# ITPolicy Tool - programmer's manual

Università degli Studi di Bergamo

version 0.1.0 - May 7th, 2013

SDSS

ITPolicyTool

PoSecCo

http://www.posecco.eu

## Contents

**1 Introduction** **2**

**2 Software architecture** **3**

Input/Output ITPolicy Tool . . . . . 4

Source Code . . . . . 4

Implementation . . . . . 5

Editor . . . . . 5

Harmonization . . . . . 6

Refinement . . . . . 7

Enrichment . . . . . 8

**3 Extending the tool** **9**

Enrichment . . . . . 9

Refinement . . . . . 10

## 1 Introduction

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

The IT Policy Tool supports the creation and the maintenance of the IT Policy and was implemented as an Eclipse plug-in and deployed as a RAP Application. The choice of Eclipse as the framework for the design of the PoSecCo IT Policy Tool is motivated by the flexibility of the framework and the support that it offers for the realization of effective interfaces for the management of rich structured elements. The IT Policy Tool facilitates the integration with the reasoning services implemented through the Semantic Web tools.

This document is structured as follows. The Section 2 is devoted to explain the IT Policy Tool internal structure by providing a bird's eye view of its architecture, its plug-ins and its types. The Section 3 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 IT Policy Tool 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/`.

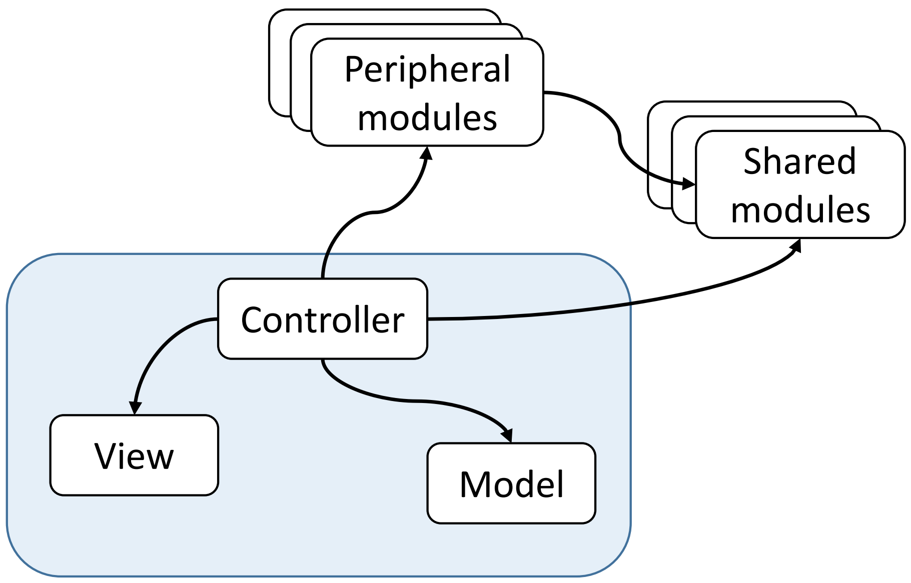


Figure 1: Architecture of the Tool

## 2 Software architecture

We introduce here the main design principles that will be followed in the construction of the PoSecCo IT Policy Tool, an Eclipse plugin that supports the definition of IT Policies. The pattern used in the development of the tool is the *Iterative and incremental development*. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system. Learning comes from both the development and use of the system, where possible key steps in the process start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added.

The ITPolicy Tool aims at allowing security administrators to define, manage and analyze security policies through several refinement steps. We have chosen to implement it on the basis of the Eclipse framework for four main reasons:

1. Eclipse is now one of the de-facto standards in terms of IDEs and has several plugins related to model driven engineering. The Eclipse framework is flexible enough to support the ITPolicy Tool requirements,
2. it provides several useful characteristics that ease the definition of the GUI of the tool,
3. it can be easily integrated with Semantic Web tools by using the *OWLAPI*, *Jena* libraries,
4. by defining a new *extension point*, it lets us define an extensible and flexible way to handle the integration and customization of new services in the architecture.

Figure 1 represents the abstract architecture of the tool. The pattern used to develop this level is the *Model-View-Controller* (MVC).

- **Controller** module. The Controller module represents the core of the entire plugin. It instantiates each module at start time, receives the command from the user (through the *View* module) and executes the related action on the data.
- **Model** module. The Model module permits to maintain a dynamic representation of the IT Policy. The model is a collection of Java classes that keep updated the information about the IT Policy.
- **View** module. The View module provides the functionalities to show the information on the screen.

Furthermore, there are two other modules:

- **Peripheral** module. It is also known as *Functional module* because it implements a specific functionality (e.g., Harmonization, Enrichment).
- **Shared** module. It provides functionalities that are common to several modules.

