# LA Generation Service - programmer manual

Politecnico di Torino

version 0.2.0 - 13 December 2013

SDSS

LA Generation Service

PoSecCo

http://www.posecco.eu

## Contents

## 1 Introduction

This document provides an overview of the *LA Generation Service* from the developer's point of view. It is considered a companion of the 'LA Generation Service - user manual' in which the use of the LA Generation Service and its UI are described.

The LA Generation Service is used to transform a set of policies into logical associations (LAs). This process is known as *policy refinement*. For further information regarding the policy refinement process and the LA generation see 'D3.5 - Models to refine the IT policy at service level'.

This service is a complex toolbox containing a number of specialized modules which are presented in depth in the following sections including their APIs, their dependencies and a how to extend them.

This document is structured as follows. The Section 2 is devoted to explain the LA Generation Service internal structure by providing a bird's eye view of its architecture, its plug-ins and its types. The Section 3 describes the tool APIs, focusing on the most important classes and interfaces. Finally, the Section 4 describes how to extend the tool by adding new components, features and UIs.

## 2 Software architecture

The goal of the LA Generation Service is to produce a set of logical associations via the policy refinement process which is graphically depicted in Figure 1.

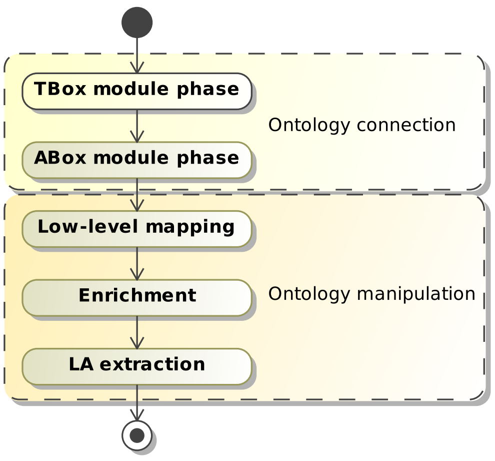

Figure 1: The policy refinement process.

The policy refinement is performed in two consecutive stages:

**Ontology connection** At the beginning, the LA Generation Service creates the PoSecCo ontology which contains all the information needed to produce the LAs. This stage is performed in two passes:

- **TBox module phase** A set of TBox ontologies[1] are merged together to form an initial PoSecCo ontology without any individual. For more information about the PoSecCo meta-models see 'D2.2 - IT policy meta-model and language' and 'D4.2 - Structural landscape meta-model'.
- **ABox module phase** The PoSecCo ontology is merged with a set of ABox ontologies[2], by filling it with the landscape and policy information.

**Ontology manipulation** When the initial PoSecCo ontology is created, the LA Generation Service analyzes it in order to extract the logical associations. This part is performed in three steps:

- **Low-level mapping** The IT level instances are mapped towards one or more low-level objects in the infrastructure layer.
- **Enrichment** A series of landscape inferences and deductions are performed in order to produce more secure and strict LAs in the following phase.
- **LA extraction** The data acquired in the previous two steps is analyzed in order to produce the logical associations which are store in the PoSecCo ontology.

The following paragraphs provides a briefly overview of the internal infrastructure of each aforementioned phase and their features.

### Implementation

The LA Generation Service is a tool entirely written using the Java programming language. In addition the technologies listed in Table 1 are extensively used in the project.

The implementation of the LA Generation Service refinement process, depicted in Figure 1, is showed in Figure 2 with the relevant class names.

[1]Roughly speaking, in PoSecCo, a *TBox ontology* is an ontology which does not contain any information about the landscape and it policies, that is it consist only of classes, properties and no proper individuals.

[2]Roughly speaking, in PoSecCo, an *ABox ontology* is an ontology which contains some information about the landscape or the policies.

| Name | Website of the project |
|---|---|
| Eclipse plug-in framework | http://eclipse.org/ |
| Remote Application Platform toolkit | http://eclipse.org/rap/ |
| Zest visualization toolkit | http://www.eclipse.org/gef/zest/ |
| OWL API ontology library | http://owlapi.sourceforge.net/ |
| Pellet reasoner | http://clarkparsia.com/pellet/ |
| Hermit reasoner | http://hermit-reasoner.com/ |
| SPARQL-DL query engine | http://www.derivo.de/en/resources/sparql-dl-api.html |
| JGraphT graph library | http://jgrapht.org/ |

Table 1: The technologies used in the LA Generation Service.

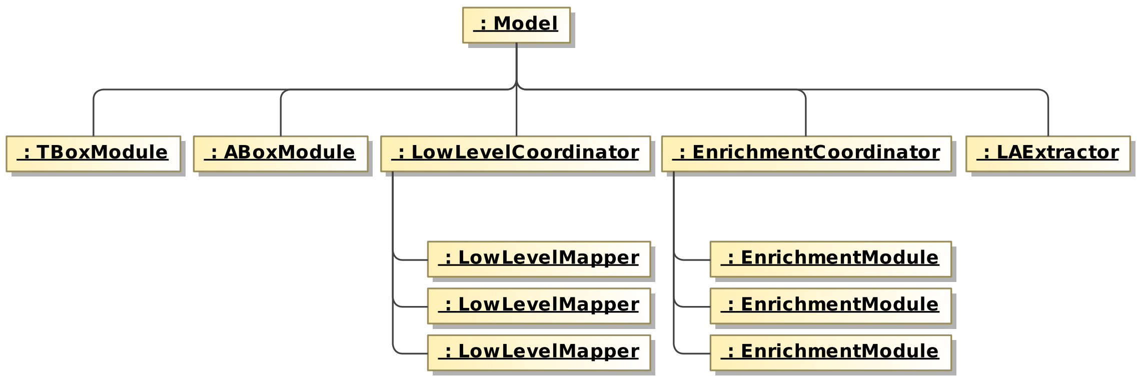

Figure 2: The core implementation schema.

Each aforementioned phase is implemented by an instance of a class acting as a coordinator for several other sub-components, that is:

- the `TBoxModule` class is the coordinator of the TBox module phase;
- the `ABoxModule` class is the coordinator of the ABox module phase;
- the `LowLevelCoordinator` class is the coordinator of the low-level mapping phase;
- the `EnrichmentCoordinator` class is the coordinator of the enrichment phase;
- the `LAExtractor` class is the coordinator of the LA extraction phase.

The entire project obeys the rules dictated by the MVC (Model-View-Controller) pattern. This is emphasized by the fact that the coordinator of each phase makes use of a central model, represented by the homonym class `Model` (which contains the PoSecCo ontology and several other data). A detailed description of these classes is given in Section 3.

In order to promote the flexibility and extensibility of the LA Generation Service, the low-level mapping and enrichment coordinators make use of a plug-in based approach. They offer the capability of adding new modules (i.e., low-level mappers and enrichment modules) by the means of an Eclipse extension point. By implementing two special abstract classes (`LowLevelMapper` and `EnrichmentModule`) the developer can extend the functionalities of the LA Generation Service without modifying the implementation source code. This topic is discussed further in Section 4.

## Plug-ins

Since the Eclipse framework was adopted, all the LA Generation Service code is split into a set of specialized plug-ins, that are:

eu.posecco.sdss.lageneration.tbox
: includes the `TBoxModule` class and all the sources needed to implement the TBox module phase

eu.posecco.sdss.lageneration.abox
: includes the `ABoxModule` class and all the sources needed to implement the ABox module phase

eu.posecco.sdss.lageneration.lowlevel
: includes the `LowLevelCoordinator`, the `LowLevelMapper` classes and all the sources needed to implement the low-level mapping phase but the low-level mappers

eu.posecco.sdss.lageneration.lowlevel.modules
: includes the standard low-level mappers

eu.posecco.sdss.lageneration.lowlevel.ui
: includes the UI for the low-level mapping phase

eu.posecco.sdss.lageneration.enrichment
: includes the `EnrichmentCoordinator`, the `EnrichmentModule` classes and all the sources needed to implement the enrichment phase but the enrichment modules

eu.posecco.sdss.lageneration.enrichment.modules
: includes the standard enrichment modules

eu.posecco.sdss.lageneration.enrichment.ui
: includes the UI for the enrichment phase

eu.posecco.sdss.lageneration.laextraction
: includes the `LAExtractor` class and all the sources needed to implement the LA extraction phase

eu.posecco.sdss.lageneration.laextraction.ui
: includes the UI for LA extraction phase

eu.posecco.sdss.lageneration.ui
: includes the perspective, the views and all the related sources

eu.posecco.sdss.lageneration.model
: includes the `Model` class and all its related sub-types

These plug-ins are extensively intertwined amongst them as shown in the dependency graph depicted in Figure 3.

