Open-PEOPLE : Architecture and Implementation

Jonathan Ponroy, Kévin Roussel, Olivier Zendra, Dominique Blouin

Open-PEOPLE

Open Power and Energy Optimization Platform and Estimator

Deliverable 2.7

Architecture and Implementation

	Document Manager	**Contributors**	**Checked by**
Name	Jonathan PONROY	Kévin ROUSSEL INRIA NGE	Kévin ROUSSEL Dominique BLOUIN Olivier ZENDRA
Contact	Jonathan.Ponroy@inria.fr		
Date	28-jun-2011 **(version 0.4)**		
Summary	This document describes implementations of plugins developed by INRIA NGE for the Open-PEOPLE SoftWare Platform (OPSWP). This document is for developers who want to modify or extend this plugins set.		

Table of contents

1. Preface

1.1. Versions

Version	*Date*	*Description & rationale of modifications*	*Sections mainly modified*
0.1	1 September 2010	First version	–
0.2	22 october 2010	Authentication plugin description	4
0.3	9 june 2011	Adding QUDV editor description	6
0.4	28 june 2011	Made corrections Added the section on the LUT editor	All 5

1.2. Table of references and applicable documents

Reference	*Title & edition*	*Author*	*Date*
Open-People project: deliverable D2.1	Specification for the software platform definition (V1)	Sophie ALEXANDRE	18 May 2010
Open-People project: deliverable D2.2	Tools Integration Protocol	Kévin ROUSSEL	10 September 2010
Open-People project: delivrable D4.1	Basic Components Model Homogenizations	Dominique BLOUIN	15 July 2010
Open-People project: delivrable D4.2	Generic models, interoperability and interchangeability	Dominique BLOUIN	N.D.

1.3. Acronyms and glossary

Term	*Definition*
OP	**Open-PEOPLE:** the name of the project we're talking about, and by extension, the name of the platform(s) developed within this project.
OPSWP	**Open-PEOPLE software platform:** the central piece of code, around which revolve all of the software developments undertaken within the Open-PEOPLE project.
Java	**The Java programming language, and the platform on which it executes,** once compiled into *bytecode* (Java virtual machine). Both created by *Sun Microsystems* (now *Oracle*), and defined respectively in: *"The Java Programming Language"* and *"The Java Virtual Machine Specification"* books.
JVM	**Java Virtual Machine:** main component of the Java platform, providing the latter with its portability and security (among others). The OPSWP is designed to be executed by the JVM.
Bytecode	**JVM bytecode:** machine code for the JVM, that is: program code directly executable by the JVM. The OPSWP and its plug-ins will ultimately take the form of executables coded in this format.
JCP	**Java Community Process,** by which *Sun Microsystems/Oracle* designs the evolution of Java (both the language and the platform) in cooperation with its users.
JSR	**Java Specification Request:** official proposals of evolution for Java (both the language and the platform), made within the JCP.

Term	***Definition***
OSGi	Formerly meaning ***"Open Services Gateway initiative",*** it is a framework designed by the *OSGi Alliance,* in order to provide dynamic modularization and *service-oriented architecture* (SOA) for Java-based applications.
CRI-NGE	**Centre de Recherche INRIA – Nancy Grand-Est:** the place where the development of OPSWP is managed and—for a major part—realized.
PCMD	**Power Consumption Models Development:** the set of features offered by the OP platform (hardware & software) which allow to develop and validate new power consumption models.
PCAO	**Power Consumption Analysis and Optimization:** the set of features offered by the OP platform which allow embedded system designers to estimate and optimize the energy consumption of the systems they create, using predefined consumption models.
GUI	**Graphical User Interface:** OPSWP will offer a GUI to allow for easy yet efficient and productive use.
RCP	**Rich Client Platform:** a framework (comprising widgets, desktop/workbench, extensible architecture, update management, and other paradigms) on which one can build coherent, robust, standardized, and feature-rich desktop stand-alone and client applications (thus named "rich client" applications). The OPSWP is to be built on Eclipse project's RCP.
IDE	**Integrated Development Environment:** a graphical application/framework gathering tools in order to help developers and programmers to perform more easily and efficiently their work; one of the most advanced, best known and most used IDE for the Java platform is the Eclipse JDT, built upon the RCP produced by the Eclipse project. A variant of this JDT is precisely specialized in development of applications based on Eclipse's RCP.
COP	**Component Oriented Programming:** Programming method which consists to use a modular approach to build software architecture using pre-existing components.
JAAS	**Java Authentication and Authorization Service:** is a Java security framework for user-centric security to augment the Java code-based security. Since Java Runtime Environment 1.4 JAAS has been integrated with the JRE - previously JAAS was supplied as an extension library by Sun.

2. Executive summary

Our software platform is designed, from the very beginning, to be extensible : not only is the library of available components (both hardware and software) for modeling open for extension with user-defined data; the set of features offered by the platform is also extensible, all of these features being provided by “plug-ins” (a.k.a. “add-ons”, i.e. code modules) integrated into OPSWP.

The purpose of this document is to describe the implementation of the Eclipse plugins developed by INRIA NGE within the development of the OPSWP, in collaboration with all project partners. More specifically, it describes in further details the functionalities cited in deliverable D2.1, at each plugin level.

This document, not planned at the beginning, is part of task 2. The task's goal is the specification, conception and implementation of OPSWP, which is the main task of the INRIA NGE team.

3. Scope of the document

This document is a ***technical*** one: it shall describe with all needed precision how plugins were implemented during the project, to allow developers to modify or extend existing plugins. This document is not to be frozen in time. It will evolve as new plugins for OPSWP are developed during and after the end of the project.

On the other hand, it is supposed that the reader of the present document knows what OP is, and what goals it is supposed to achieve. This document won't talk about these subjects: this information can be found in the D1.1 to D1.4 deliverables of the project.

The purpose of this document is not to discuss about components modeling (see task 4 of the project for this matter).

Moreover, this document focuses itself exclusively on the ***software platform*** (OPSWP). All data concerning the project's hardware platform is located in other deliverables (deliverables from task 3; plus D2.3 for the remote control of hardware platform).

4. Authentication plugin

This plugin provides a service allowing the tools composing the OPSWP to authenticate a user, and thus to limit accordingly the functionalities available to him (her). Consequently, any OPSWP add-on wishing to use any “secured” feature of the platform (and especially communication with the remote elements like the OPHWP) ***must*** use this service to meet the ad-hoc safety requirements.

4.1. Functionalities

The authentication plugin allows to meet both of these requirements (described in D2.1):

[REQ-GNFCT-010] For any remote access by the software platform, authentication will be required to ensure security

[REG-GNFCT-020] It will be necessary to implement an authorization control the access to some parts or functionalities of the software platform. For that a Role-Based Authorization mechanism could be used

This plugin is thus the mecanism that provides and manages the authentication and authorization features on which the platform's safety relies.

4.2. Architecture

From an architectural point of view, the authentication plugin is divided into:

- an *interface*, package named `fr.openpeople.platform.auth` ;
- *core* packages named:
  - `fr.openpeople.platform.auth.internal` package, for the internal mechanism,
  - `fr.openpeople.platform.auth.internal.ui` package, for the general user interface (U.I.), that is: the views provided by the plugin,
  - the `fr.openpeople.platform.auth.preferences` package, for user accounts preference page.

The plugin programming “interface” was widely described in D2.2, so we will only briefly describe it here as a reminder.

4.2.1. Interface

As we said before, the public part of the authentication plugin is implemented by the `fr.openpeople.platform.auth` package, which is described in figure 4.1.

The authentication plugin provides an extension point and an interface to use the authentication & authorization services:

- extension point: **fr.openpeople.platform.auth.tokenConsumer**
- interface: **`OPSWPAuthTokenReceptor`**

By using both these elements, an OPSWP plugin has the ability to request an **OPSWPAuthToken,** containing all the needed information about the currently connected user.