## Input/Output ITPolicy Tool

Due to the fact that all the artifacts produced by the *PoSecCo tools* (e.g., CoSerMas, SDSS) have to be stored in MoVE, the ITPolicy Tool provides a complete set of functionalities to retrieve and store artifacts in MoVE. More specifically, the ITPolicy Tool can receives as input (a) XMI files, (b) OWL files and (c) XML files, all of them in agreement with the IT layer Security meta-model. Furthermore, the ITPolicy tool can produce as output artifacts in (a) XMI format and (b) OWL format.

## Source Code

The source code of the ITPolicy Tool is available on the ITPolicy Tool web site [2]. In order to give an overview on the size and complexity of the implementative effort for the ITPolicy Tool, Table 1 provides a "qualitative" description through the use of several metrics. The metrics selected are the following:

- NOP: Number Of Packages
- NOC: Number Of Classes
- TLOC: Total Lines of Code
- NOM: Number of Methods
- VG: McCabe Cyclomatic Complexity

[2]http://cs.unibg.it/posecco/source.html

| **Project Name** | **NOP** | **NOC** | **TLOC** | **NOM** | **VG** |
|---|---|---|---|---|---|
| eu.posecco.businessrequirement | 5 | 6 | 538 | 33 | 1.175 |
| eu.posecco.enrichment.core | 12 | 27 | 3184 | 225 | 1.391 |
| eu.posecco.enrichment.landscape | 8 | 10 | 629 | 44 | 1.481 |
| eu.posecco.importfiles | 1 | 3 | 230 | 3 | 1.608 |
| eu.posecco.importontology | 2 | 6 | 322 | 16 | 1.85 |
| eu.posecco.itfunctionalmetamodel | 6 | 7 | 424 | 24 | 1.516 |
| eu.posecco.itharmonization.core | 8 | 48 | 7904 | 580 | 1.568 |
| eu.posecco.itharmonization.interactive.authenticationcheck | 1 | 1 | 55 | 1 | 1.12 |
| eu.posecco.itharmonization.interactive.cyclecheck | 1 | 4 | 188 | 4 | 1.375 |
| eu.posecco.itharmonization.modality.conflict | 4 | 12 | 2232 | 14 | 1.643 |
| eu.posecco.itharmonization.redundancy | 1 | 2 | 782 | 11 | 1.6 |
| eu.posecco.itharmonization.repair | 2 | 5 | 349 | 12 | 2.132 |
| eu.posecco.itharmonization.sod | 1 | 1 | 122 | 4 | 1.25 |
| eu.posecco.itontology | 4 | 5 | 1844 | 85 | 1.764 |
| eu.posecco.itpolicytool | 31 | 100 | 16767 | 835 | 2.164 |
| eu.posecco.itpolicytool.consumer | 2 | 3 | 64 | 7 | 0.923 |
| eu.posecco.itpolicytool.publisher | 1 | 2 | 49 | 6 | 1.067 |
| eu.posecco.itsearch | 3 | 5 | 582 | 20 | 1.576 |
| eu.posecco.itsecuritymetamodel | 5 | 16 | 5510 | 271 | 2.2 |
| eu.posecco.move | 5 | 128 | 41751 | 2084 | 3.009 |
| eu.posecco.neontoolkit.manager | 3 | 4 | 387 | 14 | 1.786 |
| eu.posecco.pellet | 3 | 4 | 1071 | 12 | 1.453 |
| eu.posecco.refinement.core | 15 | 29 | 4326 | 242 | 1.629 |
| Total | 124 | 428 | 89310 | 4547 | - |

Table 1: Metrics

## Implementation

Figure 3 provides an overview of all the packages, and relative dependencies, contained in the ITPolicy Tool.

### Editor

The ITPolicy Tool delivers several functionalities provided by different components.

- a main window to display and edit ITPolicy description files. The tool assigns a different tab panel in the window to every concept from the metamodel. This set of tabs provides the user with a high level guide through the model, always offering direct access to the main components of the model. The selection of a tab opens a form that permits to enter values for each of the properties of the corresponding entity. Instances of each class can be described in a homogeneous way by the same form used for entering its properties. The plug-in has many forms (see Figure 2 for an example). These forms allow to maintain a high usability of the tool:
- a navigation tool, that organizes resources in a finer taxonomy according to the specific top level concept selected by the user in the main window. The classification proposed to the user is driven by the ontology itself. This makes the tool quite flexible and automatically adaptable to changes in the ontology. The tool can be applied to any fragment of the metamodel.
- a search service, that offers the possibility to the user to search a specific element in the ITPolicy and the related element (e.g., business security requirements).
- a set of task buttons, published in the main toolbar, that offer direct access to functions like the creation of the OWL ontology starting from an instance of the security policy or the activation of the reasoning-based checking (i.e., harmonization).