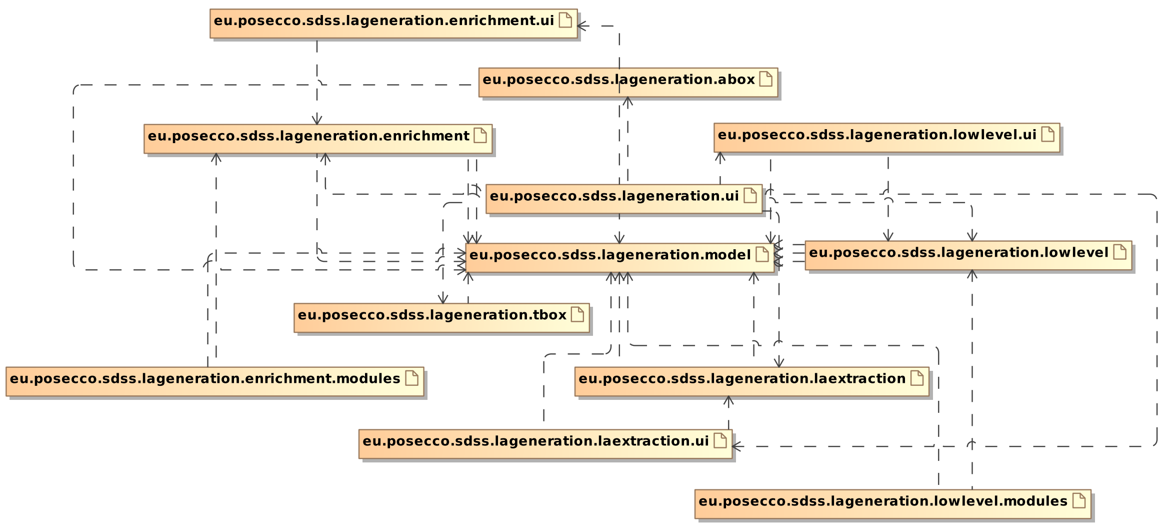
eu.posecco.sdss.lageneration.enrichment.ui
eu.posecco.sdss.lageneration.abox
eu.posecco.sdss.lageneration.enrichment
eu.posecco.sdss.lageneration.lowlevel.ui
eu.posecco.sdss.lageneration.ui
eu.posecco.sdss.lageneration.model
eu.posecco.sdss.lageneration.lowlevel
eu.posecco.sdss.lageneration.tbox
eu.posecco.sdss.lageneration.enrichment.modules
eu.posecco.sdss.lageneration.laextraction
eu.posecco.sdss.lageneration.laextraction.ui
eu.posecco.sdss.lageneration.lowlevel.modules

Figure 3: The LA Generation Service plug-ins dependencies.

In addition, these plug-ins make use of several classes, enumeration and types contained in a number of other SDSS-wide bundles:

eu.posecco.sdss.common
: includes several facilities used to pass data (e.g., the PoSecCo ontology) to the other SDSS components such as the Infrastructure Configuration Service.

eu.posecco.sdss.images
: includes all the icons and images

eu.posecco.sdss.io
: includes several MoVE utility classes

eu.posecco.sdss.libraries
: includes all the external libraries needed by the LA Generation Service

eu.posecco.sdss.ontologies
: includes all the classes and types needed to manipulate the ontologies

eu.posecco.sdss.ui
: includes a number of UI-related classes

eu.posecco.sdss.util
: includes a number of UI-independent utility classes.

The dependencies amongst these other plug-ins is depicted in Figure 4.

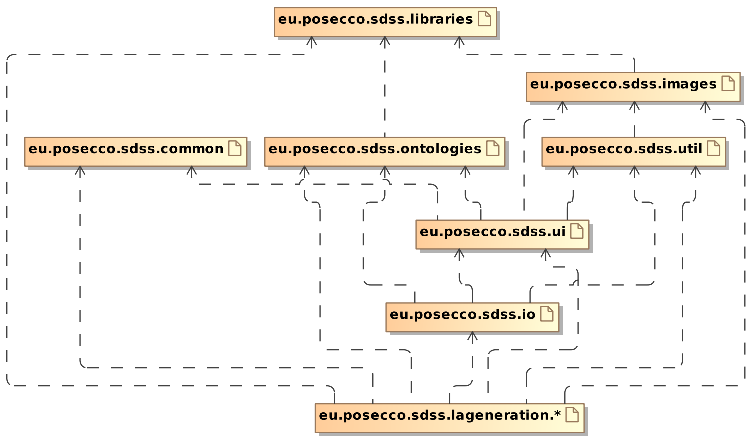

Figure 4: The SDSS-wide and LA Generation Service plug-ins dependencies.

## Metrics

Source code metrics are an effective way to intuitively understand the size and complexity of a piece of software. For instance, the Table 2 shows a series of code statistics related to the LA Generation Service.

In addition the Tables 3 and 4 lists a set of metrics related to the PoSecCo ontology immediately after the ABox phase (that is the ontology still without the LAs).

| Metric | Value |
|---|---|
| Number of plug-ins | 19 |
| Number of packages | 115 |
| Number of classes | 267 |
| Number of methods | 732 |
| Number of lines | 20116 |
| McCabe cyclomatic complexity | 1.93 |

Table 2: The source code metrics.

| Metric | Value |
|---|---|
| Number of axioms | 5643 |
| Number of classes | 333 |
| Number of object properties | 204 |
| Number of data properties | 76 |
| Number of individuals | 767 |
| DL expressivity | $\mathcal{SOIF}(\mathcal{D})$ |

Table 3: The PoSecCo ontology metrics after the ABox phase for the year 2 landscape (Thales).

| Metric | Value |
|---|---|
| Number of axioms | 5621 |
| Number of classes | 325 |
| Number of object properties | 259 |
| Number of data properties | 110 |
| Number of individuals | 859 |
| DL expressivity | $\mathcal{SIF}(\mathcal{D})$ |

Table 4: The PoSecCo ontology metrics after the ABox phase for the year 3 landscape (ATOS).

# 3 Public APIs

This section briefly describes the public methods of the most important classes of the LA Generation Service. For a more detailed description of all the available types please refer to the code Javadoc documentation.

## The model

The LA Generation Service model is represented by the homonym class `Model`. This class is contained in the eu.posecco.sdss.lageneration.model plug-in and its public methods are depicted in Figure 5.

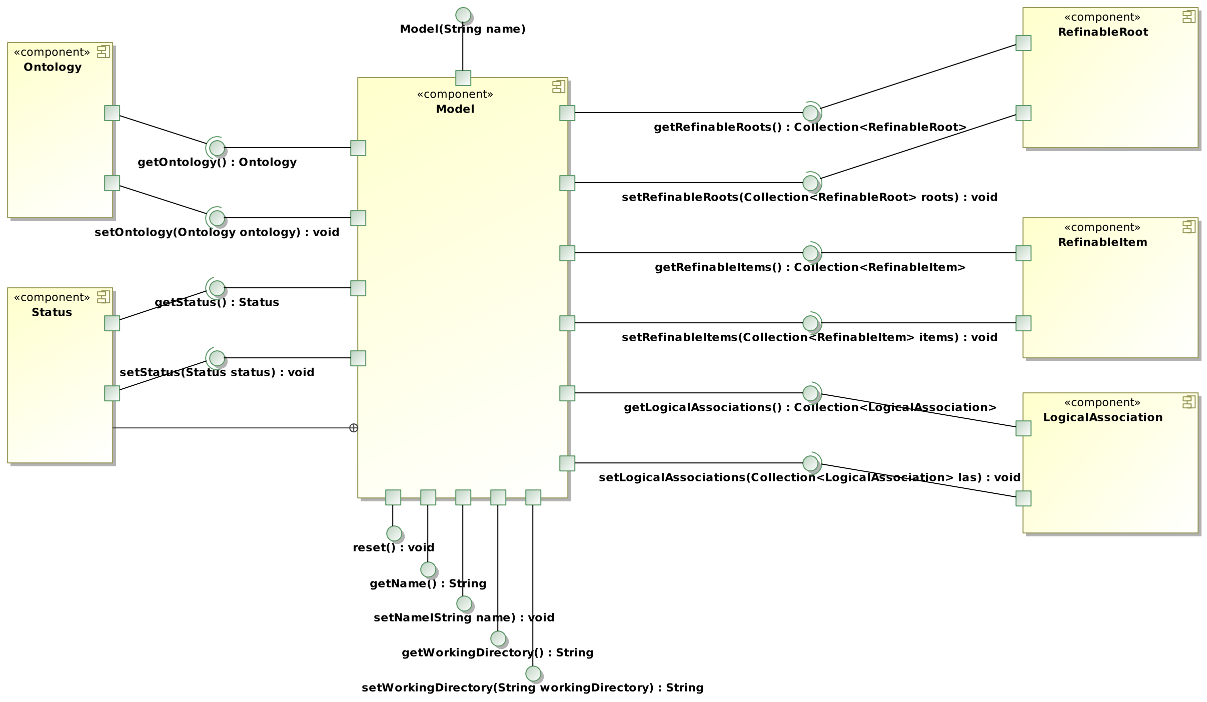
«component» Ontology
«component» Status
«component» Model
«component» RefinableRoot
«component» RefinableItem
«component» LogicalAssociation
Model(String name)
getOntology() : Ontology
setOntology(Ontology ontology) : void
getStatus() : Status
setStatus(Status status) : void
getRefinableRoots() : Collection<RefinableRoot>
setRefinableRoots(Collection<RefinableRoot> roots) : void
getRefinableItems() : Collection<RefinableItem>
setRefinableItems(Collection<RefinableItem> items) : void
getLogicalAssociations() : Collection<LogicalAssociation>
setLogicalAssociations(Collection<LogicalAssociation> las) : void
reset() : void
getName() : String
setNamelString name) : void
getWorkingDirectory() : String
setWorkingDirectory(String workingDirectory) : String