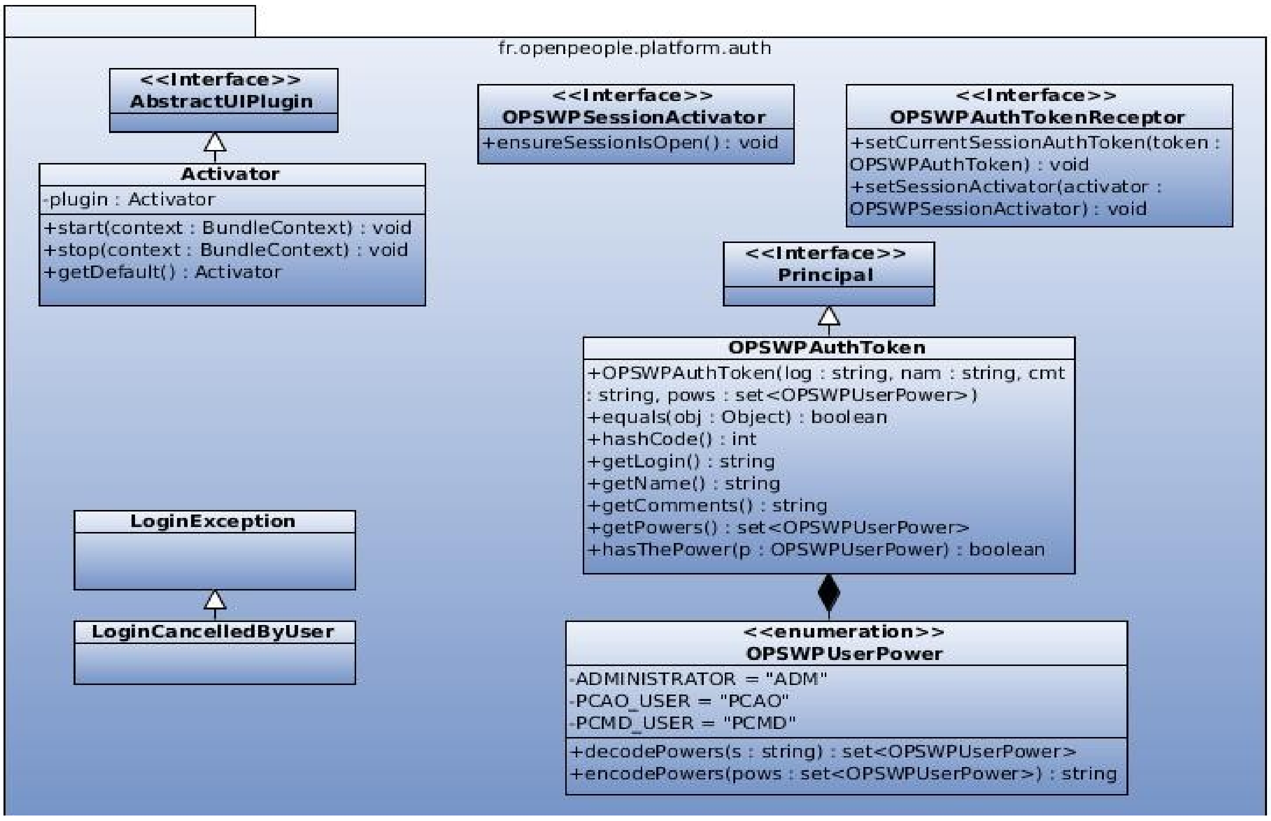
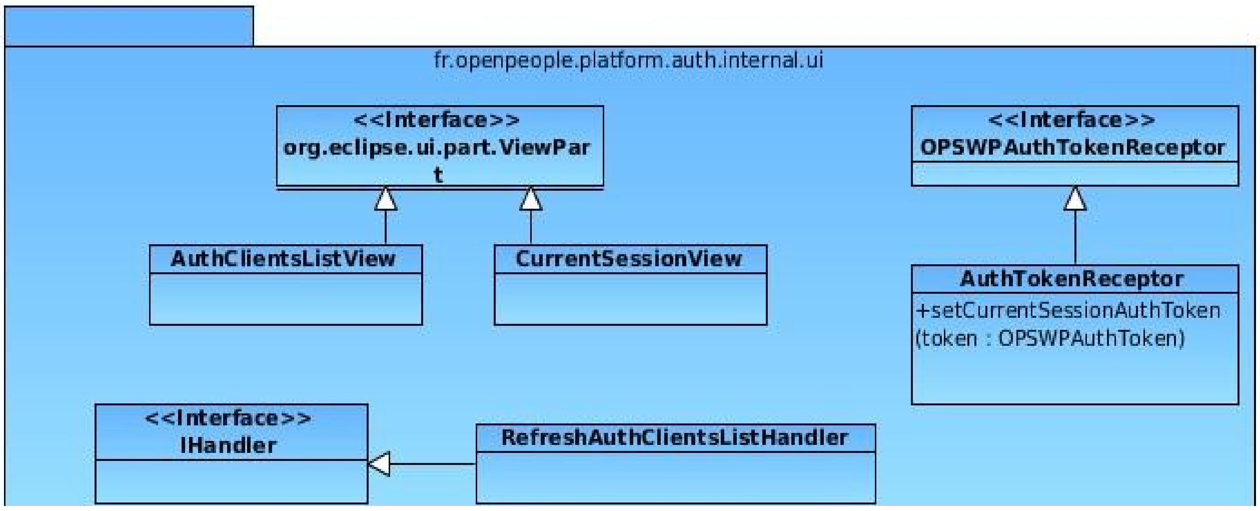

fr.openpeople.platform.auth
<<Interface>> AbstractUIPlugin
Activator
-plugin : Activator
+start(context : BundleContext) : void
+stop(context : BundleContext) : void
+getDefault() : Activator
<<Interface>> OPSWPSessionActivator
+ensureSessionIsOpen() : void
<<Interface>> OPSWPAuthTokenReceptor
+setCurrentSessionAuthToken(token : OPSWPAuthToken) : void
+setSessionActivator(activator : OPSWPSessionActivator) : void
<<Interface>> Principal
OPSWPAuthToken
+OPSWPAuthToken(log : string, nam : string, cmt : string, pows : set<OPSWPUserPower>)
+equals(obj : Object) : boolean
+hashCode() : int
+getLogin() : string
+getName() : string
+getComments() : string
+getPowers() : set<OPSWPUserPower>
+hasThePower(p : OPSWPUserPower) : boolean
LoginException
LoginCancelledByUser
<<enumeration>> OPSWPUserPower
-ADMINISTRATOR = "ADM"
-PCAO_USER = "PCAO"
-PCMD_USER = "PCMD"
+decodePowers(s : string) : set<OPSWPUserPower>
+encodePowers(pows : set<OPSWPUserPower>) : string

Figure 4.1: interface's plugin class diagram

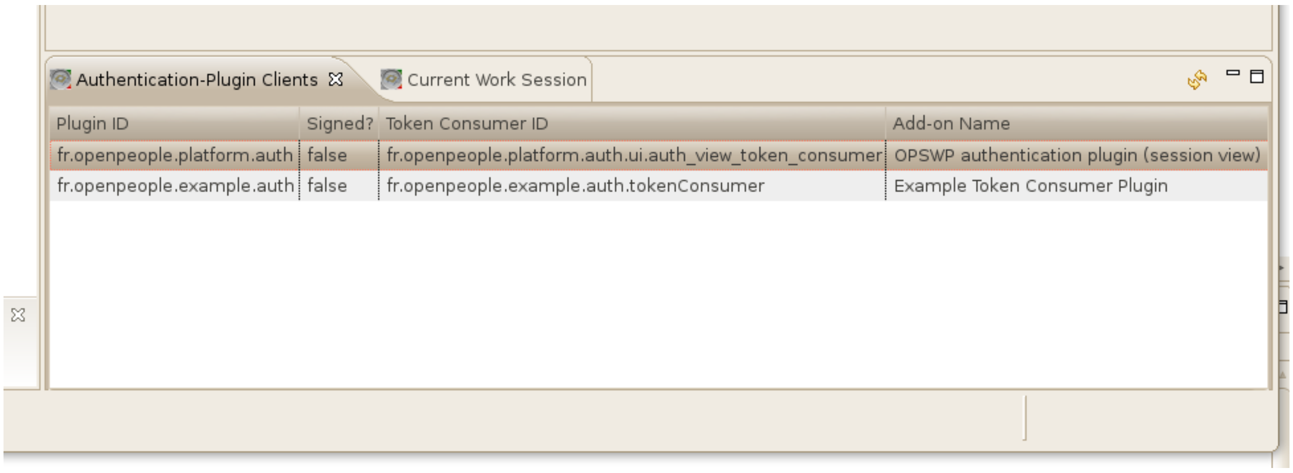
4.2.2. Core

As we said before, the plugin core is divided into 3 packages.

The `fr.openpeople.platform.auth.internal.ui` package (figure 4.2) implements the views that the plugin contributes to the Eclipse workbench.


fr.openpeople.platform.auth.internal.ui
<<Interface>>
org.eclipse.ui.part.ViewPart
AuthClientsListView
CurrentSessionView
<<Interface>>
OPSWPAuthTokenReceptor
AuthTokenReceptor
+setCurrentSessionAuthToken(token : OPSWPAuthToken)
<<Interface>>
IHandler
RefreshAuthClientsListHandler
Figure 4.2: fr.openpeople.platform.auth.internal.ui package

The `AuthClientListView` class provides the authentication-plugin clients view (figure 4.3). This view shows which third party plugins are using the authentication plugin services.