Another functionality provided by the ITPolicy Tool to fulfill the requirements ITP-R08 and ITP-R09 is the possibility to link the ITPolicy elements to (a) the Business Security requirements, (b) the Functional System model and (c) the Abstract Configuration, in order to create the Policy chain and then store all of this data in *MoVE*. MoVE is the central repository for models and for the ontologies used in PoSecCo. Due to the fact that the link between the ITPolicy and the Abstract Configuration is built automatically through the *refinement phase*, the user must build, manually, the links among the ITPolicy and other layers (i.e., Business Security requirements, Functional System model). In order to aid the user in the building of links among meta-models, the ITPolicy Tool provides two views allowing the user to easily choose the elements, respectively business security requirements and IT functional elements, to link with a specific ITPolicy.

### Harmonization

One of the main functionalities provided by the ITPolicy Tool is the *Harmonization phase*. This phase allows the tool to prevent, detect and correct possible inconsistencies in the policies. Inconsistencies can represent misconfigurations, conflicts between different policies or simply suboptimal or redundant descriptions of the intended constraints. Harmonization is focused on the following aspects:

- verification of the correctness of each policy with respect to the PoSecCo policy ontology;
- detection of possible inconsistencies between contradictory policies at a given level of abstraction;
- detection of possible inconsistencies between an abstract policy and a policy that refines it at a lower level of details;
- identification of redundant policies in the same policy set.

According to the architecture presented in Section 2, the peripheral modules in charge to provide these functionalities are the following:

**Interactive modules:** these modules implement several interactive reasoning services which are executed during the editing phase of the *ITPolicy*. They are used to detect missing individuals in the ontology or to identify axioms that introduce structural violations in the ontology. These modules are implemented by using Semantic Web technologies, primarily by using *SPARQL-DL* due to its graph matching capabilities.

**eu.posecco.itharmonization.interactive.checkMissingITElement** implements a reasoning service that checks whether a certain *ITAuthenticationRule* exists or not. This functionality aids the user during the editing phase of the ITPolicy. We know that authorization and authentication are strictly related. For instance, it makes no sense define authorizations for a user on a system without an authentication that allows the user to logon on the system, and respectively define authentications for a user on a system without adding authorizations. Thus, in order to avoid this issue, the ITPolicy Tool provides the functionality to check the presence of an authentication when the user inserts an authorization.

**eu.posecco.itharmonization.interactive.checkStructuralConstraints** implements several interactive reasoning services that prevent the creation of structural inconsistencies (e.g., cycles in the group hierarchy, in the role hierarchy and in the security object hierarchy).

**Standard reasoning modules:** these modules can be used to detect anomalies and inconsistencies in the IT policy under development. In this way the *ITPolicy Tool* can report them to the user, and let him modify the model according to the analysis' results, in order to remove inconsistencies. These modules are implemented using Semantic Web technologies, i.e., the IT policy is represented as an *OWL ontology* and the reasoning services are expressed by means of Semantic Web tools, e.g., *OWL-DL*, *SWRL* and *SPARQL-DL*.

**eu.posecco.itharmonization.modalityconflict** is the component that implements the *Policy Incompatibility* reasoning service, which can be used to detect authorizations that are incompatible; e.g., it can detect if in the *IT-Model* there is an authorization that gives to a user the permission to execute a certain action and another authorization that removes from the same user the permission to execute the same action. It defines the *PolicyIncompatibility* extension for the *ITReasoningService* extension point. Three versions of the service exist: the first two versions implement the service by means of OWL-DL reasoning (the first one uses *Hermit* as reasoner whereas the second one uses *Pellet*) and the last one implements the service by means of *SWRL* rules.

**eu.posecco.itharmonization.redundancy** is the component that implements the *Redundancy Detection* reasoning service which can be used to detect redundancies in the *IT-Model*, i.e., the model may contain authorizations that are implied by other authorizations and thus can be removed. It defines the *RedundancyDetection* extension for the *ReasoningService* extension point. The reasoning service is implemented by means of SWRL rules and SPARQL-DL queries.

**eu.posecco.itharmonization.sod** is the component that implements the *SoD Conflict Detection* reasoning service which can detect authorizations that break Separation of Duty constraints expressed in the IT policy. It defines the *SoDConflictDetection* extension for the *ReasoningService* extension point. The reasoning service is implemented by using OWL-DL reasoning. Two versions of the service exist: the first one uses *Hermit* as reasoner, whereas the second one uses *Pellet*.

**Repair module:** This module implements the repair services that can provide the user with semi-automated repair capabilities. A repair service takes as input a list of *IFix* objects, which is the result of the execution of a reasoning service, and it computes a list of repair options that can be applied to the IT policy in order to remove the detected problems.

**eu.posecco.itharmonization.repair** implements two repair services, an automated one and a semi-automated one. The repair strategies implemented in both services are simple. In case the inconsistency is due to a *Redundancy* issue, the repair service removes the redundant authorizations. In case the inconsistency is due to a *Separation of Duty* issue, the repair service removes one of the role authorizations or role hierarchy axioms that cause the SoD. In case the inconsistency is due to a *Modality Conflict*, the repair service removes one of the system authorizations causing the conflict.

