts will appear clearer as you read the rest of the documentation about the STH component but it is important to introduce them from the very beginning since they apply far and wide all around the component internals and exposed APIs.
Since most of the components which conform the FIWARE ecosystem have astrological names, we decided to follow that path in the case of the STH too. Since the STH is in charge of collecting historical context information about the values certain entity attributes took over time, we decided to name it “Comet”, in reference to the tails comets leave on their way as they move.
CHAPTER 4

Dependencies

The STH component is a Node.js application which depends on certain Node.js modules as stated in the project.json file. Currently, the STH component currently supports Node 0.10 and Node 4.0.

Apart from these Node.js modules, the STH component also needs a running MongoDB instance where the raw and aggregated time series context information is stored for its proper functioning. Since the STH component uses MongoDB update operators (see http://docs.mongodb.org/v2.6/reference/operator/update/) such as the $max and the $min update operators which were introduced in version 2.6, there is a dependency of the STH component with this concrete version of the MongoDB instance where the raw and aggregated time series context information data will be stored. Consequently, a MongoDB version \( \geq 2.6 \) is needed to store the raw and aggregated time series context information.
The STH component provides 3 alternatives for its installation:

1. Cloning the Github repository
2. Using a RPM package
3. Automatic deployment using Docker

### 5.1 Cloning the Github repository

To install the STH component cloning the Github repository, please follow the next steps:

1. Clone the repository:

The STH component server is ready to be started as a Node application.

### 5.2 Using a RPM package

In the next sections we detail the steps to follow to install, update and remove the STH component using a RPM package.

#### 5.2.1 Package generation

**Prerequisites:** To generate the RPM package from the STH component sources it is needed to have the rpm build tools (rpmbuild executable), Node and the npm utilities, as well as an Internet connection to download the required Node modules.

To generate the RPM package for the STH component, execute the following command from the root of the STH component:

```
./rpm/create-rpm.sh -v <version> -r <release>
```

If everything goes fine, a new RPM package such as `./rpm/RPMS/x86_64/fiware-sth-comet-<version>-<release>.x86_64.rpm` will be created.

Execute `./rpm/create-rpm.sh -h` for more information about the RPM package creation script.
5.2.2 Installation, upgrade and removal

**Prerequisites:** Node is needed to install the generated STH component RPM package.

To install or upgrade the STH component, execute:

```
sudo rpm -Uvh fiware-sth-comet-<version>-<release>.x86_64.rpm
```

After the installation, the following files and directories are created:

```
/etc/init.d
-- sth

/etc/logrotate.d
-- logrotate-sth-daily

/var/log/sth

/var/run/sth

/opt/sth
-- conf
  |  -- <empty> Here is where instances are configured
  -- node_modules
  |  -- <node modules directory structure and files>
-- package.json
-- src
  -- <STH SW files>
```

To remove a previous STH component installation, execute:

```
sudo rpm -e fiware-sth-comet
```

5.2.3 Configuration

STH is able to start multiple instances using the sth service script by adding and configuring certain files as detailed next.

To start multiple instances, one configuration file per instance has to be included in the `/opt/sth/conf` directory. It is important to note that the default installation includes preconfigured instances.

It is important to change the `STH_PORT` value included in the configuration files to a value not used by other STH instances/services. It is also a good practice to change the `LOG_FILE_NAME` value to avoid getting the logs from several instances mixed.

The init.d service script includes the following operations:

- **start** (sudo `/sbin/service sth start [<instance>]): if `<instance>` is not provided, the script starts an instance per configuration file found in the `/opt/sth/conf` directory matching the `sth_*`.conf template. If `<instance>` is provided, a configuration file named `sth_<instance>.conf` is searched in the `/opt/sth/conf` directory and the corresponding instance is started.