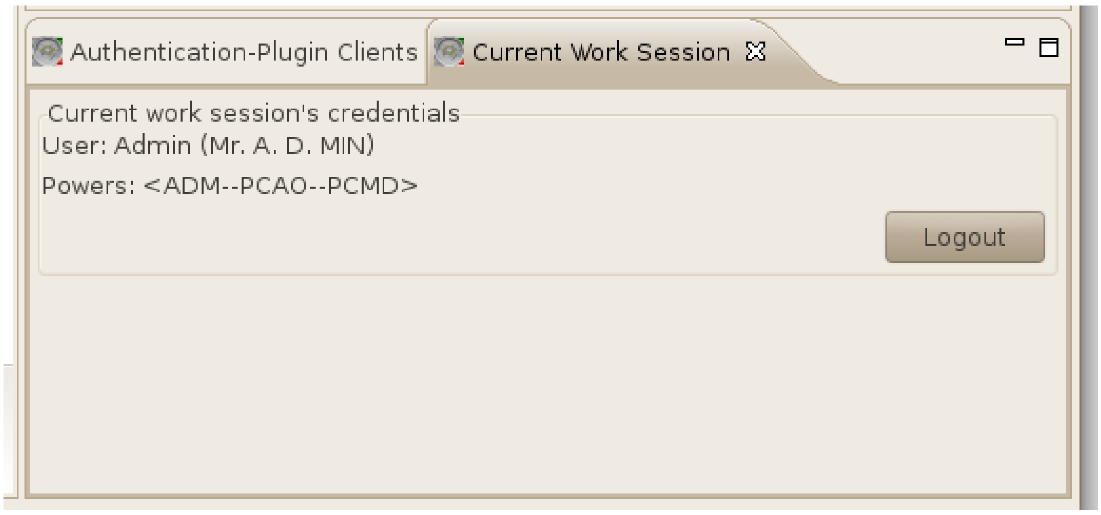


Plugin ID	Signed?	Token Consumer ID	Add-on Name
fr.openpeople.platform.auth	false	fr.openpeople.platform.auth.ui.auth_view_token_consumer	OPSWP authentication plugin (session view)
fr.openpeople.example.auth	false	fr.openpeople.example.auth.tokenConsumer	Example Token Consumer Plugin

Figure 4.3: authentication plugin client view

The `RefreshAuthClientsListHandler` class handles the refresh of the client plugins table within the former view; it is activated by that view's refresh button.

The `CurrentSessionView` class provides the current work session view (figure 4.4). This view, shows information about the currently connected user (the same information that is provided to third party plugins via `OPSWPAuthToken` instances), and allows this user to logout. It also allows to login when no work session is open.

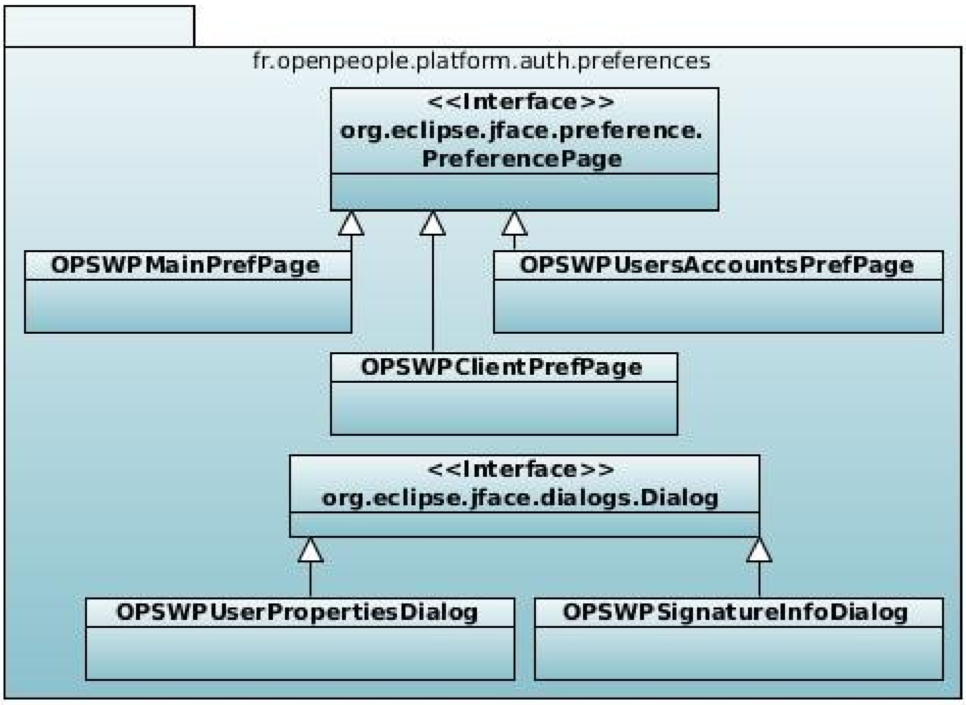


Current work session's credentials
User: Admin (Mr. A. D. MIN)
Powers: <ADM--PCAO--PCMD>

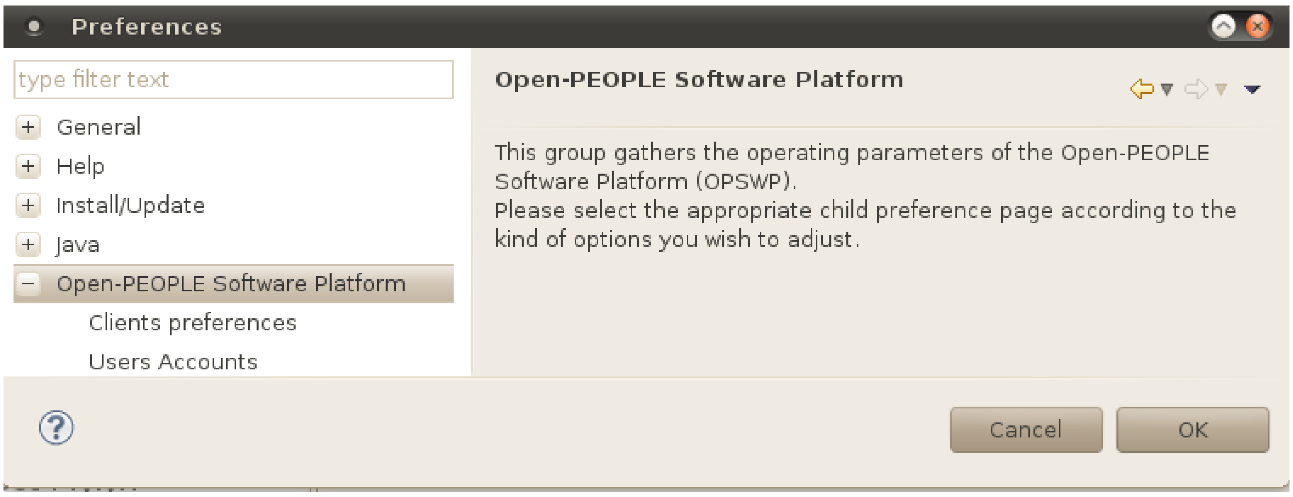
Figure 4.4: current work session view

The `AuthTokenReceptor` class implements the `OPSWPAuthTokenReceptor` interface. Reminder: this interface is to be implemented by every authentication client, in order to receive a reference on a new `OPSWPAuthToken` instance every time the information about the current work session changes (i.e.: when a user logs in or out).