Furthermore, a shared module among all the components described above is the *Harmonization core*. It is the component that contains all the functionalities shared by the reasoning services. It contains the classes that manage the ontology by using the OWL-API Java library (enriched by the use of SWRL and SPARQL-DL). It allows the definition of an ontological representation of the IT Policy. *OWL-API* through *OWLDataFactory* objects permits to instantiate all classes, properties and axioms of the ontology. This component is a dependency of the reasoning service components presented above, because all these components use core functionalities. The dedicated functionalities of a specific reasoning service, e.g., a particular reasoner or a set of SWRL rules, are included only in the specific component.

### Refinement

As described in the IT layer Security meta-model supports the representation of authentication and access control policies. It introduces several concepts that offer a high degree of flexibility and modularity. Furthermore, an important characteristic of the IT layer Security meta-model is the integration with ontologies, which increase the expressive power of the model and permit to support both the evolution of the scenario and the realization of sophisticated checks (e.g., consistency). Furthermore, ontologies enrich concepts and provide a more detailed explanation of the concepts themselves. The use of ontologies is the foundation of the refinement (and enrichment) process.

As for the harmonization process, the policy refinement process is executed by orchestrating a set of refinement modules. Each refinement module can identify the set of model elements readily available for refinement and produce a new description fragment that describes abstract configurations. In order to manage, in a flexible way, the refinement process we have decided to adopt the same approach described in the harmonization process, thus we have implemented the refinement process with the use of *extension points* (see Section 3 for an exhaustive explanation).

The peripheral module in charge to provide these functionalities is the following:

**Refinement module:** This module implements the refinement service. The refinement service takes as input the enriched ontology, which is the result of the enrichment process, and it performs the transformation from the enriched IT Level to Abstract Configuration level.

**eu.posecco.refinement.core** implements the refinement service. It is composed by several modules, each dedicated to a specific target system (e.g., DBMS, OS). It defines the following extensions (both for authentication and authorization) (a) *DBMSRefinementModule* and (b)*OSRefinementModule* for the *RefinementModule* extension point (see Section 3 for an exhaustive explanation). The service is implemented by the use of *Jena* for the ontology management.

## Enrichment

The enrichment of IT policies that contain authorization or authentication rules is realized by importing enrichment ontologies that are specific to the target system's technology and type of policy (e.g., authentication or authorization). The ITPolicy Tool implements several enrichment modules that are executed during the enrichment process. They are used to introduce additional information. As for harmonization and refinement, the import of enrichment ontology is based on the use of *extension points*. The tool defines an extension point, called *Enrichment*, that allows other plugins to contribute with new enrichment modules.

The peripheral module in charge to provide these functionalities is the following:

**Enrichment modules:** This module implements the enrichment service. The enrichment service takes as input the ontological representation of the IT Policy and, through the import of additional ontologies, generates an enriched ontology containing information about authorization and authentication and the target system.

[**eu.posecco.enrichment.core**] implements the enrichment service. As for the refinement service, it is composed by several modules, each dedicated to a specific target system (e.g., DBMS, OS). For instance, it defines the following extensions (both for authentication and authorization): (a) *DBEnrichmentModule* (for MySQL 5.*) and (b)*OSEnrichmentModule* (for CentOS 5.*) for the *EnrichmentModule* extension point (see Section 3 for an exhaustive explanation). In order to aid the user in the import of the enrichment ontology, the module provides the functionality to use a custom wizard for each enrichment ontology.

# 3 Extending the tool

This section describes how extend the ITPolicy Tool functionalities by specifying the classes, extension points and files involved in the process.

## Enrichment

The *Policy Enrichment* plug-in implements several enrichment modules which are executed during the enrichment process. They are used to introduce additional information. In general, the enrichment process is led by the CPE name attribute of the class *ITResource* related to the IT element. The *Policy Enrichment* defines an extension point, called *Enrichment*, that allows other plugins to contribute with new enrichment modules.

```
<element name="module">
    <complexType>
      <attribute name="name" type="string" />
      <attribute name="CPEName" type="string" use="required">
        <annotation>
          <documentation>
            The CPE of the element enriched by the module.
          </documentation>
        </annotation>
      </attribute>
      <attribute name="ontologyIRI" type="string" use="required">
        <annotation>
          <documentation>
            The IRI of the ontology used in the EM.
          </documentation>
        </annotation>
      </attribute>
      <attribute name="wizard" type="string">
        <annotation>
          <documentation>
            The dedicated wizard for the EM.
          </documentation>
          <appinfo>
            <meta.attribute kind="java"
            basedOn="eu.posecco.enrichment.core.wizard.
               EnrichmentModuleWizard:"/>
          </appinfo>
        </annotation>
      </attribute>
    </complexType>
  </element>
```

Listing 1: Definition of Enrichment Extension Point

The extension point, whose definition is presented in Listing 1, has four attributes:

1. *name* represents the name of the enrichment module,
2. *CPEName* represents the CPE name of the element which has to be enriched,
3. *ontologyIRI* represents the enrichment ontology IRI,
4. *wizard* represents the class that implements a dedicated wizard in order to help the user, in the addition to concepts belong to the enrichment ontology.