Figure 5: The `Model` class API.

This class uses several additional classes and enumerations to store the current LA Generation Service model. For instance, the following types are frequently encountered in the code:

- the `Ontology` class is used to store the PoSecCo ontology and provides a set of methods for reading and manipulating its content[3];
- the `Status` specifies the current phase of the refinement process (e.g., enrichment, LA extraction, . . . ).

The `Model` class exposes the following public methods:

**`Model`**`(String name)`
    creates a new empty model with the specified name

`Collection<LogicalAssociation>` **`getLogicalAssociations`**`()`
    retrieves the refined logical associations

`String` **`getName`**`()`
    retrieves the model name

`Ontology` **`getOntology`**`()`
    retrieves the current PoSecCo ontology

[3]This class is contained in the eu.posecco.sdss.ontologies plug-in.

`Collection<RefinableItem>` **`getRefinableItems`**`()`
: retrieves the items (policies and links) selected for the refinement

`Collection<RefinableRoot>` **`getRefinableRoots`**`()`
: retrieves the items (policies and links) that can be refined

`Status` **`getStatus`**`()`
: retrieves the current refinement phase

`String` **`getWorkingDirectory`**`()`
: retrieves the working directory, that is where the output files will be written

`void` **`reset`**`()`
: resets the model to its initial empty state

`void` **`setLogicalAssociations`**`(Collection<LogicalAssociation> las)`
: sets the refined logical associations

`void` **`setName`**`(String name)`
: sets the model name

`void` **`setOntology`**`(Ontology ontology)`
: sets the current PoSecCo ontology

`void` **`setRefinableItems`**`(Collection<RefinableItem> items)`
: sets the items (policies and links) selected for the refinement

`void` **`setRefinableRoots`**`(Collection<RefinableRoot> roots)`
: sets the items (policies and links) that can be refined

`void` **`setStatus`**`(Status status)`
: sets the current refinement phase

`void` **`setWorkingDirectory`**`(String workingDirectory)`
: sets the working directory

Note that a valid `Model` instance *must* be passed to the phase coordinators in order to obtain a correct and consistent output.

## TBox module

The TBox module coordinator is represented by the class `TBoxModule`. This class is contained in the eu.posecco.sdss.lageneration.tbox plug-in and its public methods are depicted in Figure 6.

This class makes use of the enumeration `ReasonerType` which contains a list of all the supported ontology reasoners[4].

The `TBoxModule` class exposes the following public methods:

**`TBoxModule`**`(Model model)`
: creates the module using the specified model

`void` **`merge`**`(ReasonerType reasonerType)`
: creates the initial PoSecCo TBox ontology, that is an ontology without the landscape and policy information, and initializes the reasoner

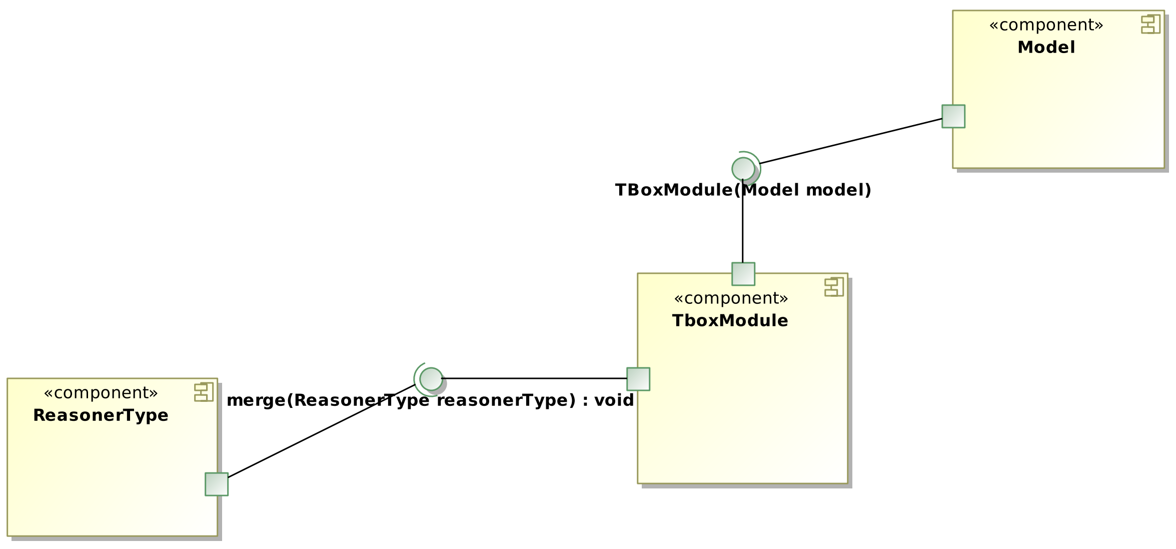

Figure 6: The `TBoxModule` class API.

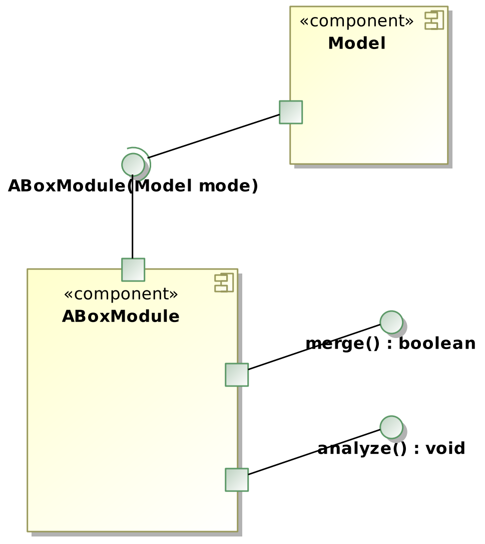

Figure 7: The `ABoxModule` class API.

## ABox module

The ABox module coordinator is represented by the class `ABoxModule`. This class is contained in the eu.posecco.sdss.lageneration.abox plug-in and its public methods are depicted in Figure 7.

The `ABoxModule` class exposes the followin public methods:

**`ABoxModule`**`(Model model)`
: creates the module using the specified model

`void` **`analyze`**`()`
: analyzes the ontology in order to detect the available refinable items

`boolean` **`merge`**`()`
: fills the PoSecCo ontology with the landscape and policy information, reading them from a local copy (and returning false) or MoVE (and returning true)

[4]This enumeration is declared in the eu.posecco.sdss.ontologies plug-in and can be used to select the Pellet or the Hermit reasoners.

## Low-level mapper

The low-level mapper coordinator is represented by the class `LowLevelCoordinator`. This class is contained in the eu.posecco.sdss.lageneration.lowlevel plug-in and its public methods are depicted in Figure 8.