- **stop** (sudo `/sbin/service sth stop [<instance>]): if `<instance>` is not provided, the script stops all the instances by listing all pid files under `/var/run/sth` matching the pattern `sth_*`.pid. If `<instance>` is provided, the scripts stops the instance with the associated pid file `/var/run/sth/sth_<instance>.pid`

- **status** (sudo `/sbin/service sth status [<instance>]): The status operation shows information about one or more running instances following the same procedure detailed in the stop operation.

- **restart** (sudo `/sbin/service sth stop [<instance>]): The restart operation executes a stop operation followed by a start operation according to the procedure detailed in those operations.
An example `sth_default.conf` file has been included in this Github repository to guide the STH instance configuration.

Last but not least, the STH process (a node process) runs the as `sth` user.

### 5.3 Automatic deployment using Docker

To ease the testing and deployment of the STH component we have prepared a Docker repository which can be found at https://registry.hub.docker.com/u/fiwareiotplatform/iot-sth/, including all the information needed to try and to deploy the STH component via the execution of a simple Docker command.

On the other hand a `Dockerfile` and a `docker-compose.yml` files have also been included in this very repository to quickly and easily start your own instance of the STH component, even including the associated MongoDB instance where all the data will be stored.

To do it, follow the next steps once you have installed Docker in your machine:

1. Navigate to the path where this repository was cloned.
2. Compose and run the new STH component image:

    ```bash
docker-compose up
    ```
Running the STH server

To run the STH server, just execute from the STH directory the following command:

```
./bin/sth
```

The STH component provides the user with 2 mechanisms to configure the component to the concrete needs of the user:

- Environment variables, which can be set assigning values to them or using the `sth_default.conf` file if a packaged version of the STH component is used.
- The `config.js` file located at the root of the STH component code, a JSON formatted file including the configuration properties.

It is important to note that environment variables, if set, take precedence over the properties defined in the `config.js` file.

On the other hand, it is also important to note that the aggregation resolutions can only be configured using the `config.js` file and consequently this is the preferred way to configure the STH component behavior. The mentioned resolutions can be configured using the `config.server.aggregation` property in the `config.js` file including the desired resolution to be used when aggregating data. Accepted resolution values include: month, day, hour, minute and second.

In case of preferring using environment variables, the script accepts the following parameters as environment variables:

- `STH_HOST`: The host where the STH server will be started. Optional. Default value: “localhost”.
- `STH_PORT`: The port where the STH server will be listening. Optional. Default value: “8666”.
- `FILTER_OUT_EMPTY`: A flag indicating if the empty results should be removed from the response. Optional. Default value: “true”.
- `TEMPORAL_DIR`: A relative path from the STH home directory to a directory where the temporary files generated by the STH component are stored. These files are generated before returning them when the `filetype` is included in any data retrieval request. Default value: “temp”.
- `DEFAULT_SERVICE`: The service to be used if not sent in the Orion Context Broker notifications. Optional. Default value: “testservice”.
- `DEFAULT_SERVICE_PATH`: The service path to be used if not sent in the Orion Context Broker notifications. Optional. Default value: “/testservicepath”.
- `DATA_MODEL`: The STH component supports 3 alternative data models when storing the raw and aggregated data into the database: 1) one collection per attribute, 2) one collection per entity and 3) one collection per service path. The possible values are: “collection-per-attribute”, “collection-per-entity” and “collection-per-service-path” respectively. Default value: “collection-per-entity”.

15
• **DB_USERNAME**: The username to use for the database connection. Optional. Default value: "".

• **DB_PASSWORD**: The password to use for the database connection. Optional. Default value: "".

• **DB_URI**: The URI to use for the database connection. This does not include the ‘mongo://’ protocol part (see a couple of examples below). Optional. Default value: “localhost:27017”.

• **REPLICA_SET**: The name of the replica set to connect to, if any. Default value: "".

• **DB_PREFIX**: The prefix to be added to the service for the creation of the databases. More information below. Optional. Default value: “sth_”.

• **COLLECTION\PREFIX**: The prefix to be added to the collections in the databases. More information below. Optional. Default value: “sth_”.

• **POOL_SIZE**: The default MongoDB pool size of database connections. Optional. Default value: “5”.

