Configuration Editor - programmer manual

Politecnico di Torino

version 0.2.0 - 13 December 2013

PoSecCo

http://www.posecco.eu

Contents

1 Introduction

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

The *Configuration Editor* supports only the insertion and removal of filtering, IPsec and WS-Security rules, this document describes how this functionalities are implemented and how a developer can extend them to support other types of configuration rules.

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 Configuration Editorinternal 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 Configuration Editorare 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

The configuration editor implementation is based on one single Eclipse plug-in, which contains all necessary perspectives, views, handlers, and helper classes to read and write the rule set configurations.

Implementation

The Configuration Editoris 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/`
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 Configuration Editor.

Plug-ins

Since the Eclipse framework was adopted, all the Configuration Editorcode is contained in a eclipse plug-in, the eu.posecco.sdss.configurationeditor plug-in is extensively intertwined amongst other plug-ins as shown in the dependency graph depicted in Figure 1.

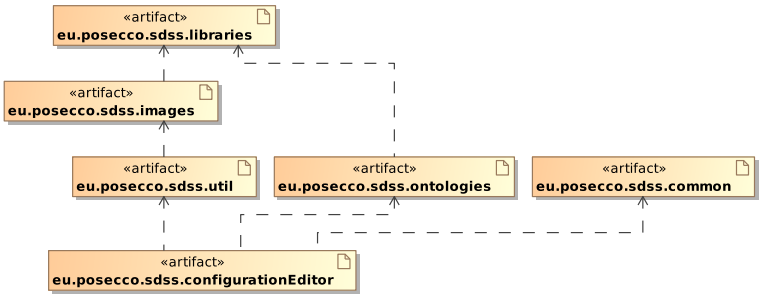

Figure 1: The Configuration Editorplug-ins dependencies.

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.images
: include all the icons and images

eu.posecco.sdss.libraries
: include all the external libraries needed by the Configuration Editor

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

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

Plug-in description

The eu.posecco.sdss.configurationeditor is composed by the following packages:

eu.posecco.sdss.configurationeditor.action
: This package contains all classes related to GUI action events such as insert new rules and remove existing ones.

eu.posecco.sdss.configurationeditor.handlers
: This package contains all classes related to GUI action handlers.

eu.posecco.sdss.configurationeditor.perspective
: This package contains the class which models the Eclipse perspective of the configuration editor.

eu.posecco.sdss.configurationeditor.provider
: This package contains all classes which implement the ContentProviders and LabelProviders for the views.

eu.posecco.sdss.configurationeditor.validator
: This package contains all classes which implement the input validators for the wizards.

eu.posecco.sdss.configurationeditor.view
: This package contain the classes which model the views for filtering configurations and data protection configurations.

eu.posecco.sdss.configurationeditor.wizard
: This packet contain the classes related to wizards, such as new filtering rule wizard, new IPsec rule wizard and new WSsec rule wizard.

eu.posecco.sdss.configurationeditor.wrapper
: This packet contains the helper classes to read and write from and to the ontology.

eu.posecco.sdss.configurationeditor.action

The package eu.posecco.sdss.configurationeditor.action contains the following classes:

`AddFilteringRuleAction`
: The AddFilteringRuleAction handles the event in the case a user wants to add a new filtering rule to a existing filtering configuration.

`AddIPSecRuleAction`
: The AddIPSecRuleAction handles the event in the case a user wants to add a new IPSec rule to a existing data protection configuration.

`AddWSSecRuleAction`
: The AddWSSecRuleAction handles the event in the case a user wants to add a new WS-Security rule to a existing data protection configuration.

`AddWSSecRuleAction`
: The AddSSHRuleAction handles the event in the case a user wants to add a new SSH rule to a existing data protection configuration.

`RemoveFilteringRuleAction`
: The RemoveFilteringRuleAction handles the event in the case a user wants to remove a filtering rule from a existing filtering configuration.

`RemoveIPSecRuleAction`
: The RemoveIPSecRuleAction handles the event in the case a user wants to remove a IPsec rule from a existing data protection configuration.

`RemoveWSSecRuleAction`
: The RemoveWSRuleAction handles the event in the case a user wants to remove a WS-Security rule from a existing data protection configuration.