As described before, the plug-in that wants to contribute with new functionalities has to define an extension satisfying the extension point. Figure 4 shows an example of the implementation of the extension point.

## Refinement

As for the policy enrichment, the policy refinement process is executed by orchestrating a set of refinement modules. Each refinement module can identify the set of model elements readily available for refinement and produce a new description fragment that describes abstract configurations. In order to manage, in a flexible way, the refinement process we have decided to adopt the same approach of policy enrichment, thus we have implemented the refinement process with the use of an *extension point*.

```
<element name="module">
    <complexType>
      <attribute name="systemType" use="required">
        <simpleType>
          <restriction base="string">
            <enumeration value="DBMS"></enumeration>
            <enumeration value="OS"></enumeration>
            . . .
          </restriction>
        </simpleType>
      </attribute>
      <attribute name="elementType" use="required">
        <simpleType>
          <restriction base="string">
            <enumeration value="ITSystemAuthorization"></enumeration>
            <enumeration value="ITRoleAuthorization"></enumeration>
            <enumeration value="ITAuthenticationRule"></enumeration>
            . . .
          </restriction>
        </simpleType>
      </attribute>
      <attribute name="class" type="string" use="required">
        <annotation>
          <appinfo>
            <meta.attribute kind="java" basedOn=":eu.posecco.refinement.
               core.IRefinementModule"/>
          </appinfo>
        </annotation>
      </attribute>
      <attribute name="name" type="string" use="required" />
    </complexType>
  </element>
```

Listing 2: Definition of Refinement Extension Point

The extension point, whose definition is presented in Listing 2, has four attributes:

1. *name* represents the name of the refinement module,
2. *class* represents the class that implements the refinement module for the specific IT element,
3. *elementType* represents the type of the IT element (e.g., ITSystemAuthorization, ITRoleAuthorization),
4. *systemType* represents the type of the concrete system involved in the policy (e.g., DBMS, OS)

Figure 5 shows an example of the implementation of a refinement module for DBMS (systemType attribute) involved in *ITSystemAuthorization*s (elementType attribute).