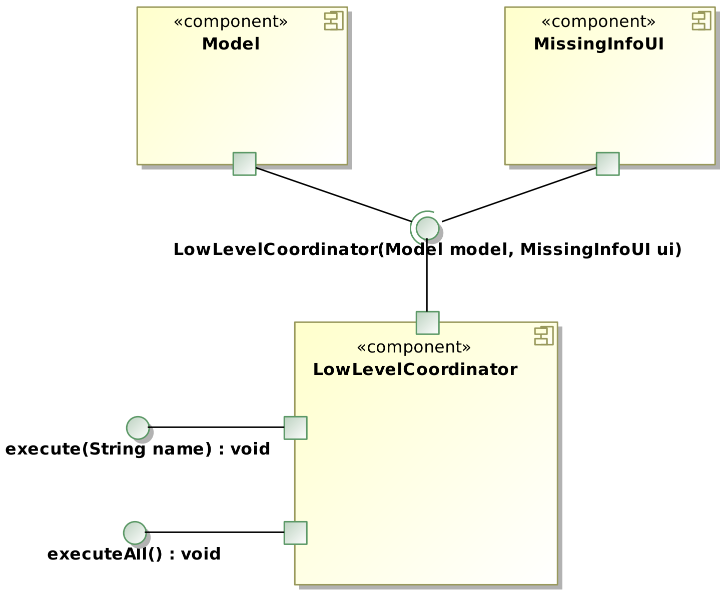

Figure 8: The `LowLevelCoordinator` class API.

This class uses a number of *low-level mapper*s, that are modules specialized in finding the relationships of an IT level instance toward a specific infrastructure level concept. The creation of new low-level mappers is discussed in Section 4. The LA Generation Service already contains the following default mappers (declared in the eu.posecco.sdss.lageneration.lowlevel.modules plug-in):

- the mapper toward the IT interfaces;
- the mapper toward the interfaces;
- the mapper toward the IT resources;
- the mapper toward the nodes.

The `LowLevelCoordinator` class exposes the following public methods:

**`LowLevelCoordinator`**`(Model model, MissingInfoUI ui)`
creates the coordinator

`void` **`execute`**`(String name)`
executes the low-level mapper called `name`

`void` **`executeAll`**`()`
executes all the low-level mappers

This class uses the interface `MissingInfoUI` which represents a UI which should ask the user for a missing relationship between two landscape elements (e.g., which are the computers used by a specific user). Its public methods are depicted in Figure 9.
The `MissingInfoUI` interface exposes only one method:

`void` **`open`**`(Model model, Set<ObjectPropertyAssertion> info)`
this method is called by the low-level coordinator whenever a set of missing relationships is found and its job is to fill the missing information in the parameter `info`

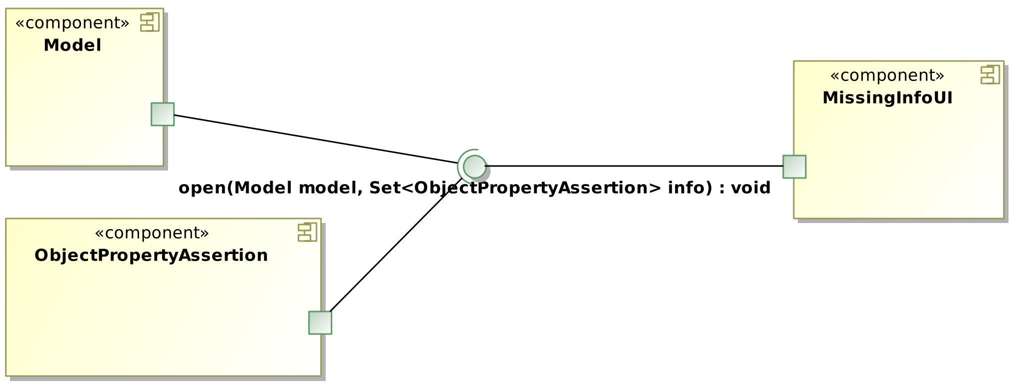

Figure 9: The `MissingInfoUI` class API.

The `ObjectPropertyAssertion` class represents a missing relationship between two individuals in the PoSecCo ontology. Its public methods are depicted in Figure 10.

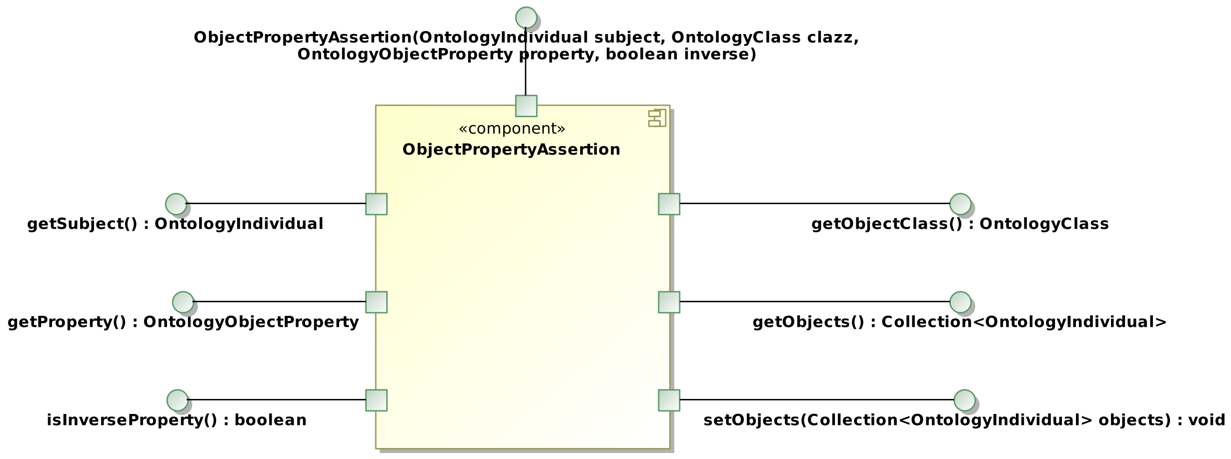
ObjectPropertyAssertion(OntologyIndividual subject, OntologyClass clazz, OntologyObjectProperty property, boolean inverse)
«component»
ObjectPropertyAssertion
getSubject() : OntologyIndividual
getObjectClass() : OntologyClass
getProperty() : OntologyObjectProperty
getObjects() : Collection<OntologyIndividual>
isInverseProperty() : boolean
setObjects(Collection<OntologyIndividual> objects) : void

Figure 10: The `ObjectPropertyAssertion` class API.

This class exposes the following public methods:

**`ObjectPropertyAssertion`**`(OntologyIndividual subject, OntologyClass clazz, OntologyObjectProperty property, boolean inverse)`
: creates the missing relationship assertion

`boolean` **`isInverseProperty`**`()`
: indicates if the assertion regards the property or its inverse

`OntologyClass` **`getObjectClass`**`()`
: retrieves the object class of the assertion

`Collection<OntologyIndividual>` **`getObjects`**`()`
: retrieves the objects of the assertion or `null` if they are missing

`OntologyObjectProperty` **`getProperty`**`()`
: retrieves the object property of the assertion

`OntologyIndividual` **`getSubject`**`()`
: retrieves the subject of the assertion

`void` **`setObjects`**`(Collection<OntologyIndividual>)`
: sets the objects of the assertion

### Enrichment

The enrichment coordinator is represented by the class `EnrichmentCoordinator`. This class is contained in the eu.posecco.sdss.lageneration.enrichment plug-in and its public methods are depicted in Figure 11.

