Analysis Service - programmer manual

Politecnico di Torino

version 0.2.0 - 13 December 2013

http://www.posecco.eu

Contents

1 Introduction

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

The Analysis Service is used to perform a intra-policy and inter-policy analysis of filtering and data protection configurations. Filtering and data protection configurations are defined in "D3.3 Configuration Meta-Model".

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 Analysis 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.

Disclaimer This manual describes an *experimental prototype* that may be subject to substantial changes in future releases. Do not consider this documentation as in its final version.

Note In the current release, all the MoVE[1] integration features of the Analysis Service are temporarily disabled. When the MoVE project will be mature enough, they will be reactivated.

[1] More information about the MoVE project are available at `http://move.q-e.at/`.

2 Software architecture

Implementation

The Analysis 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.

Name	Website of the project
Eclipse plug-in framework	`http://eclipse.org/`
Remote Application Platform tool-kit	`http://eclipse.org/rap/`
Zest visualization tool-kit	`http://www.eclipse.org/gef/zest/`
OWL API ontology library	`http://owlapi.sourceforge.net/`
Pellet reasoner	`http://clarkparsia.com/pellet/`

Table 1: The technologies used in the Analysis Service .

Plug-ins

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

eu.posecco.sdss.analisysService

eu.posecco.sdss.analisysServiceLib

eu.posecco.sdss.analisysService.singleAnalyserDP

eu.posecco.sdss.analisysService.singleAnalyserF

eu.posecco.sdss.analisysService.distributedAnalyser

These plug-ins are extensively intertwined amongst them as shown in the dependency graph depicted in Figure 1.

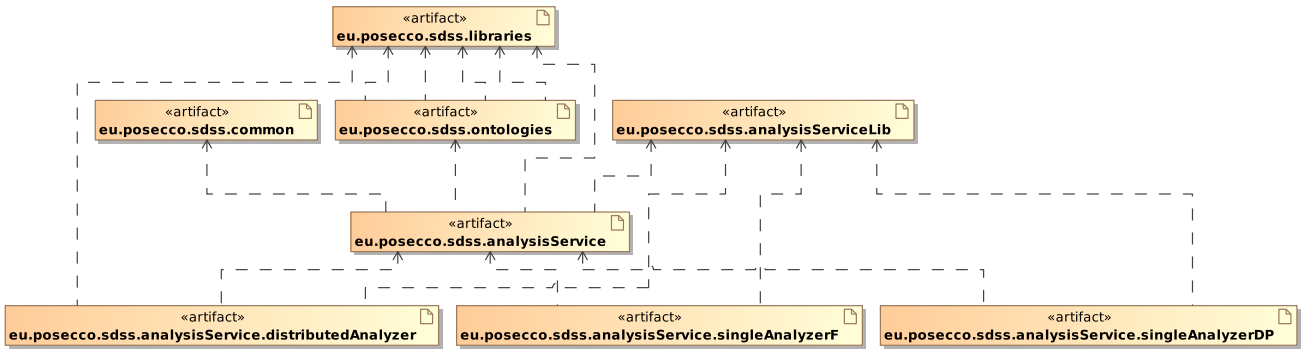

Figure 1: The Analysis Service plug-ins dependencies.

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
: include 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.libraries
: include all the external libraries needed by the Analysis Service

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

Plug-ins description

eu.posecco.sdss.analisysService

The "eu.posecco.sdss.analisysService" plug-in contains the views and the coordinator for the analysis service.
The main classes are:

AnalisysControler : This class is the centre part of the analysis service, its implemented as singleton.

AnalisysModel : This class is the centre model for the analysis Service, it contains the actual state of the ongoing analysis.

AnalyserInterface : This class is used by the extension point.

AnalyserView : This class implements the centre view, it contains the necessary widgets to allow the user to perform the various types of analysis.

DPPolicyView : This class implements the view for displaying a data protection configuration.

FPolicyView : This class implements the view for displaying a filtering configuration.

AnalyserResultView : This class implements the view for displaying the result of an analysis.

AnalyserDistResultView : This class implements the view for displaying the result of an analysis.

eu.posecco.sdss.analisysServiceLib

The "eu.posecco.sdss.analisysServiceLib" plugin contains the implementation of the internal model and the worker classes.

eu.posecco.sdss.analisysService.distributedAnalyser

The "eu.posecco.sdss.analisysService.distributedAnalyser" plug-in contains a implementation of the distributed analyser worker classes. It contains the class "DistributedAnalyser" which implements the abstract class "AnalyserInterface".

eu.posecco.sdss.analisysService.singleAnalyserDP

The "eu.posecco.sdss.analisysService.singleAnalyserDP" plug-in contains a implementation of the single filtering analyser worker classes. It contains the class "SingleAnalyserDP" which implements the abstract class "AnalyserInterface".

eu.posecco.sdss.analisysService.singleAnalyserF

The "eu.posecco.sdss.analisysService.singleAnalyserF" plug-in contains a implementation of the single data protection analyser worker classes. It contains the class "SingleAnalyserF" which implements the abstract class "AnalyserInterface".

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 Analysis Service .

Metric	Value
Number of plug-ins	6
Number of packages	35
Number of classes	134
Number of methods	1067
Number of lines	12386
McCabe cyclomatic complexity	1.88

Table 2: The source code metrics.