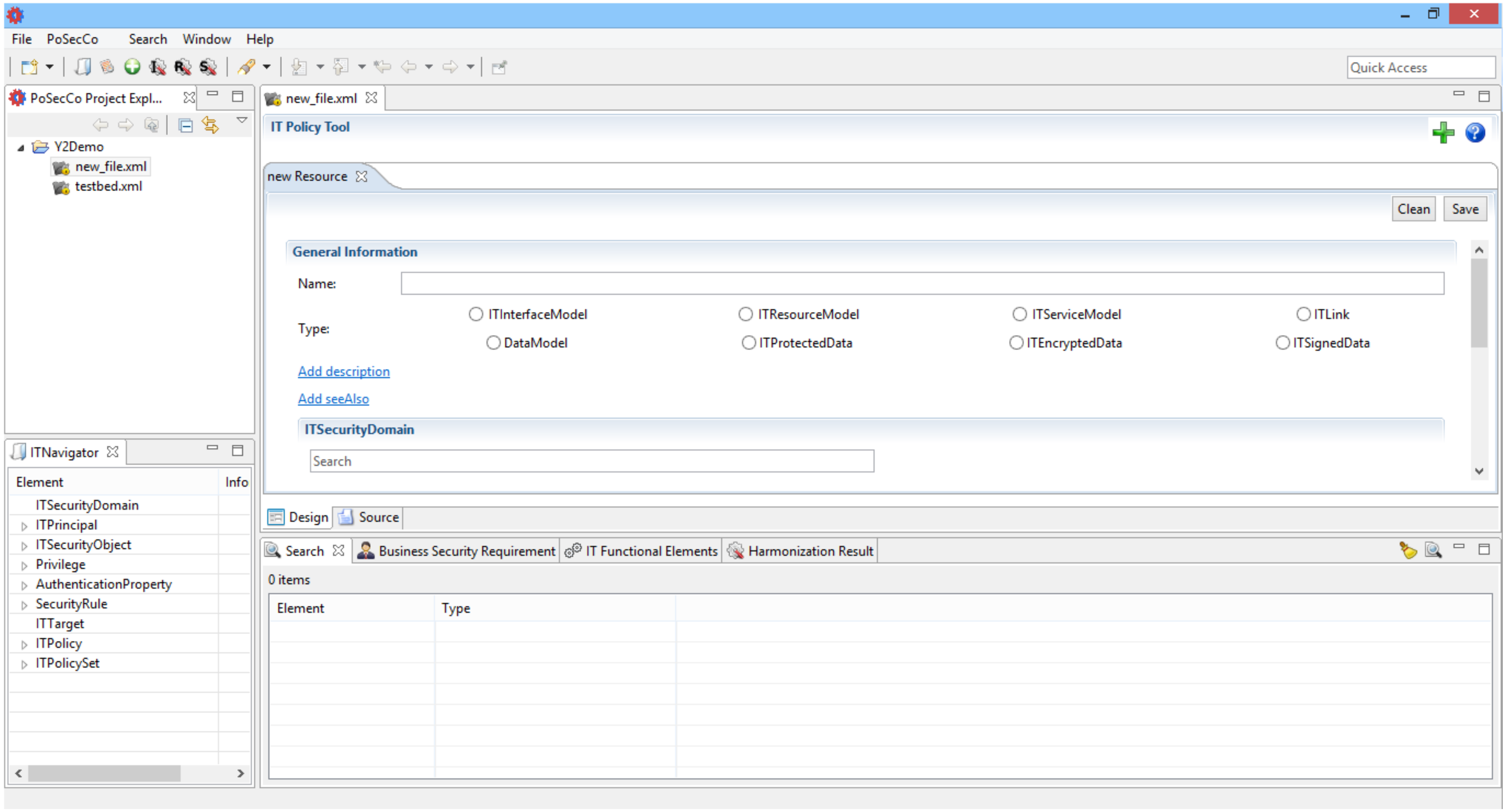


Figure 2: Components of the Tool