`RemoveWSSecRuleAction`
: The RemoveSSHRuleAction handles the event in the case a user wants to remove a SSH rule from a existing data protection configuration.

eu.posecco.sdss.configurationeditor.provider

The package eu.posecco.sdss.configurationeditor.provider contains the following classes:

`ChprotConfContentProvider`
: The ChprotConfContentProvider class is the ContentProvider for the ChprotConfView.

`ChprotConfLabelProvider`
: The ChprotConfLabelProvider class is the LabelProvider for the ChprotConfView.

`FilteringConfContentProvider`
: The FilteringConfContentProvider class is the ContentProvider for the FilteringConfView.

`FilteringConfLabelProvider`
: The FilteringConfLabelProvider class is the LabelProvider for the FilteringConfView.

eu.posecco.sdss.configurationeditor.validator

The package eu.posecco.sdss.configurationeditor.validator contains the following classes:

`IntegerValidator`
: The IntegerValidator class verifies if the value of a TextBox is a valid number, otherwise the TextBox is painted red.

`IPValidator`
: The IPValidator class verifies if the value of a TextBox is a valid IP address , otherwise the TextBox is painted red.

`PortValidator`
: The PortValidator class verifies if the value of a TextBox is a valid port number, otherwise the TextBox is painted red.

eu.posecco.sdss.configurationeditor.view

The package eu.posecco.sdss.configurationeditor.view contains the following classes:

`ChprotConfView`
: The ChprotConfView is the view used to display data protection configurations in a tree structure. It uses the classes ChprotConfContentProvider and ChprotConfLabelProvider as ContentProvider and LabelProvider respectively.

`FilteringConfView`
: The FilteringConfView is the view used to display filtering configurations in a tree structure. It uses the classes FilteringConfContentProvider and FilteringConfLabelProvider as ContentProvider and LabelProvider respectively.

eu.posecco.sdss.configurationeditor.wizard

The package eu.posecco.sdss.configurationeditor.wizard contains the following classes:

`NewFilteringRuleWizard`
: The class NewFilteringRuleWizard is the wizard which is used as input form for a new filtering rule, it is composed by one single page (NewFilteringRuleWizardPage1).

`NewFilteringRuleWizardPage1`
: The class NewFilteringRuleWizardPage1 is the wizard page used by the NewFilteringRuleWizard and contains all necessary input fields and associated validators for a new filtering rule.

`NewIPsecRuleWizard`
: : The class NewIPsecRuleWizard is the wizard which is used as input form for a new IPsec rule, it is composed by one single page (NewIPsecRuleWizardPage1).

`NewIPsecRuleWizardPage1`
: The class NewIPsecRuleWizardPage1 is the wizard page used by the NewIPsecRuleWizard and contains all necessary input fields and associated validators for a new IPsec rule.

`NewWSRuleWizard`
: : The class NewWSRuleWizard is the wizard which is used as input form for a new WS-Security rule, it is composed by one single page (NewWSRuleWizardPage1).

`NewWSRuleWizardPage1`
: The class NewWSRuleWizardPage1 is the wizard page used by the NewWSRuleWizard and contains all necessary input fields and associated validators for a new WS-Security rule.

`NewSSHRuleWizard`
: : The class NewSSHRuleWizard is the wizard which is used as input form for a new SSH rule, it is composed by one single page (NewSSHRuleWizardPage1).

`NewSSHRuleWizardPage1`
: The class NewSSHRuleWizardPage1 is the wizard page used by the NewSSHRuleWizard and contains all necessary input fields and associated validators for a new WS-Security rule.

eu.posecco.sdss.configurationeditor.wrapper

The package eu.posecco.sdss.configurationeditor.wrapper contains the following classes:

`ReadDataProtectionConfiguration`
: The class ReadDataProtectionConfiguration is used to read data protection configurations from the ontology into the ConfigurationMetaModel used by PoSecCo.

`ReadFilteringConfiguration`
: The class ReadFilteringConfiguration is used to read filtering configurations from the ontology into the ConfigurationMetaModel used by Posecco.

`WriteConfiguration`
: The class WriteConfiguration is used to save data protection configurations and filtering configurations into the ontology.

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 Configuration Editor.

Metric	Value
Number of plug-ins	1
Number of packages	9
Number of classes	32
Number of methods	423
Number of lines	2502

Table 2: The source code metrics.

3 Public API

The centre part of the Configuration Editoris the `ConfigurationEditorController`, which contains all the model, the ontology and the model. This class `ConfigurationEditorController` is contained in the eu.posecco.sdss.configurationeditor plug-in.