• **WRITE_CONCERN**: The **write concern policy** to apply when writing data to the MongoDB database. Default value: “1”.

• **SHOULD_STORE**: Flag indicating if the raw and/or aggregated data should be persisted. Valid values are: “only-raw”, “only-aggregated” and “both”. Default value: “both”.

• **SHOULD_HASH**: Flag indicating if the raw and/or aggregated data collection names should include a hash portion. This is mostly due to MongoDB’s limitation regarding the number of bytes a namespace may have (currently limited to 120 bytes). In case of hashing, information about the final collection name and its correspondence to each concrete service path, entity and (if applicable) attribute is stored in a collection named \( COLLECTION\PREFIX + "\text{collection_names}" \). Default value: “false”.

• **TRUNCATION_EXPIRE_AFTER SECONDS**: Data from the raw and aggregated data collections will be removed if older than the value specified in seconds. In case of raw data the reference time is the one stored in the \( \text{recvTime} \) property whereas in the case of the aggregated data the reference of time is the one stored in the \( _\text{id}.\text{origin} \) property. Set the value to 0 not to apply this time-based truncation policy. Default value: “0”.

• **TRUNCATION_SIZE**: The oldest raw data (according to insertion time) will be removed if the size of the raw data collection gets bigger than the value specified in bytes. Set the value to 0 not to apply this truncation policy. Take into consideration than the “size” configuration parameter is mandatory in case size collection truncation is desired as required by MongoDB. Default value: “0”. Notice that this configuration parameter does not affect the aggregated data collections since MongoDB does not currently support updating documents in capped collections which increase the size of the documents. Notice also that in case of the raw data, the size-based truncation policy takes precedence over the TTL one. More concretely, if a size limitation is set, the previous time expiration is ignored for the raw data collections since currently MongoDB does not support TTL in capped collections. Default value: “0”.

• **TRUNCATION_MAX**: The oldest raw data (according to insertion time) will be removed if the number of documents in the raw data collections goes beyond the specified value. Set the value to 0 not to apply this truncation policy. Notice that this configuration parameter does not affect the aggregated data collections since MongoDB does not currently support updating documents in capped collections which increase the size of the documents. Default value: “0”.

• **IGNORE_BLANK_SPACES**: Attribute values to one or more blank spaces should be ignored and not processed either as raw data or for the aggregated computations. Default value: “true”.

• **LOGOPS_LEVEL**: The log level to use. Possible values are: “DEBUG”, “INFO”, “WARN”, “ERROR” and “FATAL”. Since the STH component uses the logops package for logging, for further information check out the logops npm package information online. Default value: “INFO”.

• **LOGOPS_FORMAT**: The log format to use. Possible values are: “json” (writes logs as JSON), “dev” (for development, used when the NODE_ENV variable is set to ‘development’). Since the STH component uses the logops package for logging, for further information please check out the logops npm package information online. Default value: “json”.

**Chapter 6. Running the STH server**
• **PROOF_OF_LIFE_INTERVAL**: The time in seconds between proof of life logging messages informing that the server is up and running normally. Default value: “60”.

For example, to start the STH server listening on port 7777, connecting to a MongoDB instance listening on my-mongo.com:27777 and without filtering out the empty results, use:

```
STH_PORT=7777 DB_URI=mymongo.com:27777 FILTER_OUT_EMPTY=false ./bin/sth
```

On the other hand, in case of connecting to a MongoDB replica set composed of 3 machines with IPs addresses 1.1.1.1, 1.1.1.2, 1.1.1.3 listening on ports 27771, 27772 and 27773, respectively, use:

```
DB_URI=1.1.1.1:27771,1.1.1.2:27772,1.1.1.3:27773 ./bin/sth
```

The STH component creates a new database for each **service**. The name of these databases will be the concatenation of the **DB_PREFIX** environment variable and the service, using an underscore (_) as the separator.

As already mentioned, all this configuration parameters can also be adjusted using the `config.js` file whose contents are self-explanatory.