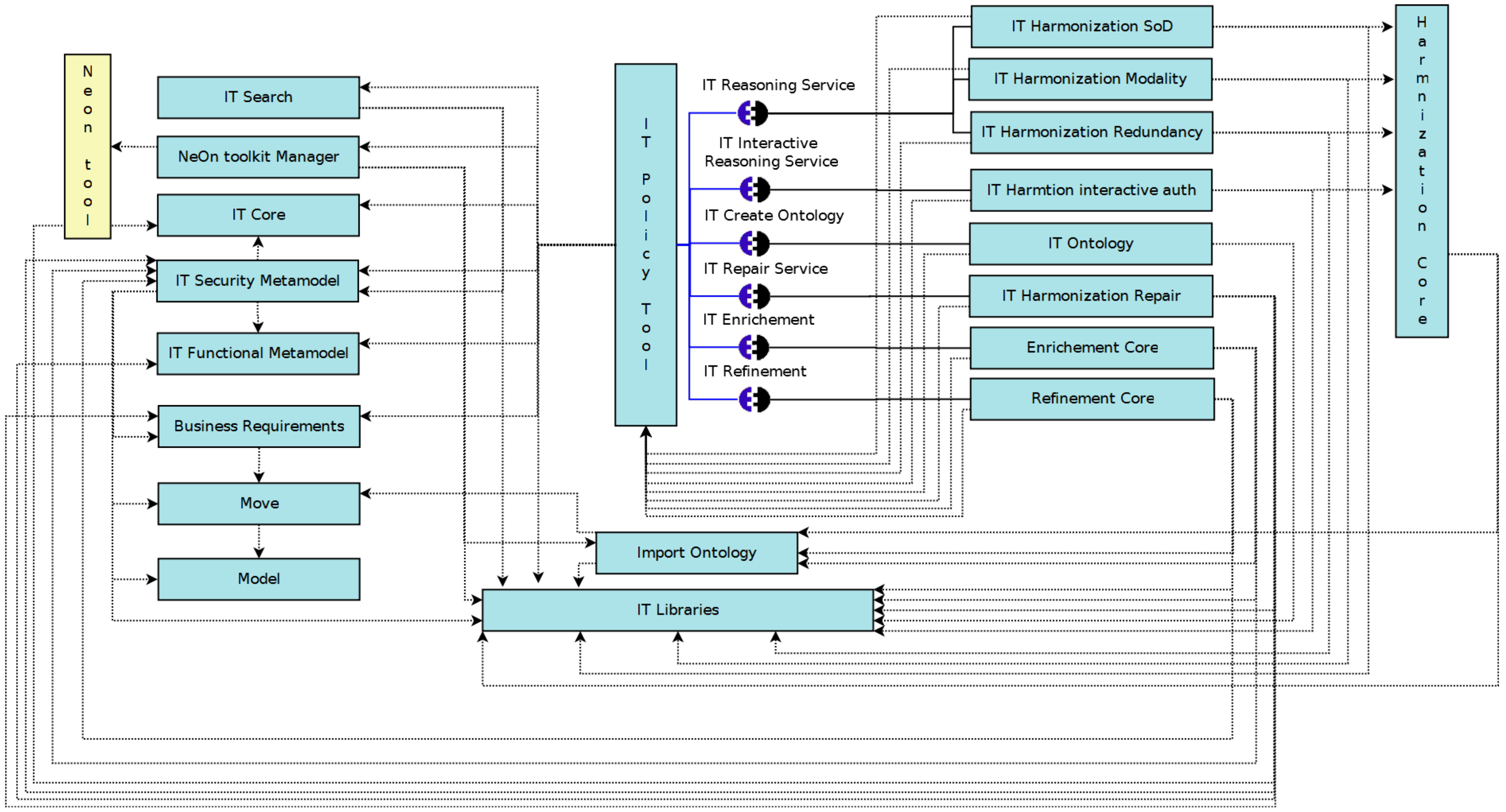
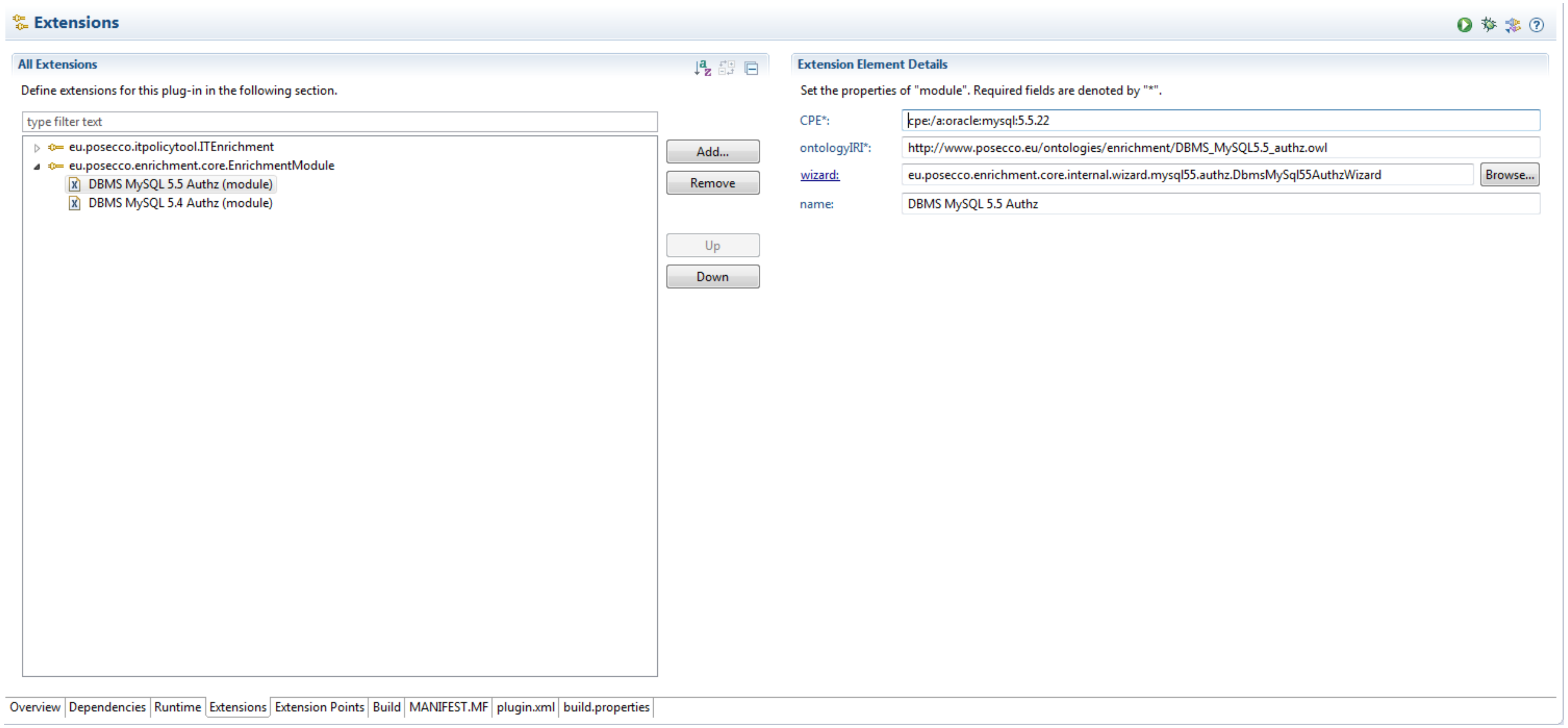