ConfigurationEditorController

The `ConfigurationEditorController` class exposes the following public methods:

`ConfigurationEditorController``()`
: create a new ConfigurationEditorController, it's a private constructor since the class is implemented as a singleton.

`ConfigurationEditorController` **`getInstance`**`()`
: returns a instance of the class ConfigurationEditorController

`Ontology` **`getOwl`**`()`
: retrieve the current PoSecCo ontology

`Map<String, Collection<DataProtectionConfiguration>>` **`getDataProtConfList`**`()`
: returns all data protection configurations read from the ontology

`Map<String, Collection<FilteringConfiguration>>` **`getFilteringConfList`**`()`
: returns all filtering configurations read from the ontology

`void` **`refreshView`**`()`
: refreshes all registered views

`void` **`setFilteringConfView`**`(FilteringConfView filteringConfView)`
: sets the filtering configuration view

`void` **`setChprotConfView`**`(ChprotConfView chprotConfView)`
: sets the data protection configuration view

4 Extending the tool

This section describes how a new type of rule set configuration can be handled by the configuration editor.

View

First a new view for the new rule set configuration has to be created, the new view also needs to be inserted into the Perspective of the Configuration Editor.

Starting form the FilteringConfView the following lines of code needs to be modified:

The name of the view:

```
frmLogicalAssociationImplementations.setText("Configurations Explorer");
```

The list of configuration available needs to be loaded:

```
update(ConfigurationEditorController.getInstance().getFilteringConfList());
```

Insert the addRuleAction to the view, the element on which the right click was performed needs to be converted in the appropriated configuration type, and a new Action class needs to be implemented.

```
FilteringConfiguration fruleconf= (FilteringConfiguration)element;
menuMgr.add(new AddFilteringRuleAction(parent, fruleconf));
```

Insert the removeRuleAction to the view, the element on which the right click was performed needs to be converted in the appropriated rule type, and a new Action class needs to be implemented.

```
FilteringConfigurationRule frule= (FilteringConfigurationRule)element;
menuMgr.add(new RemoveFilteringRuleAction(parent, frule));
```

Insert the new ContentProvider to the treeViewer.

```
treeViewer.setContentProvider(new FilteringConfContentProvider());
```

Add the new LabelProvider to the treeViewer.

```
treeViewer.setLabelProvider(new FilteringConfLabelProvider());
```

ContentProvider

This class needs to be written for every configuration type, the FilteringConfContentProvider can be used as template.

LabelProvider

This class needs to be written for every configuration type, the FilteringConfLabelProvider can be used as template.

addRuleAction

Starting from the AddFilteringRuleAction the following lines of code needs to be modified:

The private variable which holds the configuration:

```
private FilteringConfiguration filteringConf;
```

The function which returns the text displayed after the right click:

```
public String getText() {
 return "Add Filtering Rule";
}
```

The creation of the appropriated Wizard, using the new Wizard:

```
WizardDialog wizardDialog = new WizardDialog(parent.getShell(),
         new NewFilteringRuleWizard(filteringConf));
```

removeRuleAction

Starting from the RemoveFilteringRuleAction the following lines of code needs to be modified:
The private variable which holds the configuration:

```
private FilteringConfiguration filteringConf;
```

The function which returns the text displayed after the right click:

```
public String getText() {
 return "Add Filtering Rule";
}
```

Wizard

Starting from NewFilteringRuleWizard the following lines of code needs to be modified:
The private variable which holds the configuration:

```
private FilteringConfiguration filteringConf;
```

The new WizardPage which contains the input form:

```
page_one = new NewFilteringRuleWizardPage1(new LinkedList<String>());
```

The function performFinish() has to be rewritten so that the new type of rule is inserted in the appropriated type of configuration.

```
public boolean performFinish() {}
```

Wizardpage

Starting from NewFilteringRuleWizardPage1 the following lines of code needs to be modified:
The function getFilteringRule() needs to be rewritten so that it returns the appropriated type of rule with the values inserted by the user in the input form.

```
public FilteringConfigurationRule getFilteringRule(){}
```

The function createControl(Composite parent) needs to be rewritten so that it models the input form appropriated for the new type of rule.

```
public void createControl(Composite parent) {}
```

Validator

Starting from the IntegerValidator the following lines of code needs to be rewritten:
The variable regexp specifies the regular expression which validates the value inserted by the user.

```
String regexp = "^[0-9]*$";
```