It is important to note that there is a **limitation** of 120 bytes for the namespaces (concatenation of the database name and collection names) in MongoDB. Related to this, the STH generates the collection names using 2 possible mechanisms:

1. **Plain text**: In case the **SHOULD_HASH** configuration parameter is set to **false** (the default option), the collection names are generated as a concatenation of the **COLLECTION_PREFIX** plus the service path plus the entity id plus the entity type plus `.aggr` for the collections storing the aggregated data. The length of the collection name plus the **DB_PREFIX** plus the database name (or service) should not be more than 120 bytes using UTF-8 format or MongoDB will complain and will not create the collection, and consequently no data would be stored by the STH. A warning message is logged in case this happens.

2. **Hash based**: In case the **SHOULD_HASH** option is set to something distinct from **false**, the collection names are generated as a concatenation of the **COLLECTION_PREFIX** plus a generated hash plus `.aggr` for the collections of the aggregated data. To avoid collisions in the generation of these hashes, they are forced to be 20 bytes long at least. Once again, the length of the collection name plus the **DB_PREFIX** plus the database name (or service) should not be more than 120 bytes using UTF-8 or MongoDB will complain and will not create the collection, and consequently no data would be stored by the STH. The hash function used is SHA-512. A warning message is logged in case this happens.

In case of using hashes as part of the collection names and to let the user or developer easily recover this information, a collection named **DB_COLLECTION_PREFIX + _collection_names** is created and fed with information regarding the mapping of the collection names and the combination of concrete services, service paths, entities and attributes.
Registering historical context information

There are 2 main ways to register raw and aggregated time series context information into the STH component:

1. The formal option.
2. The minimalistic option.

The formal option uses an additional component of the FIWARE ecosystem as it is Cygnus. Cygnus is the component in charge of persisting in distinct repositories or data storages the context information managed by an Orion Context Broker instance over time. To do it, Cygnus supports distinct connectors (aka., sinks) to many external repositories or data storages such as Apache Hadoop, Apache Kafka, CartoDB, CKAN, MySQL, PostgreSQL, amongst others.

To register the raw and aggregated time series context information into the STH component using Cygnus, we implemented 2 additional sinks such as:

1. The MongoDB sink: the MongoDB sink is in charge of persisting into MongoDB databases the desired context information as it is registered into an Orion Context Broker instance. Once it is stored in the desired MongoDB databases, the STH component is able to make it available by means of the raw context information API it provides which will be presented in the next sections.

2. The STH sink: the STH sink is in charge of persisting into MongoDB databases the desired aggregated time series context information as it is registered into an Orion Context Broker instance. The STH sink pre-aggregates the context information according to the configured resolutions making the retrieval of this aggregated time series context information almost instantaneous, using the API the STH component exposes and which will be presented in the next sections.

To properly configure and use the MongoDB and the STH sinks to register raw and aggregated time series context information into MongoDB databases susceptible of being retrieved by the STH component, please refer to the following documentation provided at the Cygnus Github repository site:

- Connecting Orion Context Broker and Cygnus
- NGSIMongoSink
- NGSISTHSink

The second and so-called minimalistic option to register raw and aggregated time series context information susceptible of being retrieved by the STH component consists on using the STH component itself to receive and process the notifications sent by an Orion Context Broker instance as the values of the entity attributes of interest change over time.

As a way to subscribe the STH component instance to the entity attributes of interest, a request such as the following one has to be sent to the Orion Context Broker instance whose raw and aggregated time series context information wants to be managed by the STH component:

Notice that in the previous subscriptions we are using templates instead of real values. These templates should be substituted by the desired values in each concrete case.
It is important to note that the subscription expire and must be re-enabled. More concretely, the duration property sets the duration of the subscription.

On the other hand, for the time being the STH component only is able to manage notifications in JSON format and consequently it is very important to set the Accept header to application/json.