Figure 3: Components of the Tool



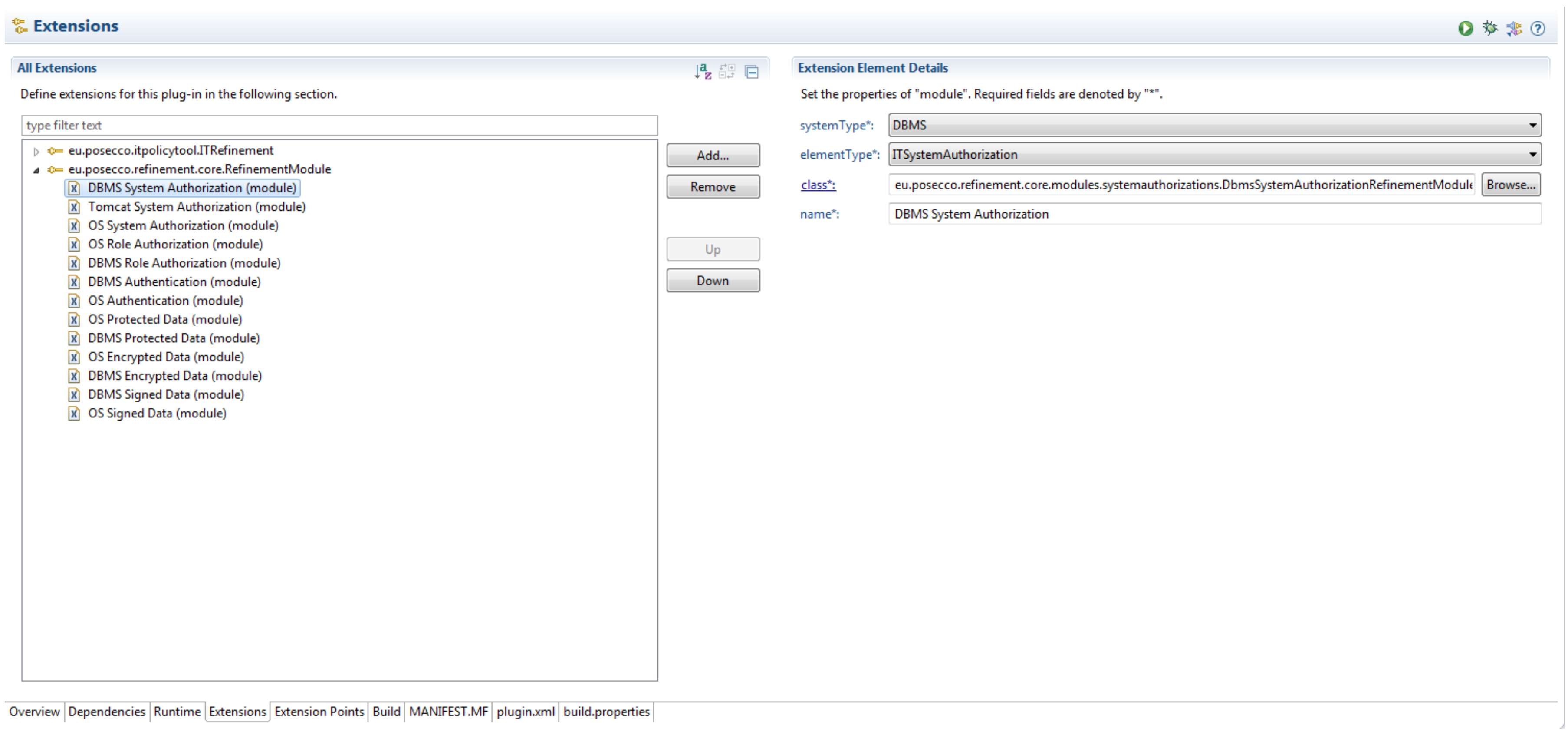
Define extensions for this plug-in in the following section.

- eu.posecco.itpolicytool.ITEnrichment
- eu.posecco.enrichment.core.EnrichmentModule
  - DBMS MySQL 5.5 Authz (module)
  - DBMS MySQL 5.4 Authz (module)

Set the properties of "module". Required fields are denoted by "*".

| | |
|---|---|
| CPE*: | cpe:/a:oracle:mysql:5.5.22 |
| ontologyIRI*: | http://www.posecco.eu/ontologies/enrichment/DBMS_MySQL5.5_authz.owl |
| wizard: | eu.posecco.enrichment.core.internal.wizard.mysql55.authz.DbmsMySql55AuthzWizard |
| name: | DBMS MySQL 5.5 Authz |

Figure 4: Definition of an Enrichment extension.



- eu.posecco.itpolicytool.ITRefinement
- eu.posecco.refinement.core.RefinementModule
  - DBMS System Authorization (module)
  - Tomcat System Authorization (module)
  - OS System Authorization (module)
  - OS Role Authorization (module)
  - DBMS Role Authorization (module)
  - DBMS Authentication (module)
  - OS Authentication (module)
  - OS Protected Data (module)
  - DBMS Protected Data (module)
  - OS Encrypted Data (module)
  - DBMS Encrypted Data (module)
  - DBMS Signed Data (module)
  - OS Signed Data (module)

| | |
|---|---|
| systemType*: | DBMS |
| elementType*: | ITSystemAuthorization |
| class*: | eu.posecco.refinement.core.modules.systemauthorizations.DbmsSystemAuthorizationRefinementModule |
| name*: | DBMS System Authorization |

Figure 5: Definition of a Refinement extension.