The `fr.openpeople.platform.auth.preferences` package provides the preference pages contributed to the Eclipse workbench, as well as the dialog boxes used from within these preference pages.


fr.openpeople.platform.auth.preferences
<<Interface>> org.eclipse.jface.preference. PreferencePage
OPSWPMainPrefPage
OPSWPUsersAccountsPrefPage
OPSWPClientPrefPage
<<Interface>> org.eclipse.jface.dialogs.Dialog
OPSWPUserPropertiesDialog
OPSWPSignatureInfoDialog

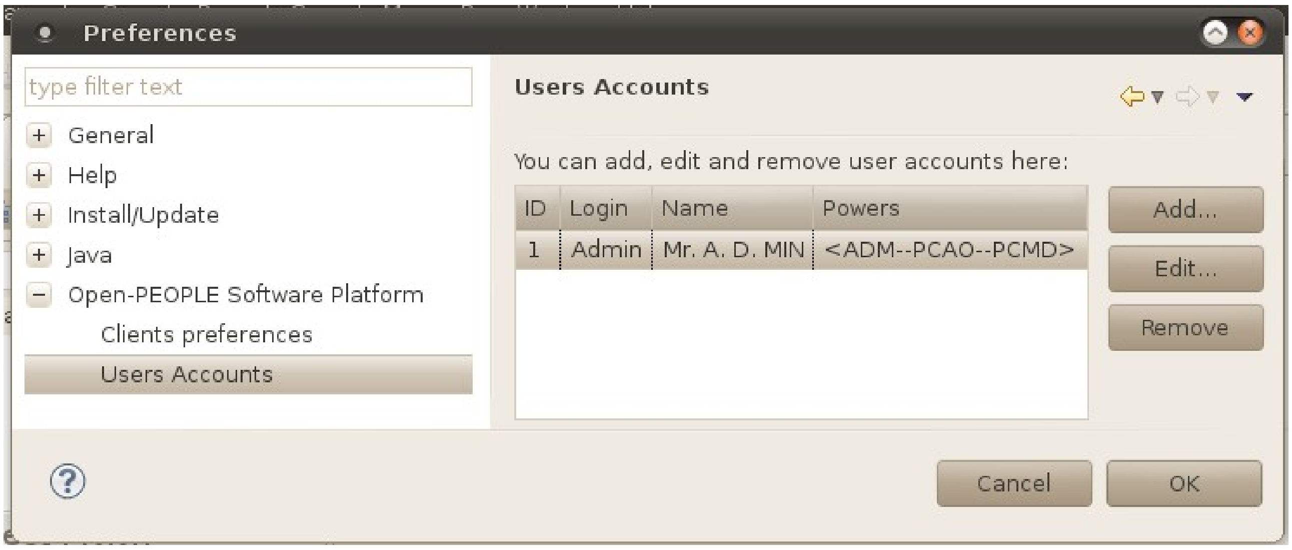
The `OPSWPMainPrefPage`, `OPSWPClientPrefPage` and `OPSWPUsersAccountsPrefPage` classes are the implementations of the cited preference pages. They respectively provide the OPSWP preference group introduction (figure 4.5), user accounts management interface (figure 4.6) and client plugins' authorizations management (figure 4.7).



Open-PEOPLE Software Platform

This group gathers the operating parameters of the Open-PEOPLE Software Platform (OPSWP).
Please select the appropriate child preference page according to the kind of options you wish to adjust.

Figure 4.5: Main preference page

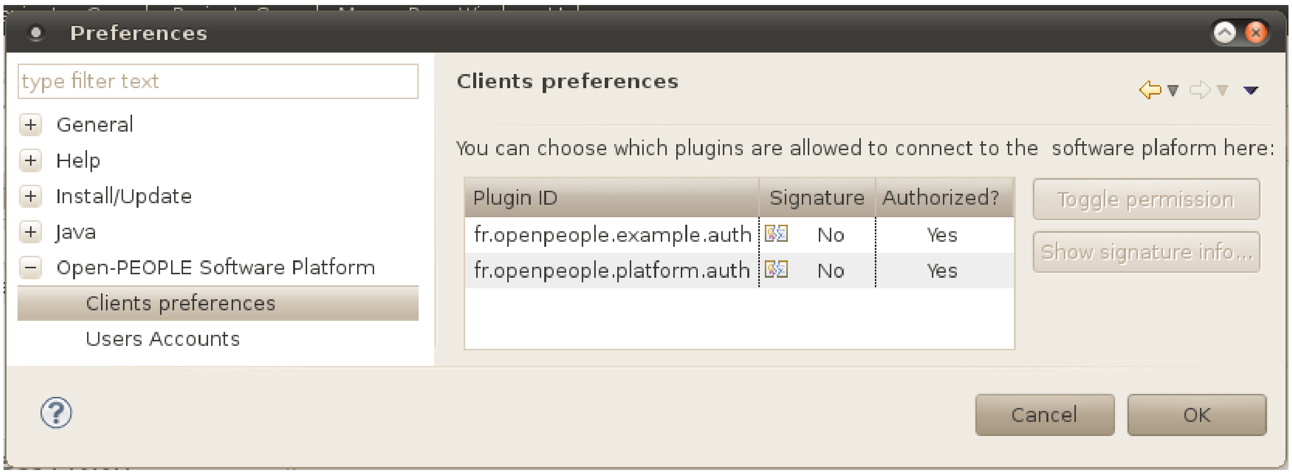


Users Accounts

You can add, edit and remove user accounts here:

ID	Login	Name	Powers
1	Admin	Mr. A. D. MIN	<ADM--PCAO--PCMD>

Figure 4.6: User account preference page



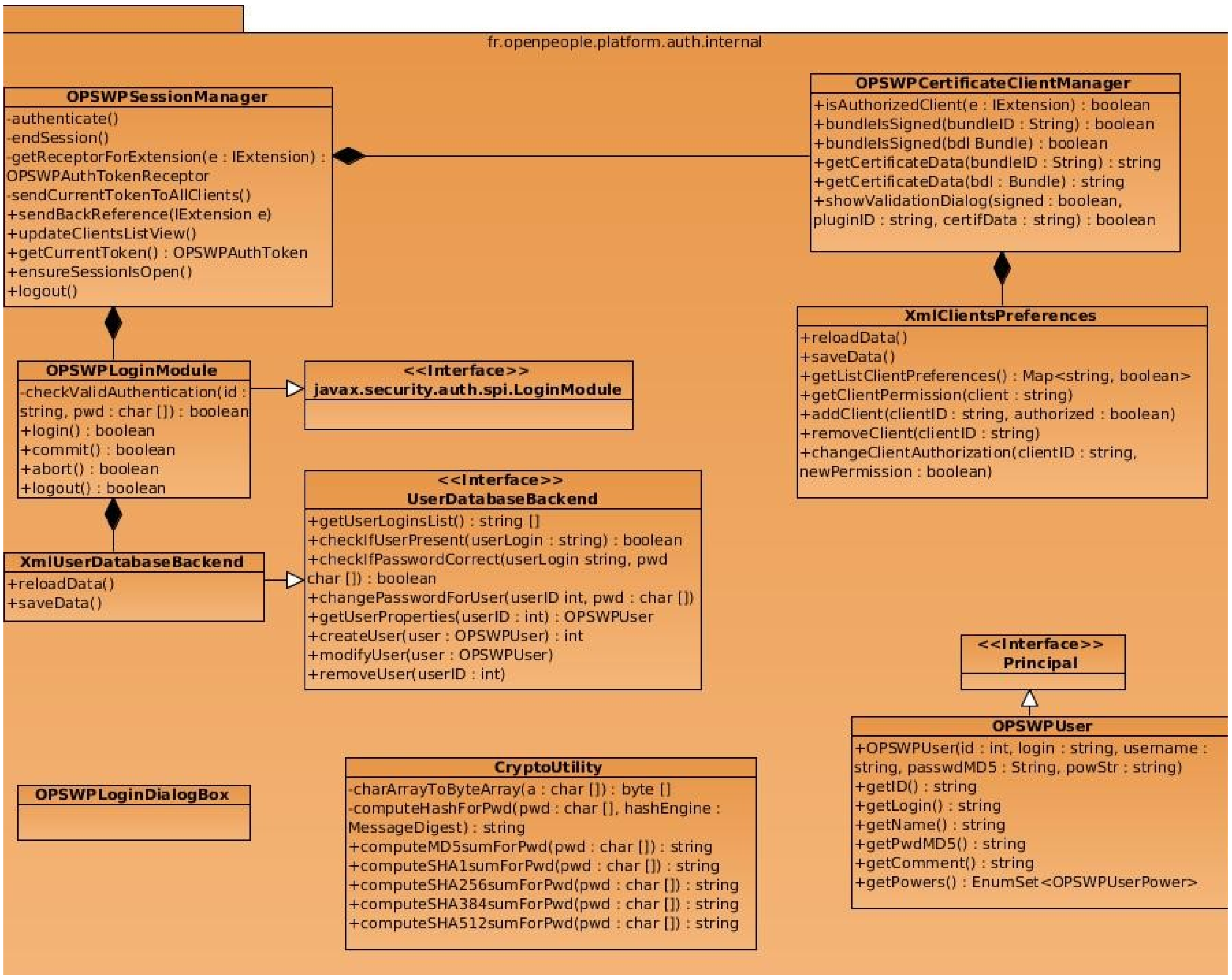
Clients preferences

You can choose which plugins are allowed to connect to the software plaform here:

Plugin ID	Signature	Authorized?
fr.openpeople.example.auth	No	Yes
fr.openpeople.platform.auth	No	Yes

Figure 4.7: Client plugins preference page

The `fr.openpeople.platform.auth.internal` package (figure 4.7) gathers the plugin internal mechanism classes; this includes: user accounts and client plugins authorizations management (based on JAAS technology) and serialization, plus cryptology-related tools.