Last but not least, the throttling makes it possible to control the frequency of the notifications. Depending on the resolution of the aggregated time series context information you are interested in, the throttling should be fine-tuned accordingly. For example, it may make no sense to set the minimum resolution in the STH component to second but set the throttling to PT60s (60 seconds), since with this configuration 1 value update will be notified every 60 seconds (1 minute) the most, and corresponding the minimum recommended resolutions should be minute.

Raw context information retrieval

The STH component exposes an HTTP REST API to let external clients query the raw context information registered. A typical URL querying for this information using a GET request is the following:

Notice that in the previous URL we have used some templates between < and > which should be substituted by the corresponding real values.

The requests for raw context information can use the following query parameters:

- **lastN**: Only the requested last entries will be returned. It is a mandatory parameter if no `hLimit` and `hOffset` are provided.

- **hLimit**: In case of pagination, the number of entries per page. It is a mandatory parameter if no `lastN` is provided.

- **hOffset**: In case of pagination, the offset to apply to the requested search of raw context information. It is a mandatory parameter if no `lastN` is provided.

- **dateFrom**: The starting date and time from which the raw context information is desired. It is an optional parameter.

- **dateTo**: The final date and time until which the raw context information is desired. It is an optional parameter.

- **filetype**: The raw context information can be requested as a file setting this query parameter to the desired file type. Currently, the only supported value and file type is `csv`. It is an optional parameter.

An example response provided by the STH component to a request such as the previous one could be the following:

Notice that in the previous request a paginated response has been requested with a limit of 3 entries and an offset of 0 entries (first page).

It is important to note that if a valid query is made but it returns no data (for example because there is no raw context information for the specified time frame), a response with code 200 is returned including an empty `values` property array, since it is a valid query.
Consuming aggregated time series context information

The STH component exposes an HTTP REST API to let external clients query the aggregated time series context information.

A typical URL querying for this information using a GET request is the following:

```
Notice that in the previous URL we have used some templates between < and > which should be substituted by the corresponding real values.

The requests for aggregated time series context information can use the following query parameters:

- **aggrMethod**: The aggregation method. The STH component supports the following aggregation methods: max (maximum value), min (minimum value), sum (sum of all the samples) and sum2 (sum of the square value of all the samples) for numeric attribute values and occur for attributes values of type string. Combining the information provided by these aggregated methods with the number of samples, it is possible to calculate probabilistic values such as the average value, the variance as well as the standard deviation. It is a mandatory parameter.

- **aggrPeriod**: Aggregation period or resolution. A fixed resolution determines the origin time format and the possible offsets. It is a mandatory parameter.

- **dateFrom**: The starting date and time from which the aggregated time series information is desired. It is an optional parameter.

- **dateTo**: The final date and time until which the aggregated time series information is desired. It is an optional parameter.

An example response provided by the STH component to a request such as the previous one (for a numeric attribute value) could be the following:

```
In the previous example response, aggregated time series context information for a resolution of 'second is returned. This information has as its origin the 46nd minute, of the 2nd hour of January, the 1st, 2016. And includes data for the 13th second, for which there is a sample and the sum (and value of that sample) is 34.59.

On the other hand, if the attribute value was of type string, a query such as the following (with aggrMethod as occur) could be sent to the STH component (properly substituting the templates between < and > and the example entity id, type and attribute name values):

```
An example response for the previous request could be:

```
It is important to note that if a valid query is made but it returns no data (for example because there is no aggregated data for the specified time frame), a response with code 200 is returned including an empty values property array, since it is a valid query.

Another very important aspect is that since the strings are used as properties in the generated aggregated data, the limitations to this regard imposed by MongoDB must be respected. More concretely: “In some cases, you may wish to build a BSON object with a user-provided key. In these situations, keys will need to substitute the reserved $ and
characters. Any character is sufficient, but consider using the Unicode full width equivalents: U+FF04 (i.e. “”') and U+FF0E (i.e. “”’).