3 Public API

The Analysis Services is divided into four parts, the Filtering, the Data protection, the Distributed, and the main part.

The main part loads dynamically the other three parts, where each of this three parts have there own public API. Each of the three parts can be loaded contemporary by multiple instantiations, but for each part at least one instantiations has to be loaded. At runtime the user can chose which instantiations he wants to use for the analysis.

The three extension parts are based on the extension points of Eclipse plug-ins. The extension point used is called “analysisService” and it is defined in the plug-in “eu.posecco.sdss.analisysService”.

The extension point “analysisService” uses the abstract class AnalyserInterface, which has the following definition:

Listing 1: AnalyserInterface

```
public abstract class AnalyserInterface {
   private Policy policy;

   public void setPolicy(Policy policy){
     this.policy=policy;
   }

   public Policy getPolicy(){
     return policy;
   }

   public abstract List<PolicyAnomaly> getAnomalies() throws Exception;

   public abstract String getName();
}
```

`String` **`getName`**`()`
: returns the name of the analyser

`List<PolicyAnomaly>` **`getAnomalies`**`()`
: returns all anomalies found in the configuration

`Policy` **`getPolicy`**`()`
: returns the class Policy

`void` **`setPolicy`**`(Policy policy)`
: sets the class Policy

`Policy`

The class `Policy` used by the various implementations of an analyser is an abstract representation of configuration which has to be analysed.

`PolicyAnomaly`

The class `PolicyAnomaly` contains the anomaly found between two rules of the analysed configuration.

`DistributedAnalyser`

The class `DistributedAnalyser` implemented in the plug-in "eu.posecco.sdss.analisysService.distributedAnalyser" implements the abstract class "AnalyserInterface". The overwritten function "`List<PolicyAnomaly> getAnomalies()`" is a implementation of a distributed analysis of rule set configurations.

`SingleAnalyserDP`

The class `SingleAnalyserDP` implemented in the plug-in "eu.posecco.sdss.analisysService.singleAnalyserDP" implements the abstract class "AnalyserInterface". The overwritten function "`List<PolicyAnomaly> getAnomalies()`" is a implementation of a analysis of a data protection configuration.

`SingleAnalyserF`

The class `SingleAnalyserF` implemented in the plug-in "eu.posecco.sdss.analisysService.singleAnalyserF" implements the abstract class "AnalyserInterface". The overwritten function "`List<PolicyAnomaly> getAnomalies()`" is a implementation of a analysis of a filtering configuration.

4 Extending the tool

This section describes how extend the Analysis Service functionalities by specifying the classes, extension points and files involved in the process. For all three types of analysers a new Eclipse plug-in must be implemented which uses the extension point “analysisService”.

To use this extension point it must be configured with the following three parameters: name, type, and class. The name parameter is the name of the new extension. The type parameter is the type of the new extension and can be set to “SingleAnalyserF” for a filtering analyser, “SingleAnalyserDP” for a data protection analyser, and “RuleAnalyser” for a distributed analyser. The class parameter defines the class which implements the abstract class “AnalyserInterface”.

Filtering

To implement a new filtering configuration analyser, a new plug-in must be created which uses the extension point “analysisService” and implements a class which extends the abstract class AnalyserInterface.

This is a example definition for a filtering analysis service, the name of the analysis service chosen is “Single Analyser F”, for a filtering configuration analyser the type must be SingleAnalyserF and the class extending the abstract class AnalyserInterface is “eu.posecco.sdss.analysisService.singleAnalyserF.SingleAnalyserF”.

```
<extension point="eu.posecco.sdss.analysisService.analysisService">
 <module
  name = "Single Analyser F"
  class =
   "eu.posecco.sdss.analysisService.singleAnalyserF.SingleAnalyserF"
   type = "SingleAnalyserF">
 </module>
</extension>
```

DataProtection

To implement a new data protection configuration analyser, a new plug-in must be created which uses the extension point “analysisService” and implements a class extends the abstract class AnalyserInterface.

This is a example definition for a filtering analysis service, the name of the analysis service chosen is “Single Analyser DP”, for a data protection configuration analyser the type must be SingleAnalyserDP and the class extending the abstract class AnalyserInterface is “eu.posecco.sdss.analysisService.singleAnalyserDP.SingleAnalyserDP”.

```
<extension point="eu.posecco.sdss.analysisService.analysisService">
 <module
  name = "Single Analyser DP"
  class =
   "eu.posecco.sdss.analysisService.singleAnalyserDP.SingleAnalyserDP"
  type = "SingleAnalyserDP">
 </module>
</extension>
```

Distributed

To implement a new distributed analyser, a new plug-in must be created which uses the extension point “analysisService” and implements a class which extends the abstract class AnalyserInterface.

This is a example definition for a filtering analysis service, the name of the analysis service chosen is “Distributed Analyser”, for a distributed analyser the type must be RuleAnalyser and the class extending the abstract class AnalyserInterface is “eu.posecco.sdss.analysisService.distributedAnalyser.DistributedAnalyser”.

```
<extension point="eu.posecco.sdss.analysisService.analysisService">
 <module
  name = "Distributed Analyser"
  class =
   "eu.posecco.sdss.analysisService.distributedAnalyser.DistributedAnalyser"
  type = "RuleAnalyser">
 </module>
</extension>
```