fr.openpeople.platform.auth.internal
OPSWPSessionManager
-authenticate()
-endSession()
-getReceptorForExtension(e : IExtension) : OPSWPAuthTokenReceptor
-sendCurrentTokenToAllClients()
+sendBackReference(IExtension e)
+updateClientsListView()
+getCurrentToken() : OPSWPAuthToken
+ensureSessionIsOpen()
+logout()
OPSWPCertificateClientManager
+isAuthorizedClient(e : IExtension) : boolean
+bundleIsSigned(bundleID : String) : boolean
+bundleIsSigned(bdl Bundle) : boolean
+getCertificateData(bundleID : String) : string
+getCertificateData(bdl : Bundle) : string
+showValidationDialog(signed : boolean, pluginID : string, certifData : string) : boolean
XmlClientsPreferences
+reloadData()
+saveData()
+getListClientPreferences() : Map<string, boolean>
+getClientPermission(client : string)
+addClient(clientID : string, authorized : boolean)
+removeClient(clientID : string)
+changeClientAuthorization(clientID : string, newPermission : boolean)
OPSWPLoginModule
-checkValidAuthentication(id : string, pwd : char []) : boolean
+login() : boolean
+commit() : boolean
+abort() : boolean
+logout() : boolean
<<Interface>>
javax.security.auth.spi.LoginModule
<<Interface>>
UserDatabaseBackend
+getUserLoginsList() : string []
+checkIfUserPresent(userLogin : string) : boolean
+checkIfPasswordCorrect(userLogin string, pwd char []) : boolean
+changePasswordForUser(userID int, pwd : char [])
+getUserProperties(userID : int) : OPSWPUser
+createUser(user : OPSWPUser) : int
+modifyUser(user : OPSWPUser)
+removeUser(userID : int)
XmlUserDatabaseBackend
+reloadData()
+saveData()
<<Interface>>
Principal
OPSWPUser
+OPSWPUser(id : int, login : string, username : string, passwdMD5 : String, powStr : string)
+getID() : string
+getLogin() : string
+getName() : string
+getPwdMD5() : string
+getComment() : string
+getPowers() : EnumSet<OPSWPUserPower>
OPSWPLoginDialogBox
CryptoUtility
-charArrayToByteArray(a : char []) : byte []
-computeHashForPwd(pwd : char [], hashEngine : MessageDigest) : string
+computeMD5sumForPwd(pwd : char []) : string
+computeSHA1sumForPwd(pwd : char []) : string
+computeSHA256sumForPwd(pwd : char []) : string
+computeSHA384sumForPwd(pwd : char []) : string
+computeSHA512sumForPwd(pwd : char []) : string

We can simplify this package with the following concept schema (figure 4.8) :

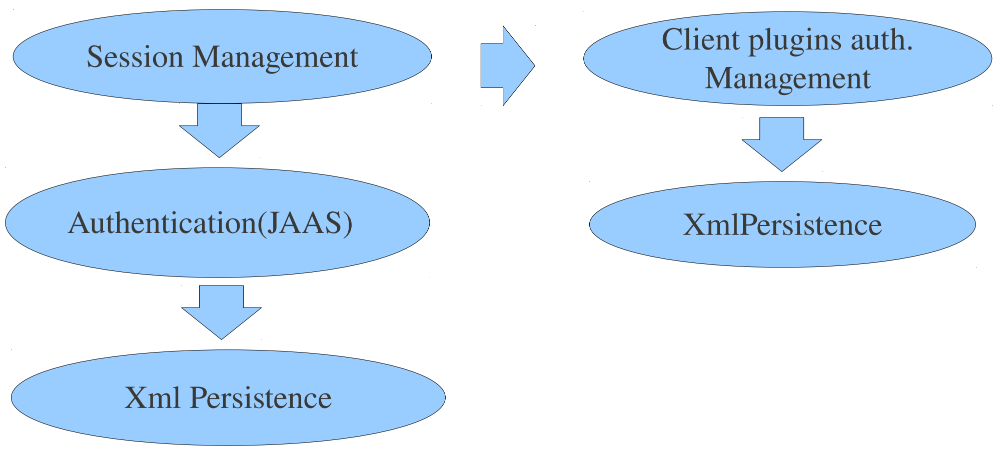


Figure 4.8: fr.openpeople.platform.auth.internal concept schema

We can thus consider 2 main functionalities for this package: session management (centered around the `OPSWPSessionManager` class) and client plugins authorizations management (centered around the `OPSWPCertificateClientManager` class).

Session managament mainly consists in an authentication system for the OPSWP. This authentication is based on Java's standard JAAS technology (the JAAS `LoginModule` interface being implemented by our `OPSWPLoginModule` class), while user accounts data is persisted by our own flexible mechanisms (`UserDatabaseBackend` interface, and the `XMLUserDatabaseBackend` implementing class). As soon as an user is authenticated, the session manager creates an adequate token, representing the current work session, and provides a reference on this token to all the registered client plugins.

Client plugins are allowed to receive authentication tokens if and only if they have the corresponding authorization; these authorizations are managed and checked by the `OPSWPCertificateClientManager` class. Authorizations are granted by OPSWP users each time a new client plugin try to use the authentication service. The user can at that moment choose to grant or deny authorization to the requesting plugin, after seeing this plugin's signature if it exists (since OPSWP plugins are specialized JARs, we make use of Java's JAR signing mechanism as a standardized way to authentify OPSWP plugins' contents and origins). These decisions are persisted into custom XML files via the `XmlClientPreferences` class.

`CryptoUtility` is a tool class that provides us with several convenient, easy-to-use cryptology-related methods (especially MD5 and SHA* hash computations) that we rely on to manage passwords. `OPSWPUser` is a model class, whose instances represent the OPSWP authenticated users.

4.3. Security

Security is an important aspect in the development of the OPSWP and its services. Concerning that the authentication plugin is a sensitive element , we employ several mechanisms to protect passwords and client plugins authorizations.

4.3.1. Passwords storage

The mechanism used to protect passwords storage (currently in an XML file) is a widely used one. We will thus explain it only briefly.

To avoid keeping passwords in clear-text form, we use cryptographic hash functions (example: MD5, SHA1, SHA-256, etc.) to obtain an unique “fingerprint” for each entered password. Algorithms of this kind are designed to provide a unique *digest* for every different possible input (bitstream) of any size; more precisely:

I. they allow to easily compute the digest for a given input, but it is (normally) impossible to regenerate the input from its computed hash;

II. moreover, it is (normally) impossible to obtain the same hash for two different inputs.

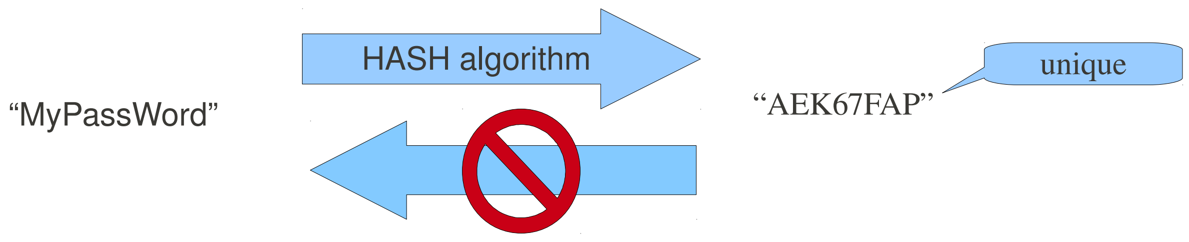


Figure 4.9: the main properties of cryptographic hash functions

We specifically take advantage of the two properties cited above (see also figure 4.9) to safely check that a user entered its expected password: if the hashes of an entered string (password) is equal to the hash of another stored string, then these two strings are necessarily equals.

So, instead of recording the passwords themselves, we record their hashes (we could say: their "fingerprints") in our users database. These hashes are easily computed thanks to our `fr.openpeople.platform.auth.internal.CryptoUtility` class.