Due to the previous MongoDB limitation, if the textual values stored in the attributes for which aggregated context information is being generated contain the $ or the . characters, they will be substituted for their Javascript Unicode full width equivalents, this is: \uFF04 instead of $ and \uFF0E instead of ..
Removing raw and aggregated time series context information

Last, but not least, the STH component exposes through its REST API the possibility to remove previously registered raw and aggregated data. Due to the sensible nature of these operations, they should be used with caution since their effects cannot be undone.

The STH component exposes 3 main URLs for data removal, all of them invoked using DELETE as the HTTP method and including the service and service path information as headers (Fiware-Service and Fiware-ServicePath headers, respectively) such as in the curl examples included in the previous sections. The provided URLs are the following ones:

1. Removing all the data associated to certain service and service path:
   
   http://<sth-host>:<sth-port>/STH/v1/contextEntities

2. Removing all the data associated to certain entity, service and service path:

   http://<sth-host>:<sth-port>/STH/v1/contextEntities/type/<entityType>/id/<entityId>

3. Removing all the data associated to certain attribute of certain entity, service and service path:

   http://<sth-host>:<sth-host>/STH/v1/contextEntities/type/<entityType>/id/<entityId>/attributes/<attrName>

The values between < and > should be substituted by their real values.

It is important to note that the data removal accomplished depends on the value of the SHOULD_STORE configuration parameter. This means that depending on this configuration option only the associated data (raw, aggregated or both) will be removed.
Migrating raw and aggregated time series context information

The STH component supports 3 alternative data models when storing the raw and aggregated time series context information into the database:

1. One collection per service path.
2. One collection per entity.
3. One collection per attribute.

As their names reflect, each one of the supported data models stores the raw and aggregated time series context information into one collection per service path, entity or attribute respectively.

The default data model is the collection per entity one, but sometimes to get the best performance out of the MongoDB database where the raw and aggregated time series context information is stored, alternative data models are needed. This is the reason why we introduced the data model migration tools.

To set the desired data model to be used, please take a look at Running the STH server below.

To run the data migration tool, please execute the following command:

```
./bin/sth_database_model
```

which will present the command help information:

```
Usage: sth_database_model [options]

Options:
-h, --help            output usage information
-V, --version        output the version number
-a, --analysis       prints the results of the data model analysis including the databases and collections which need to be migrated to the currently configured data model (mandatory if not -m or --migrate)
-m, --migrate        migrates to the currently configured data model all the databases and collections which has been created using a distinct data model (mandatory if not -a or --analysis)
-v, --verbose [documents] shows migration progress information if the number of documents to migrate in the collection is bigger or equal to the optional value passed (1 if no value passed)
-r, --remove-collection the original data model collection will be removed to avoid conflict if migrating back to that data model in the future
-u, --update-collection the migration will take place even if the target collections already exist combining the data of the ones to be migrated to the current data model
-f, --full            the migration will continue with the pending collections in case a previous collection throws an error and the migration operation is not idempotent for the aggregated data collections
-d, --database <databaseName> only this database will be taken into consideration for the analysis and/or migration process
-c, --collection <collectionName> only this collection will be taken into consideration, a database is mandatory if a collection is set
-x, --dictionary <dictionary> the path to a file including a dictionary to resolve the names of the collections to be analyzed and migrated (some of which may not apply and can be left as blank)
```

Special care should be taken when requesting a data model migration since the migration of aggregated time series context information is not an idempotent operation if the target data model collections already exist. In this case, the already existent data stored in these collections is combined with the one stored in the original data model collections pending migration. Based on this fact, we suggest the next procedure when considering a data model migration:
1. Check the current configured data model used by the STH component based on the options detailed in *Running the STH server* below. Typically it will be the default collection per entity data model.

2. Get a data model analysis report about the databases and collections which need to be migrated to the new desired data model (set in the `DATA_MODEL` environment variable) running the following command:

   ```
   LOGOPS_FORMAT=dev DATA_MODEL=collection-per-service-path ./bin/sth_database_model -a
   ```

3. Request the desired data model migration (set in the `DATA_MODEL` environment variable) without forcing the update of the target data model collections running the following command:

   ```
   LOGOPS_FORMAT=dev DATA_MODEL=collection-per-service-path ./bin/sth_database_model -a -m
   ```

   If any of the target collections already exist, the data model migration will stop and it will not be made for the first already existent target data model collection, either for any subsequent collection. If none of the target collections already exist, the data model migration will successfully complete.