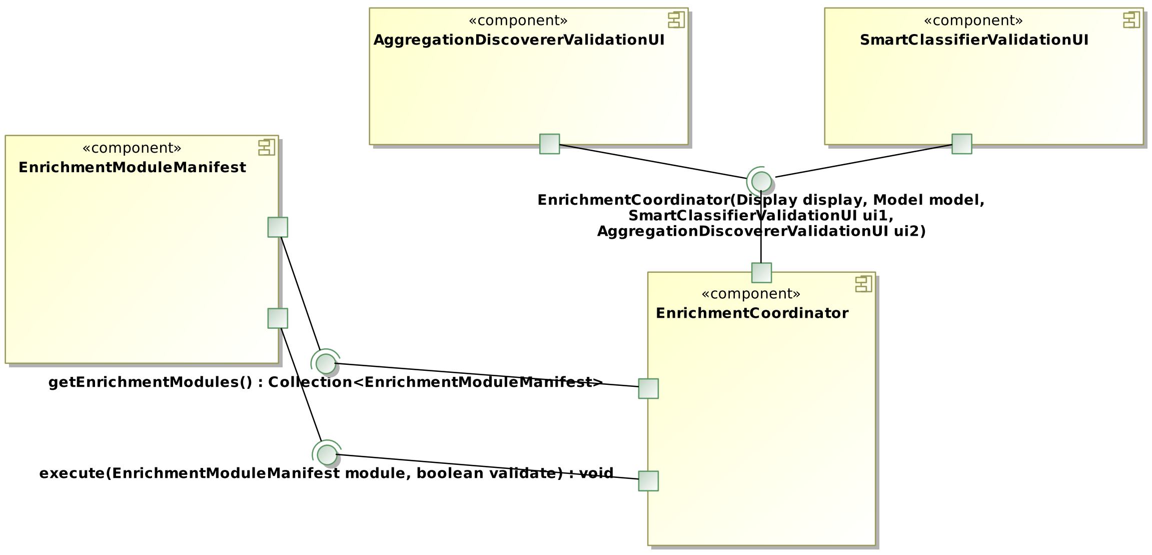
«component»
AggregationDiscovererValidationUI
«component»
SmartClassifierValidationUI
«component»
EnrichmentModuleManifest
EnrichmentCoordinator(Display display, Model model, SmartClassifierValidationUI ui1, AggregationDiscovererValidationUI ui2)
«component»
EnrichmentCoordinator
getEnrichmentModules() : Collection<EnrichmentModuleManifest>
execute(EnrichmentModuleManifest module, boolean validate) : void

Figure 11: The `EnrichmentCoordinator` class API.

This class uses a set of *enrichment modules*, that are bundles used to perform a number of inferences and deductions on the landscape contained in the PoSecCo ontology. Their creation is discussed in detail in Section 4. All the default enrichment modules are declared in the eu.posecco.sdss.lageneration.enrichment.modules plug-in and are:

- the *data model through SSH* aggregation discoverer, which infer if a connection toward a data model should be protected by using SSH;
- the *filtering zones* aggregation discoverer, which detect the filtering zones[5];
- the *unused objects* smart classifier, which find the used and unused IT objects;
- the *tunnelling zones* aggregation discoverer, which detect the tunnelling zones[6];
- the *WS-Security links classifier* smart classifier, which infer if a link should be protected using the WS-Security technology.

The `EnrichmentCoordinator` class exposes the following public methods:

**`EnrichmentCoordinator`**`(Display display, Model model, SmartClassifierValidationUI ui1, AggregationDiscovererValidationUI ui2)`
    creates the coordinator

`void` **`execute`**`(EnrichmentModuleManifest module, boolean validate)`
    executes an enrichment module and open the validation UI if requested

`Collection<EnrichmentModuleManifest>` **`getEnrichmentModules`**`()`
    retrieves all the available enrichment modules

The enrichment coordinator class make use of the two interfaces `SmartClassifierValidationUI` and `AggregationDiscovererValidationUI` which represent two UIs which should report to the user the results of an enrichment module and give him the possibility to abort the enrichment module deductions.

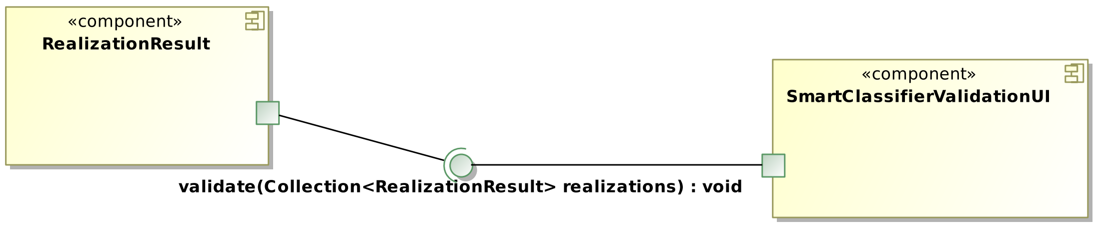
«component»
RealizationResult
«component»
SmartClassifierValidationUI
validate(Collection<RealizationResult> realizations) : void

Figure 12: The `SmartClassifierValidationUI` class API.

The public methods of `SmartClassifierValidationUI` are depicted in Figure 12.

The `SmartClassifierValidationUI` interface exposes only one method:

`void` **`validate`**`(Collection<RealizationResult> realizations)`
: this method is called by the enrichment coordinator if the validation UI for an enrichment module is requested and its job is to allow the user to edit the content of the collection `realizations` that is the result of the enrichment

The `RealizationResult` class represents a suggested realization (classification) of an ontology individual by an enrichment module. Its public methods are depicted in Figure 13.

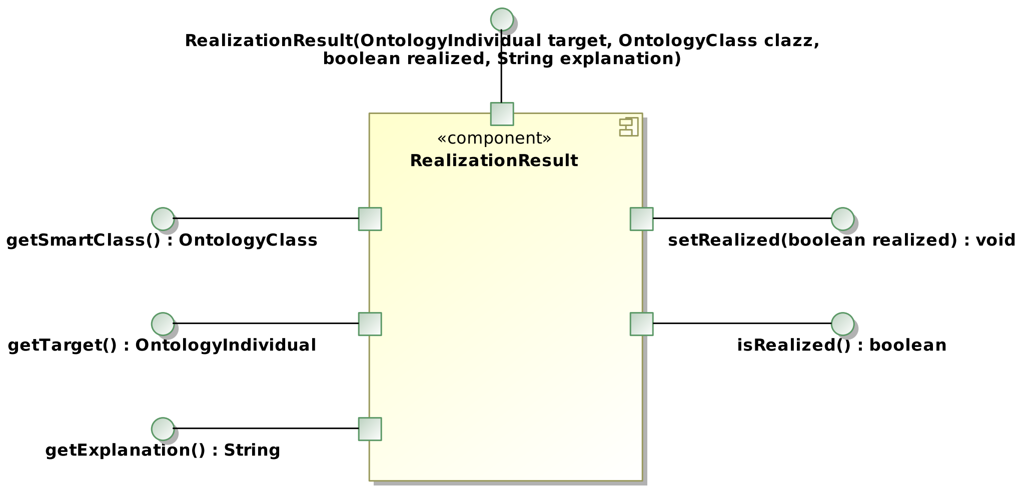
RealizationResult(OntologyIndividual target, OntologyClass clazz, boolean realized, String explanation)
«component»
RealizationResult
getSmartClass() : OntologyClass
getTarget() : OntologyIndividual
getExplanation() : String
setRealized(boolean realized) : void
isRealized() : boolean

Figure 13: The `RealizationResult` class API.

The `RealizationResult` class exposes the following public methods:

**`RealizationResult`**`(OntologyIndividual target, OntologyClass clazz,`
`boolean realized, String explanation)`
: creates the realization suggesting that the individual `target` should belong to the class `clazz`

`String` **`getExplanation`**`()`
: retrieves the explanation of the realization

`OntologyClass` **`getSmartClass`**`()`
: retrieves the target class of the realization

`OntologyIndividual` **`getTarget`**`()`
: retrieves the target of the realization

[5]A *filtering zone* is a set of network nodes that can communicate together without crossing any filtering device.
[6]A *tunnelling zone* is a set of network nodes that lies at the end of a single IPsec tunnel.

`boolean` **`isRealized`**`()`
: indicates if the realization is positive (the individual should belong to the class) or negative (the individual should not belong to the class)

`void` **`setRealized`**`(boolean realized)`
: sets the realization sign

The public methods of `AggregationDiscovererValidationUI` are depicted in Figure 14.

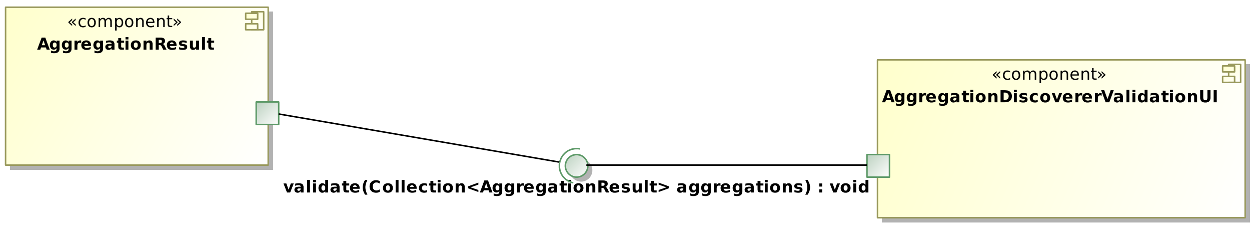
«component»
AggregationResult
«component»
AggregationDiscovererValidationUI
validate(Collection<AggregationResult> aggregations) : void

Figure 14: The `AggregationDiscovererValidationUI` class API.