4.3.2. Preferences security

Eclipse is an open system that stores its client preferences in clear-text form (generally within custom XML files). Even if it is impossible to totally secure such a system (protection of files is a function that ultimately depends on the operating system, not on applications), we introduced a mechanism to detect unexpected modifications of some of our configuration files (more precisely the file containing the client plugins authorizations); this mechanism can be summarized by figure 4.10 below:

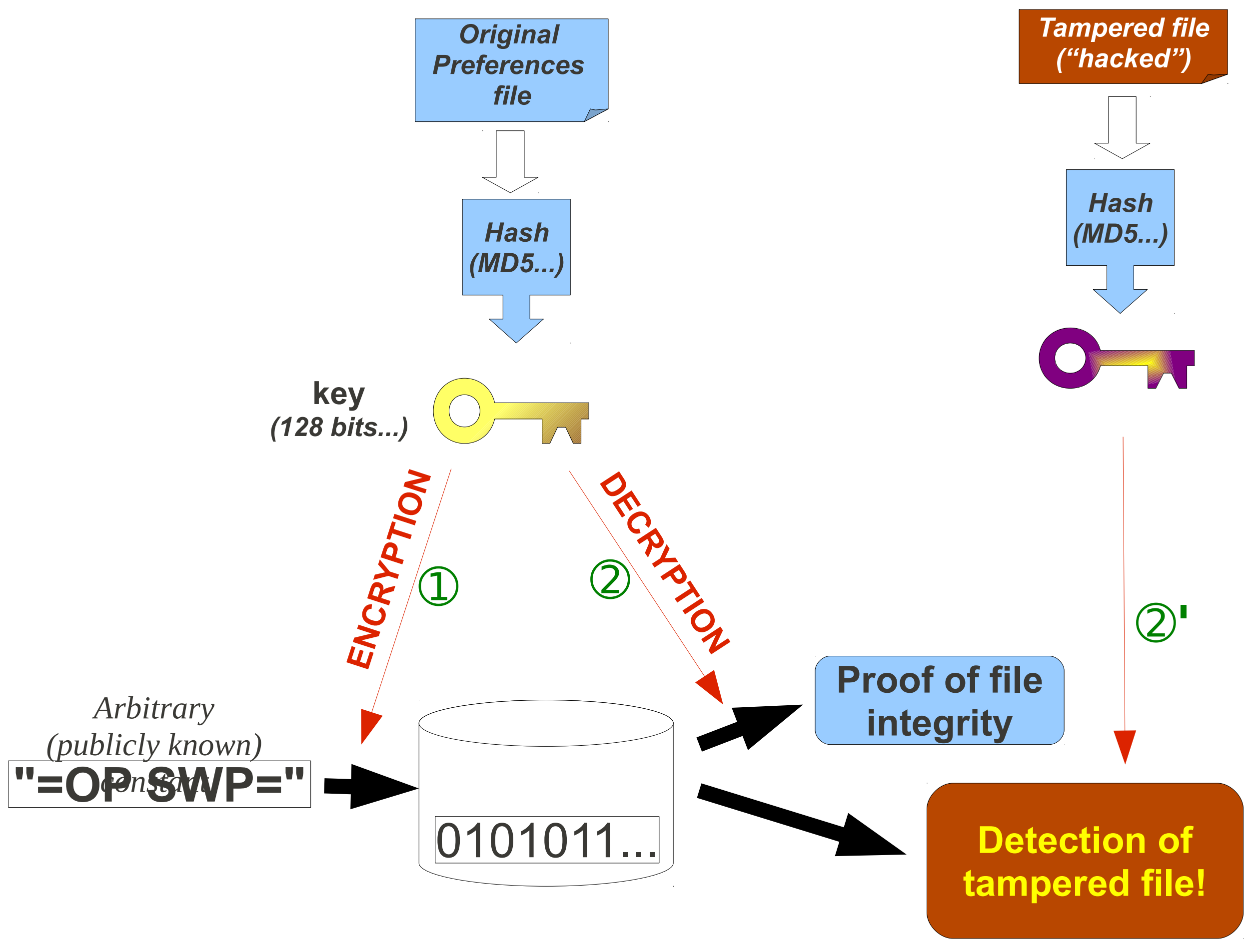


Figure 4.10: hacking detection mechanism

This mechanism uses once again cryptographic hash functions, this time coupled with a symmetric encryption algorithm, to detect easily and safely any tampering of our configuration file.

As soon as tampering of our authorizations file is detected, this file is reseted by the the OPSWP; then, users will be asked to grant or deny authorizations again to third party plugins once after another when they activate.

5. Look-Up Tables (LUT) Editor

This role of this editor is to provide an easy and efficient way to edit Look-Up Tables (LUT) that constitute the raw materials from which energy consumption models will be built in the OPSWP.

The editor allows to manage not only the LUTs metadata (for example: columns names and units), but also the LUTs data themselves, using a specifically built interface summarized hereafter (and further detailed in deliverable D2.8).

This editor is also packaged with an OPTR importation wizard, that allows to create LUTs directly from raw material provided by the OP HardWare Platform (OPHWP).

5.1. Architecture

The LUT editor is based on two different plugins:

- the `fr.openpeople.model.lutml` plugin, providing the LUTML model that is summarized in figure 5.1 hereafter:

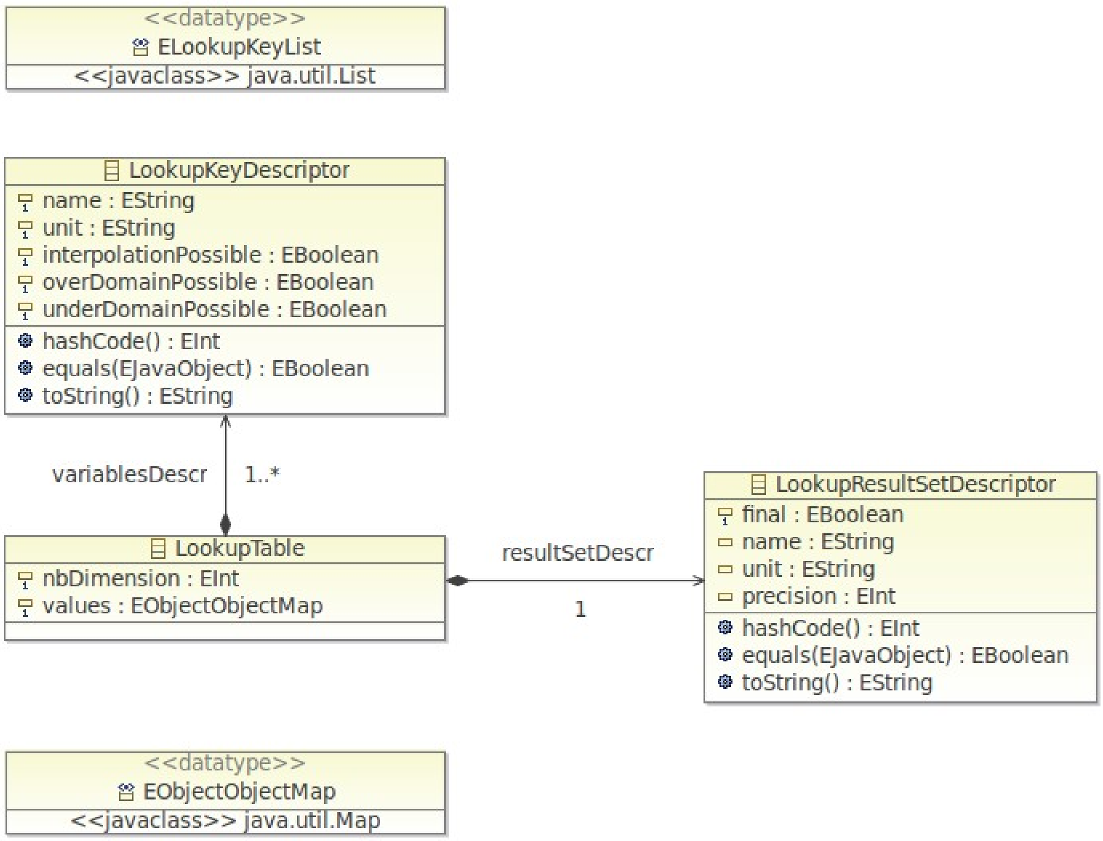


Figure 5.1: the LUTML model diagram

please refer to deliverable D4.1 for further details on this LUTML model

- the `fr.openpeople.ui.lutml` plugin, implementing and contributing the LUT editor itself – as well as the OPTR import wizard – to the Eclipse workbench: it is this plugin that we will now describe in the next section

5.2. The LUT editor plugin

This plugin, named `fr.openpeople.ui.lutml`, named contains only one Java package with the same name. The structure of this package is described in the figure hereafter:

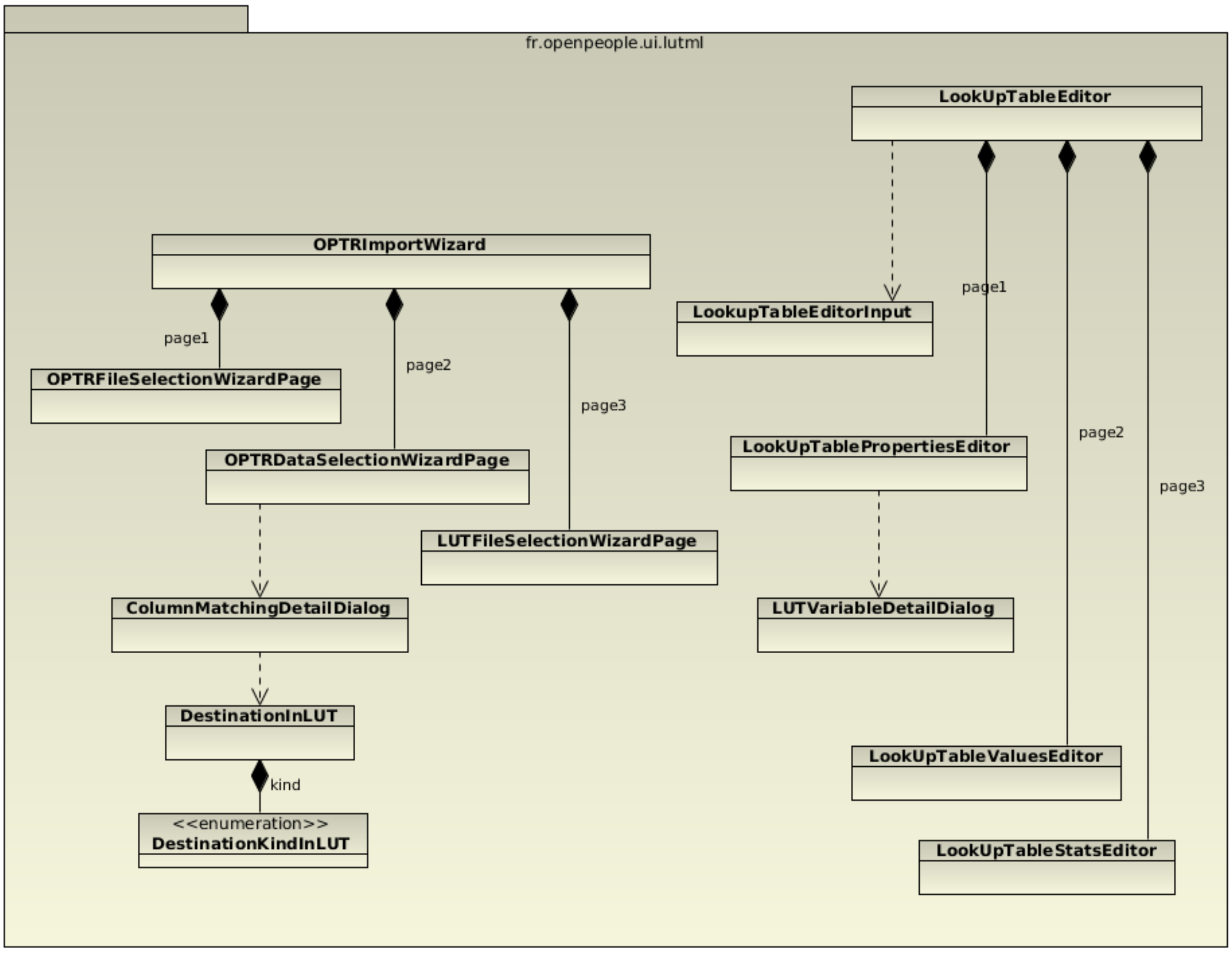
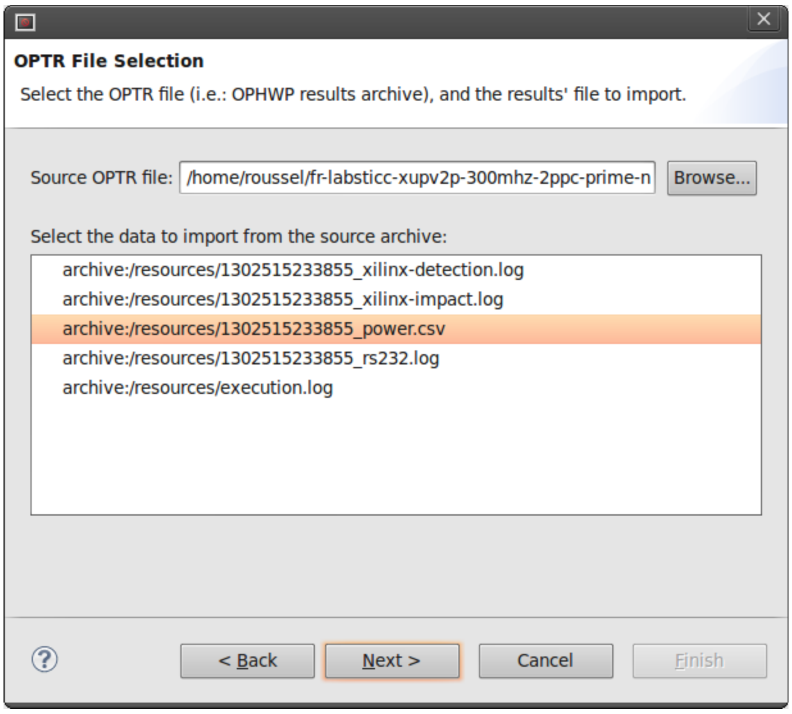


Figure 5.2: the `fr.openpeople.ui.lutml` *package*

As one can see, the package is subdivided into two parts, corresponding to the two main features it implements:

- the first part corresponds to the OPTR importation mechanism: the central class of this part is `OPTRImportWizard` that extends the standard Eclipse `Wizard` class; this wizard comprises three successive pages, that are respectively: `OPTRFileSelectionPage`, `OPTRDataSelectionPage`, and `LUTFileSelectionPage`; these three pages are shown respectively in the screenshots in figures 5.3, 5.4 and 5.5 below; the second page uses the `ColumnMatchingDetailDialog` class to edit the details corresponding to the way a column in the source file will be imported (see figure 5.6 for a screenshot), this dialog itself uses the `DestinationInLUT` as a model class, whose kind is represented by the `DestinationKindInLUT` enum;



OPTR File Selection

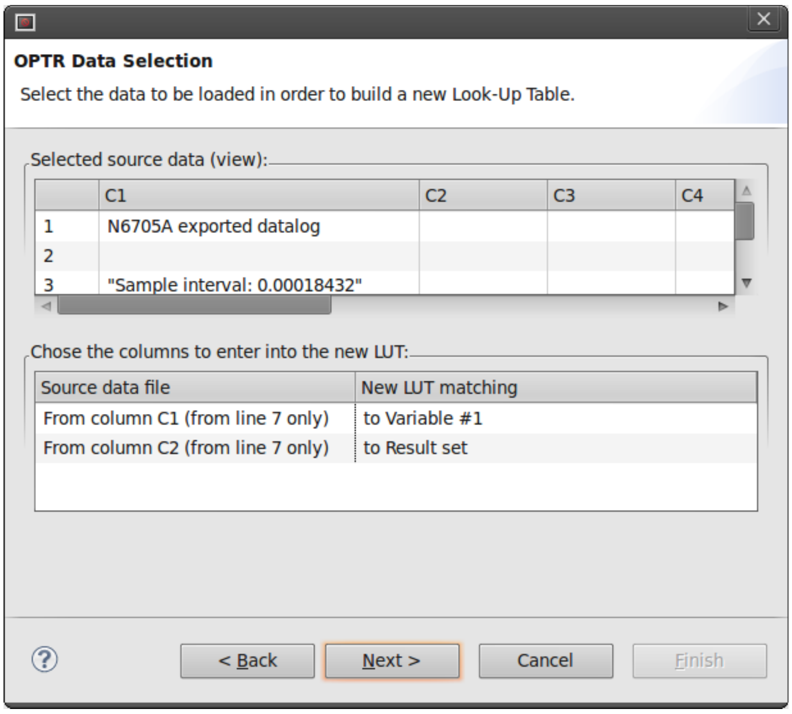
Select the OPTR file (i.e.: OPHWP results archive), and the results' file to import.

Source OPTR file: /home/roussel/fr-labsticc-xupv2p-300mhz-2ppc-prime-n

Select the data to import from the source archive:

```
archive:/resources/1302515233855_xilinx-detection.log
archive:/resources/1302515233855_xilinx-impact.log
archive:/resources/1302515233855_power.csv
archive:/resources/1302515233855_rs232.log
archive:/resources/execution.log
```

Figure 5.3: the first page of the OPTR import wizard



OPTR Data Selection

Select the data to be loaded in order to build a new Look-Up Table.