   (a) If the data model migration completed successfully, remove the original data model collections already migrated to avoid problems if in the future you decide to go back to the original data model. Information about the successfully migrated collections is provided in the logs. Setting the `-r` option when running the command mentioned in point 3 will make this removal automatically for you. The data model migration has successfully finished.

   (b) If the data model migration did not complete successfully because any of the target data model collections already exist:

   i. Remove the original data model collections which were successfully migrated, if any, so they are not migrated again in the future (details about the successfully migrated collections is provided in the logs). The `-r` option will make this removal automatically for you when running the command mentioned in point 3.

   ii. You have to decide if the target data model collection causing the conflict contains valuable data. If they does, just keep it. If it does not, just remove it.

   iii. If you decided to keep the target data model collection causing the conflict since it contains valuable data, force its data model migration using the following command:

   ```
   LOGOPS_FORMAT=dev DATA_MODEL=collection-per-service-path ./bin/sth_database_model -a -m
   ```

   The original data model collection will be combined with the already existent data stored in the target data model collection.

   iv. Remove the `<original_data_model_collection_to_be_migrated>` collection whose migration you just forced so it is not migrated again in the future.

4. Get back and repeat from point 3.

Currently the only data model migration supported is the default collection per entity data model to the collection per service path data model.
Performance tests coverage

The Performance tests section of the repository includes information to run performance tests on the STH component. If you are interested on them, please navigate to that section of the repository for further information.
The Additional resources section of the repository includes some scripts and utilities which may make the developer’s life easier. If you are interested in them, please navigate to that section of the repository for further information.
Contribution guidelines

14.1 Overview

Being an open source project, everyone can contribute, provided that you follow the next guidelines:

- Before contributing any code, the author must make sure all the tests work (see below how to run the tests).
- Developed code must adhere to the syntax guidelines enforced by the linters.
- Code must be developed following the branching model and changelog policies defined below.
- For any new feature added, unit tests must be provided, following the example of the ones already created.

14.2 How to contribute

In order to start contributing:

1. Fork this repository clicking on the “Fork” button on the upper-right area of the page.
2. Clone your just forked repository:
   ```
   git clone https://github.com/your-github-username/fiware-sth-comet.git
   ```
3. Add the main fiware-sth-comet repository as a remote to your forked repository (use any name for your remote repository, it does not have to be fiware-sth-comet, although we will use it in the next steps):
   ```
   git remote add fiware-sth-comet https://github.com/telefonicaid/fiware-sth-comet.git
   ```

Before starting your contribution, remember to synchronize the develop branch in your forked repository with the develop branch in the main fiware-sth-comet repository following the next steps:

1. Change to your local develop branch (in case you are not in it already):
   ```
   git checkout develop
   ```
2. Fetch the remote changes:
   ```
   git fetch fiware-sth-comet
   ```
3. Merge them:
   ```
   git rebase fiware-sth-comet/develop
   ```

Contributions following these guidelines will be added to the develop branch, and released in the next version. The release process is explained in the Releasing section below.
14.3 Coding guidelines

Coding guidelines are defined via the provided .jshintrc and .gjslintrc flag files. The latter requires Python and its use can be disabled while creating the project skeleton with grunt-init. To check source code style, type:

```
grunt lint
```

Checkstyle reports can be used together with Jenkins to monitor project quality metrics by means of Checkstyle and Violations plugins. To generate Checkstyle and JSLint reports under report/lint/, type:

```
grunt lint-report
```

14.4 Branching model

There are two special branches in the repository:

- **master**: holds the code for the last stable version of the project. It is only updated when a new version is released.