The `AggregationDiscovererValidationUI` interface exposes only one method:

`void` **`validate`**`(Collection<AggregationResult> aggregations)`
: this method is called by the enrichment coordinator if the validation UI for an enrichment module is requested and its job is to allow the user to edit the content of the collection `aggregations` that is the result of the enrichment

The `AggregationResult` class represents a suggested aggregation (set of individuals) by an enrichment module. Its public methods are depicted in Figure 15.

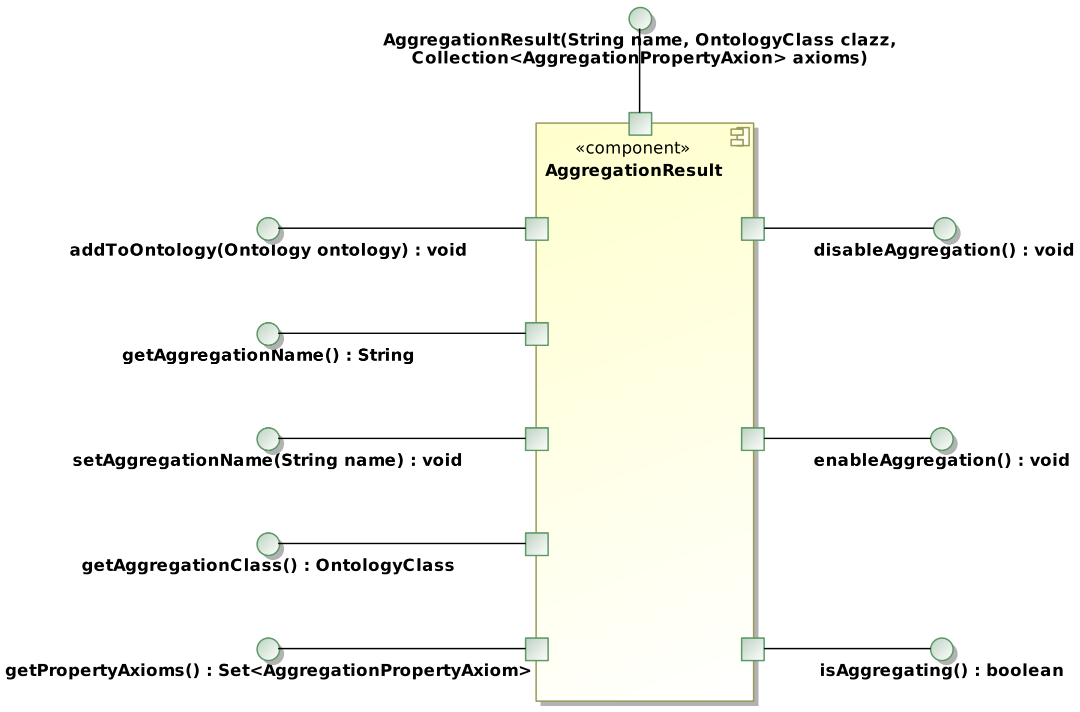
AggregationResult(String name, OntologyClass clazz, Collection<AggregationPropertyAxion> axioms)
«component»
AggregationResult
addToOntology(Ontology ontology) : void
getAggregationName() : String
setAggregationName(String name) : void
getAggregationClass() : OntologyClass
getPropertyAxioms() : Set<AggregationPropertyAxiom>
disableAggregation() : void
enableAggregation() : void
isAggregating() : boolean

Figure 15: The `RealizationResult` class API.

The `AggregationResult` class exposes the following public methods:

**`AggregationResult`**`(String name, OntologyClass clazz,`
`Collection<AggregationPropertyAxiom> axioms)`
: creates the aggregation named `name` in the class `clazz` having the properties `axioms`

`void` **`addToOntology`**`(Ontology ontology)`
: adds the current result to an ontology

`void` **`disableAggregation`**`()`
: disables this result

`void` **`enableAggregation`**`()`
: enables this result

`boolean` **`isAggregating`**`()`
: detects if this result refers to an aggregating or aggregated individual[7]

`OntologyClass` **`getAggregationClass`**`()`
: retrieves the aggregation class

`String` **`getAggregationName`**`()`
: retrieves the aggregation name

`Collection<AggregationPropertyAxiom>` **`getPropertyAxioms`**`()`
: retrieves the aggregation properties

`void` **`setAggregationName`**`(String name)`
: sets the aggregation name

In addition, the `EnrichmentCoordinator` uses the `EnrichmentModuleManifest` class which represent all the information about a specific enrichment module.Its public methods are depicted in Figure 16.

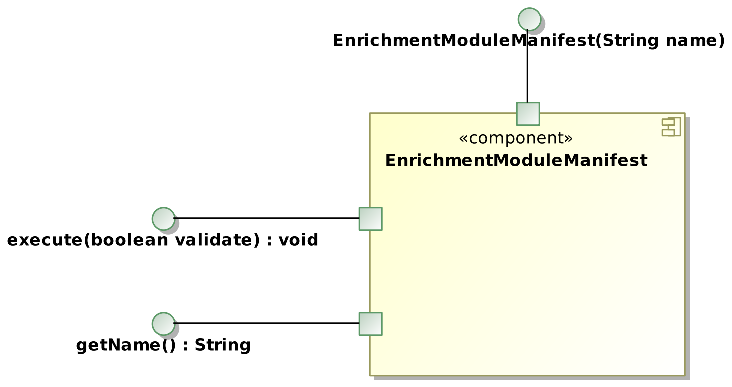
EnrichmentModuleManifest(String name)
«component»
EnrichmentModuleManifest
execute(boolean validate) : void
getName() : String

Figure 16: The `EnrichmentModuleManifest` class API.

The `EnrichmentModuleManifest` class exposes the following methods:

**`EnrichmentModuleManifest`**`(String name)`
: creates an enrichment module manifest for the module called `name`

`String` **`getName`**`()`
: retrieves the name of the enrichment module

`void` **`execute`**`(boolean validate)`
: executes the enrichment module and open the validation UI if requested

## The LA extraction

The LA extraction coordinator is represented by the class `LAExtractor`. This class is contained in the eu.posecco.sdss.lageneration.laextraction plug-in and its public methods are depicted in Figure 17.
It exposes the following public methods:

[7] An *aggregating* individual is an object that logically contains a set of *aggregated* individuals.

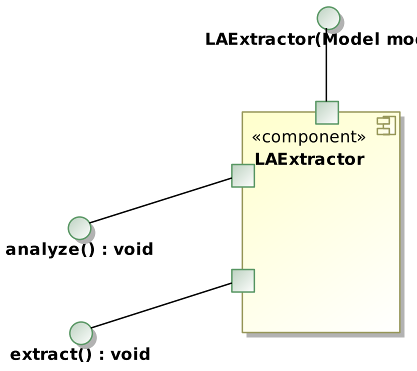

Figure 17: The `LAExtractor` class API.

**`LAExtractor`**`(Model model)`
: creates a new LA extractor for the specified model

`void` **`analyze`**`()`
: analyzes the ontology in order to detect the technologies and properties for generating the LAs

`void` **`extract`**`()`
: extracts the LAs

# 4 Extending the tool

This section describes how extend the LA Generation Service functionalities by specifying the classes, extension points and files involved in the process.

## Developing a new UI

The LA Generation Service modular system allows a developer to easily deploy it into a new user interface since there is a marked division between the UI and the core (UI-independent) plug-ins.
The core plug-ins are:

- the eu.posecco.sdss.lageneration.tbox;
- the eu.posecco.sdss.lageneration.abox;
- the eu.posecco.sdss.lageneration.lowlevel;
- the eu.posecco.sdss.lageneration.lowlevel.modules;
- the eu.posecco.sdss.lageneration.enrichment;
- the eu.posecco.sdss.lageneration.enrichment.modules;
- the eu.posecco.sdss.lageneration.laextraction;
- the eu.posecco.sdss.lageneration.model.

While the following ones contains the default UI:

- the eu.posecco.sdss.lageneration.lowlevel.ui;
- the eu.posecco.sdss.lageneration.enrichment.ui;
- the eu.posecco.sdss.lageneration.laextraction.ui;
- the eu.posecco.sdss.lageneration.ui.

The only requirement is that the coordinators *must* be called in the right order and that the `MissingInfoUI`, `SmartClassifierValidationUI` and `AggregationDiscovererValidationUI` interfaces for the low-level mapping and enrichment *must* be implemented by a concrete class calling the right user interface as needed.
This architecture allows the developer to easily integrate the LA Generation Service into its own system by using a fully customized GUI, CUI or even by removing any UI in order to perform the policy refinement in a totally automatic way.

## Adding a new phase in the refinement

In the following paragraphs it is described how to add a new phase in the refinement process using the default UI. The process can be substantially different if the developer uses a custom user interface.
There are several choices available to add a new job, but the simplest and easiest consists of the following steps:

**Create a new status item** Create a new item in the `Status` enumeration that will identify the new stage. The `Status` enumeration is located in the eu.posecco.sdss.lageneration.model plug-in.

**Create a new Eclipse job** Create a new Eclipse job or use the `Phase`[8] class which contains the code to be executed. The developer is encouraged to use one of these classes so that the new code can be launched in a separated thread in order to avoid freezing the UI during its execution. The only requirement is that the job *must* use the `setStatus()` method on a `Model` instance to signal the end of the phase.

**Modify the automatic process** Modify the view defined in the `LAGenerationView` class, located in the eu.posecco.sdss.lageneration.ui plug-in, by editing the selection listener of the `doAllButton` field. This is needed in order to execute the new phase when the user choose the automatic process option.

**Modify the manual process** Add a new control (e.g., a new hyperlink) and a selection listener in the class `LAGenerationView` that will execute the new phase. The `initDataBindings()` method should also be modified to enable or disable the new control accordingly to the model status. This is needed in order to execute the new phase when the user choose the manual process option.

To accomplish these modifications the developer needs at least a basic understanding of the Eclipse framework and the SWT/RWT libraries.

## Adding new low-level mappers

A low-level mapper is a special module that maps an IT level object to its infrastructure layer counterpart. In order to add a new low-level mapper the developer must perform two tasks:

- adding a new extension to the extension point eu.posecco.sdss.lageneration.lowlevel.modules;
- create a new sub-class of the class `LowLevelMapper` that will contains the module code.

Once performed these steps, the module is automatically recognized and registered in the LA Generation Service without touching its internal source code. When the tool is launched, the new mapper is executed together with the default ones.

### The eu.posecco.sdss.lageneration.lowlevel.modules extension point

The eu.posecco.sdss.lageneration.lowlevel.modules extension point exhibits the structure graphically depicted in Figure 18.

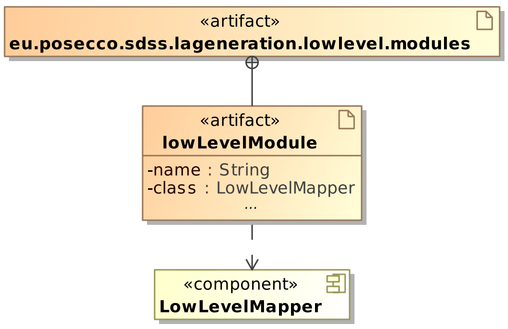

Figure 18: The eu.posecco.sdss.lageneration.lowlevel.modules extension point attributes.

This extension point defines the element lowLevelModule which represent a low-level mapper by exposing the following attributes:

**`name`** : String
the name of the low-level mapper

**`class`** : LowLevelMapper
the class implementing the low-level mapper

[8]This class is a custom extension of `org.eclipse.core.runtime.jobs.Jobs` and it is contained in the eu.posecco.sdss.util plug-in.

### The `LowLevelMapper` class

A low-level mapper is represented by the abstract class `LowLevelMapper`. This class is contained in the eu.posecco.sdss.lageneration.lowlevel plug-in and its public methods are depicted in Figure 19.

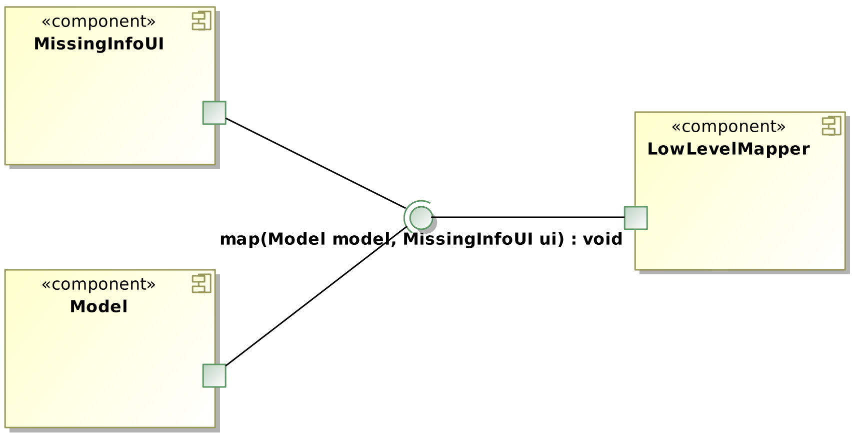

Figure 19: The `LowLevelMapper` class API.

The `LowLevelMapper` class exposes only one public method:

`void` **`map`**`(Model model, MissingInfoUI ui)`
performs the mapping and display the user interface `ui` requesting a missing relationship, if needed

The `map()` method should analyze the ontology and insert a series of property assertions between an IT level object and an infrastructure level individual. Typically the property used is a child of `refinestTo`, such as `refinesToNode` or `refinesToITInterface`. If a new object property needs to be added to the ontology, please refer to the Section 4.

## Adding new enrichment modules

An enrichment module is a special bundle that performs some inferences over the landscape elements. These deductions will be used in the later stages of the refinement process to generate more secure logical associations. In order to add a new enrichment module the developer must perform two tasks:

- adding a new extension to the extension point eu.posecco.sdss.lageneration.enrichment.modules;
- create a new sub-class of the class `EnrichmentModule` that will contains the module code.

Once performed these steps, the module is automatically recognized and registered in the LA Generation Service. When the tool is launched, the new bundle should appear in the UI listing the available enrichment modules.
The enrichment modules can be split in two different types:

- the *smart classifiers* try to classify a landscape element in a dynamically generated class, called a *smart class* (e.g., classification of a service as a public service);
- the *aggregation discoverers* find *aggregations* of landscape elements, that are set of individuals sharing some common feature (e.g., detecting the filtering zones).

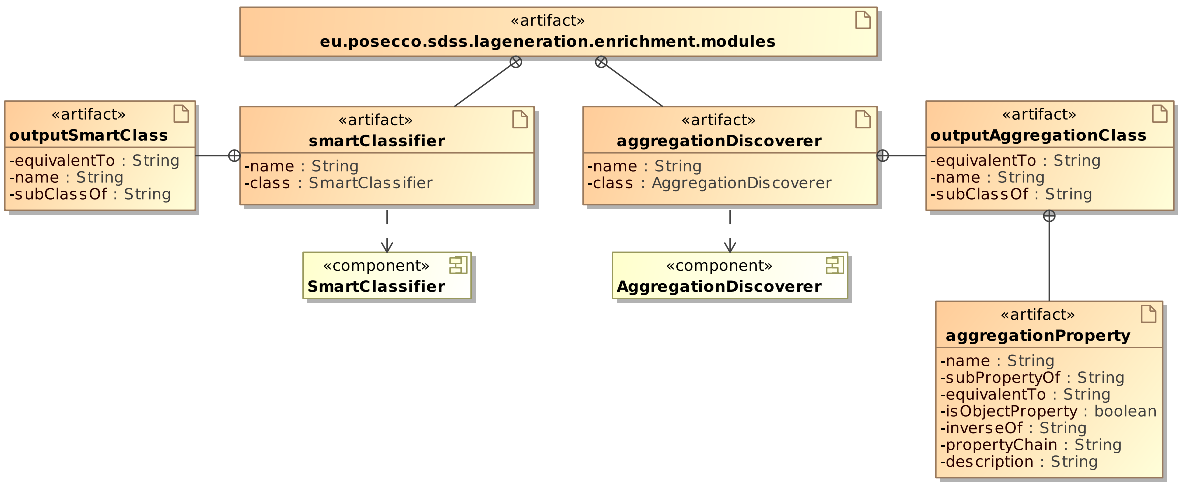

Figure 20: The eu.posecco.sdss.lageneration.enrichment.modules extension point attributes.

### The eu.posecco.sdss.lageneration.enrichment.modules extension point

The eu.posecco.sdss.lageneration.enrichment.modules extension point exhibits the structure graphically depicted in Figure 20.

This extension point defines the two elements: smartClassifier and aggregationDiscoverer.