- **develop**: contains the last stable development code. New features and bug fixes are always merged to develop.

In order to start developing a new feature or refactoring, a new branch should be created with name `task/<taskName>` in the newly forked repository. This new branch must be created from the current version of the `develop` branch (remember to fetch the latest changes from the remote `develop` branch before creating this new branch). Once the new functionality has been completed, a pull request should be created from the new branch to the `develop` branch in the main remote repository. Remember to check both the linters and the tests before creating the pull request.

Fixing bugs follow the same branching guidelines as in the case of adding a new feature or refactoring code with the exception of the branch name. In the case of bug fixes, the new branch should be called `bug/<bugName>`.

There are another set of branches called `release/<versionNumber>`, one for each version of the product. These branches point to each one of the released versions of the project. They are permanent and they are created with each release.

14.5 Changelog

The project contains a version changelog file, called `CHANGES_NEXT_RELEASE`, that can be found in the root of the project. Whenever a new feature or bug fix is going to be merged with `develop`, a new entry should be added to this changelog. The new entry should contain the reference number of the issue it is solving (if any).

When a new version is released, the changelog is cleared, and remains fixed in the last commit of that version. The content of the changelog is also moved to the release description in the Github release.

14.6 Testing

The test environment is preconfigured to run the Mocha Test Runner with support for the Chai assertion library as well as for Sinon spies, stubs, etc., following a BDD testing style with chai.expect and chai.should() available globally while executing tests, as well as the Sinon-Chai plugin.

Module mocking during testing can be done with proxyquire.
To run tests, type:

```bash
grun test
```

Tests reports can be used together with Jenkins to monitor project quality metrics by means of TAP or XUnit plugins. To generate TAP report in `report/test/unit_tests.tap`, type

```bash
grun test-report
```

### 14.7 Continuous testing

Support for continuous testing is provided so that tests are run when any source file or test is modified. For continuous testing, type:

```bash
grun watch
```

### 14.8 Code coverage

A very good practice is to measure the code coverage of your tests. To generate an HTML coverage report under the `site/coverage/` path and to print out a summary, type:

```bash
# Use git-bash on Windows
grun coverage
```

To generate a Cobertura report in `report/coverage/cobertura-coverage.xml` that can be used together with Jenkins to monitor project quality metrics by means of Cobertura plugin, type

```bash
# Use git-bash on Windows
grun coverage-report
```

### 14.9 Code complexity

Another very good practice is to analise code complexity. Support for using Plato and storing the generated report in the `site/report/` path is provided. This capability can be used together with Jenkins by means of DocLinks plugin. To generate a code complexity report, type:

```bash
grun complexity
```

### 14.10 Source code documentation

HTML code documentation can be generated under the `site/doc/` path. It can be used together with Jenkins by means of DocLinks plugin. For compiling source code documentation, type:

```bash
grun doc
```
14.11 Releasing

The process of making a release consists of the following steps and should be made by any of the owners or administrators of the main repository:

1. Create a new task branch changing the development version number in the package.json file (with a suffix -next) to the new target version (without any suffix), and create a pull request of this new task branch into develop. Remember to delete the temporary created task branch.

2. Create a release branch named release/<versionNumber> from the last version of develop using the corresponding version number.

3. Create a new release in Github setting the tag version as <versionNumber> from the new release branch release/<versionNumber> and publish it.

4. Create a pull request from the new release branch release/<versionNumber> to master.

5. Create a new task branch to prepare the develop branch for the next release, adding the -next suffix to the current version number in the package.json file (to signal this as the development version) and removing the contents of the CHANGES_NEXT_RELEASE changelog file. Create a pull request from this new task branch to develop. Remember to delete the temporary created task branch.

To further guide you through your first contributions, we have created the label mentored which are assigned to those bugs and issues simple and interesting enough to be solved by people new to the project. Feel free to assign any of them to yourself and do not hesitate to mention any of the main developers (this is, @gtorodelvalle or @frbattid) in the issue’s comments to get help from them during its resolution. They will be glad to help you.