The smartClassifier element represents a smart classifier and exposes the following attributes:

**name** : String
the name of the smart classifier

**class** : SmartClassifier
the Java class implementing the smart classifier

A smart classifier *must* also declare the set of smart classes which will fill by defining a set of the sub-elements of type outputSmartClass which exposes the following attributes:

**equivalentTo** : String
an expression, that can be empty, which declares the equivalent classes of the smart class — it can be a full qualified class name, a short name without the IRI or a Manchester syntax expression

**name** : String
the full qualified name of the smart class with its IRI

**subClassOf** : String
an expression, that can be empty, which declares the super-classes of the smart class — it can be a full qualified class name, a short name without the IRI or a Manchester syntax expression

The aggregationDiscoverer element represents an aggregation discoverer and exposes the following attributes:

**name** : String
the name of the aggregation discoverer

**class** : SmartClassifier
the Java class implementing the aggregation discoverer

An aggregation discoverer *must* also declare the set of aggregation classes which will fill by defining a set of the sub-elements of type outputAggregationClass which exposes the following attributes:

**`equivalentTo`** : String
: an expression, that can be empty, which declares the equivalent classes of the aggregation class — it can be a full qualified class name, a short name without the IRI or a Manchester syntax expression

**`name`** : String
: the full qualified name of the aggregation class with its IRI

**`subClassOf`** : String
: an expression, that can be empty, which declares the super-classes of the aggregation class — it can be a full qualified class name, a short name without the IRI or a Manchester syntax expression

In addition, an aggregation class can also express a set of properties that will be used by the enrichment module by defining a set of aggregationProperty sub-elements which exposes the following attributes:

**`name`** : String
: the full qualified name of the aggregation property with its IRI

**`subPropertyOf`** : String
: an expression, that can be empty, which declares the super-property of the aggregation property

**`equivalentTo`** : String
: an expression, that can be empty, which declares an equivalent relation of the aggregation property

**`isObjectProperty`** : boolean
: a boolean indicating if the aggregation property is an object or data property

**`inverseOf`** : String
: an expression, that can be empty, which declares the inverse relation of the aggregation property

**`propertyChain`** : String
: an expression, that can be empty, which declares a property chain that will be used to infer the aggregation property

**`description`** : String
: an optional human-readable description of the aggregation property

Note that the enrichment coordinator will automatically create the smart and aggregation classes, so the only job of the enrichment module is to fill them as it seems fit.

### The `SmartClassifier` class

A smart classifier is represented by the abstract class `SmartClassifier` which extends the base class `EnrichmentModule`. This class is contained in the eu.posecco.sdss.lageneration.enrichment plug-in and its public methods are depicted in Figure 21.

This class exposes the following public methods:

`void` **`add`**`(OntologyIndividual individual, OntologyClass clazz)`
: effectively add the individual `individual` to the class `clazz`

`Collection<RealizationResult>` **`collect`**`()`
: retrieves the list of the suggested realizations

The `collect()` method should analyze the ontology and return the suggested classification for a number of ontology individuals, while the `add()` should effectively add these individuals to a specific class. Note that for performances reasons the `collect()` method should only read the PoSecCo ontology.

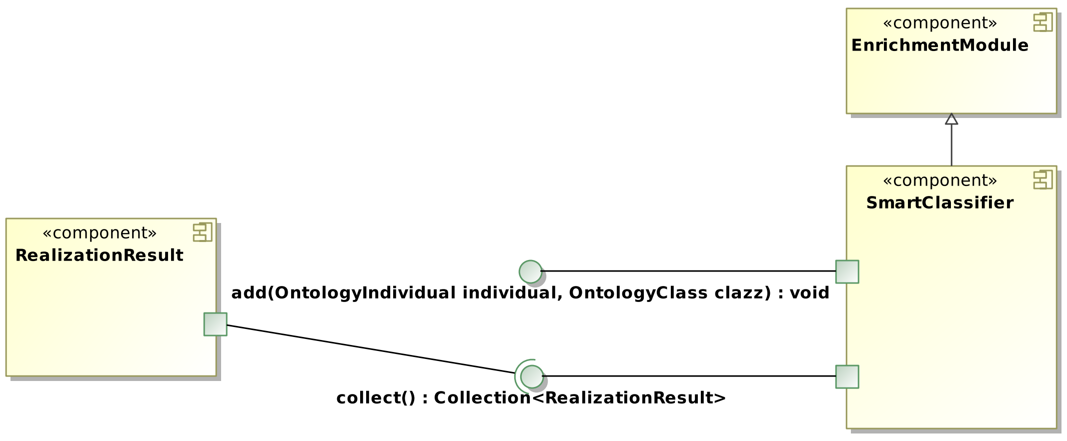
«component»
EnrichmentModule
«component»
SmartClassifier
«component»
RealizationResult
add(OntologyIndividual individual, OntologyClass clazz) : void
collect() : Collection<RealizationResult>

Figure 21: The `SmartClassifier` class API.

### The `AggregationDiscoverer` class

An aggregation discoverer is represented by the abstract class `AggregationDiscoverer` which extends the base class `EnrichmentModule`. This class is contained in the eu.posecco.sdss.lageneration.enrichment plug-in and its public methods are depicted in Figure 22.

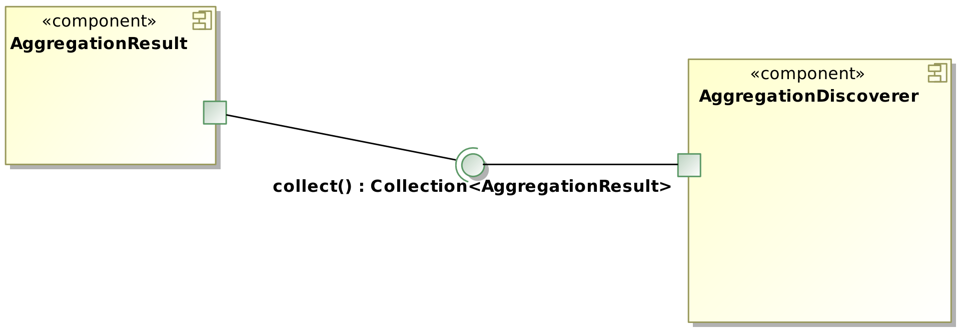

Figure 22: The `AggregationDiscoverer` class API.

This class exposes only one public methods:

`Collection<AggregationResult>` **`collect`**`()`
  retrieves the list of the suggested aggregations

The `collect()` method should analyze the ontology and return the suggested aggregation for a number of ontology individuals. Note that for performances reasons the `collect()` method should only read the PoSecCo ontology.

## Modifying the PoSecCo ontology

The first two steps of the LA Generation Service (the TBox and ABox module phases) create the initial PoSecCo ontology by merging together a set of smaller and independent ontologies. If the developer needs to modify the ontology structure (i.e., in order to add a new object property), he must edit these elementary ontologies which are stored in the eu.posecco.sdss.lageneration.model plug-in and in particular:

- all the TBox ontologies are located in the `tbox` directory;
- all the ABox ontologies are located in the `abox` directory.

In addition, the eu.posecco.sdss.lageneration.model plug-in contains the file `xml/tbox.xml` which declares a set of axioms that are automatically inserted in the PoSecCo ontology by the TBox module. These axioms are used to define a number of equivalence axioms in order to better merge together the TBox ontologies.

This file supports the following tags, defined by the XML schema listed in Listing 1:

**`classesEquivalence`**
declares an equivalence between two classes

**`objectPropertyEquivalence`**
declares an equivalence between two object properties

**`datPropertyEquivalence`**
declares an equivalence between two data properties

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="axioms" type="AxiomsType"></xsd:element>
  <xsd:simpleType name="AxiomKindType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="classesEquivalence">
      </xsd:enumeration>
      <xsd:enumeration value="objectPropertiesEquivalence">
      </xsd:enumeration>
      <xsd:enumeration value="dataPropertiesEquivalence">
      </xsd:enumeration>
  </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="AxiomType">
    <xsd:sequence>
    <xsd:element name="first" type="xsd:string" minOccurs="1" maxOccurs="
       1">
    </xsd:element>
    <xsd:element name="second" type="xsd:string" minOccurs="1" maxOccurs=
       "1">
    </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="kind" type="AxiomKindType" use="required">
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="AxiomsType">
    <xsd:sequence>
      <xsd:element name="axiom" type="AxiomType" maxOccurs="unbounded"
         minOccurs="1">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Listing 1: The XML schema for the equivalence axioms file.