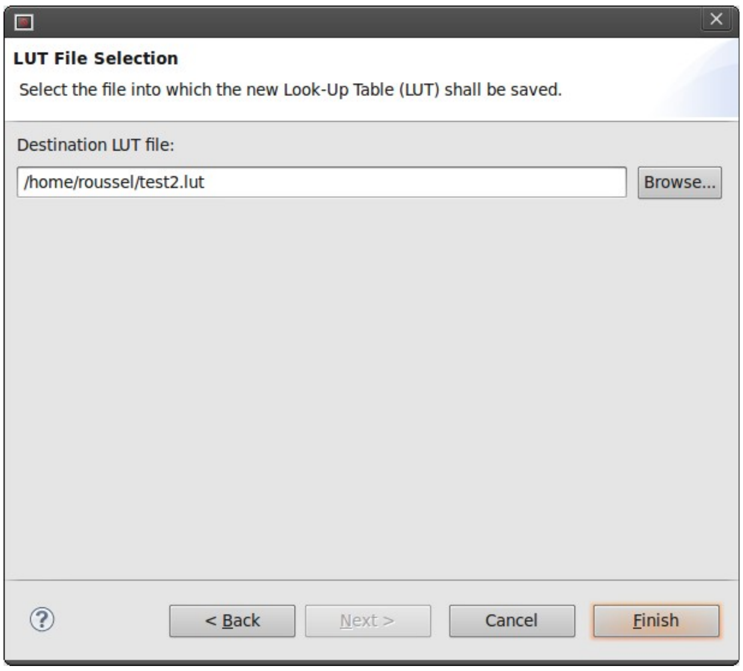
Selected source data (view):

	C1	C2	C3	C4
1	N6705A exported datalog			
2				
3	"Sample interval: 0.00018432"			

Chose the columns to enter into the new LUT:

Source data file	New LUT matching
From column C1 (from line 7 only)	to Variable #1
From column C2 (from line 7 only)	to Result set

Figure 5.4: the second page of the OPTR import wizard

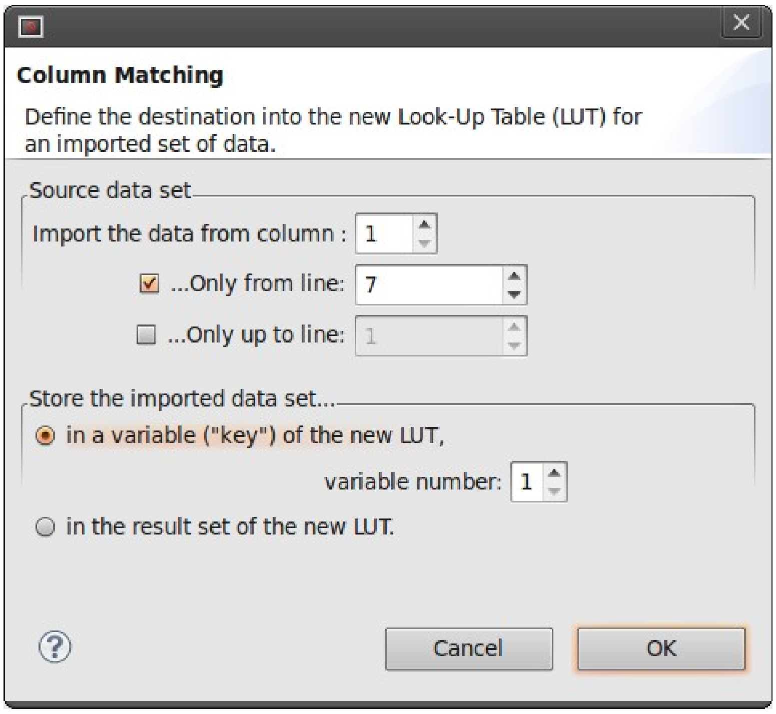


LUT File Selection

Select the file into which the new Look-Up Table (LUT) shall be saved.

Destination LUT file: /home/roussel/test2.lut

Figure 5.5: the third page of the OPTR import wizard



Column Matching

Define the destination into the new Look-Up Table (LUT) for an imported set of data.

Source data set

Import the data from column : 1

[✓] ...Only from line: 7

[ ] ...Only up to line: 1

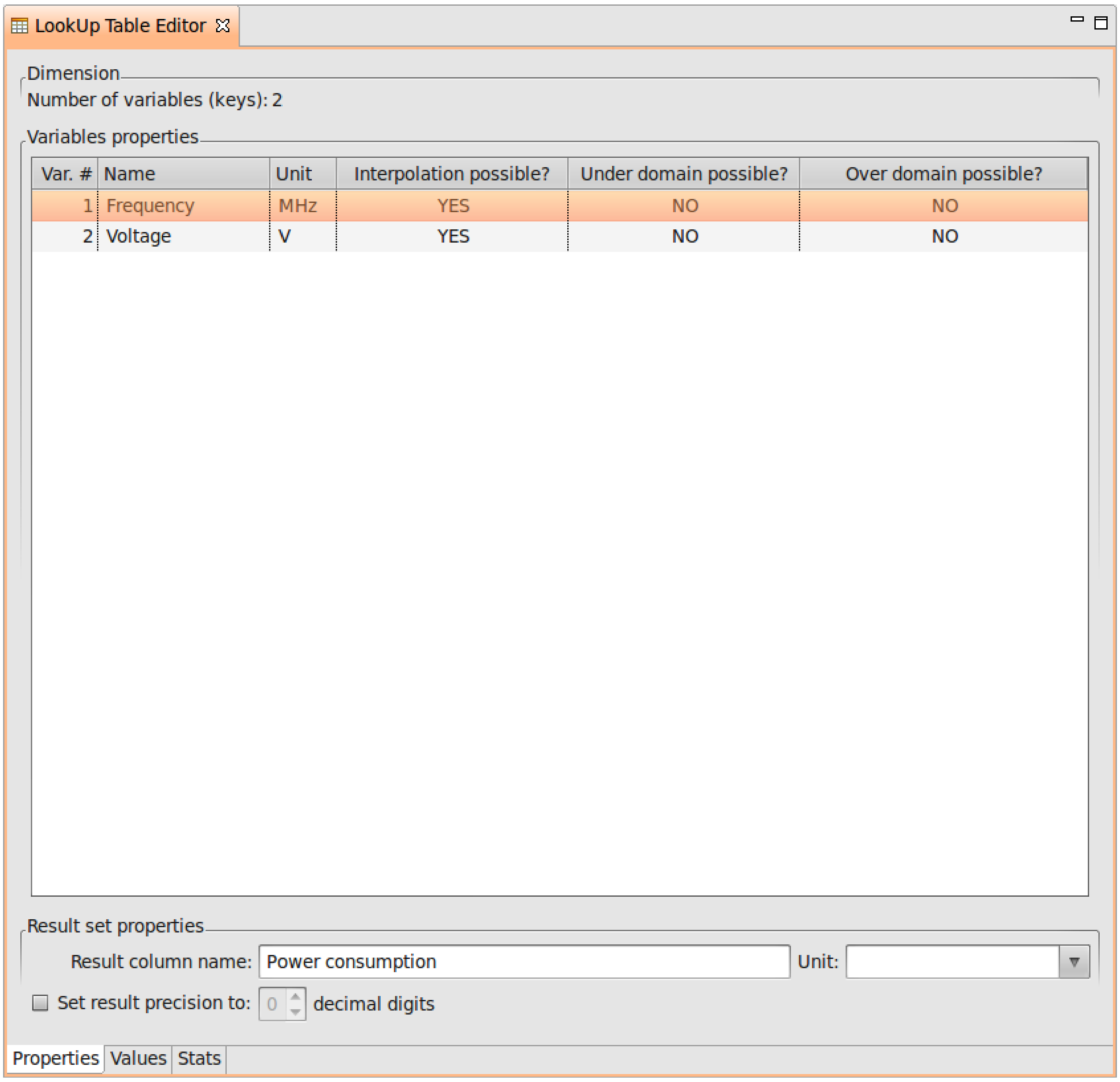
Store the imported data set...

[● in a variable ("key") of the new LUT,] variable number: 1

○ in the result set of the new LUT.

Figure 5.6: the column matching dialog box

- the second part corresponds to the LUT editor itself: the central class of this part is `LookupTableEditor` that extends the standard Eclipse `MultiPageEditorPart` class; this editor comprises three pages, that are: `LookupTablePropertiesEditor`, `LookupTableValuesEditor`, and `LookupStatsEditor`; these three pages are shown respectively in the screenshots in figures 5.7, 5.8 and 5.9 below; the first page uses the `LUTVariableDetailDialog` class to edit the details corresponding to a column of the currently edited LUT (see figure 5.10 for a screenshot); finally, since all Eclipse editors work on specific objects implementing the standard `IEditorInput` interface, a specific `LookupTableEditorInput` class is defined to encapsulate a LUT object as required.



Dimension

Number of variables (keys): 2

Variables properties

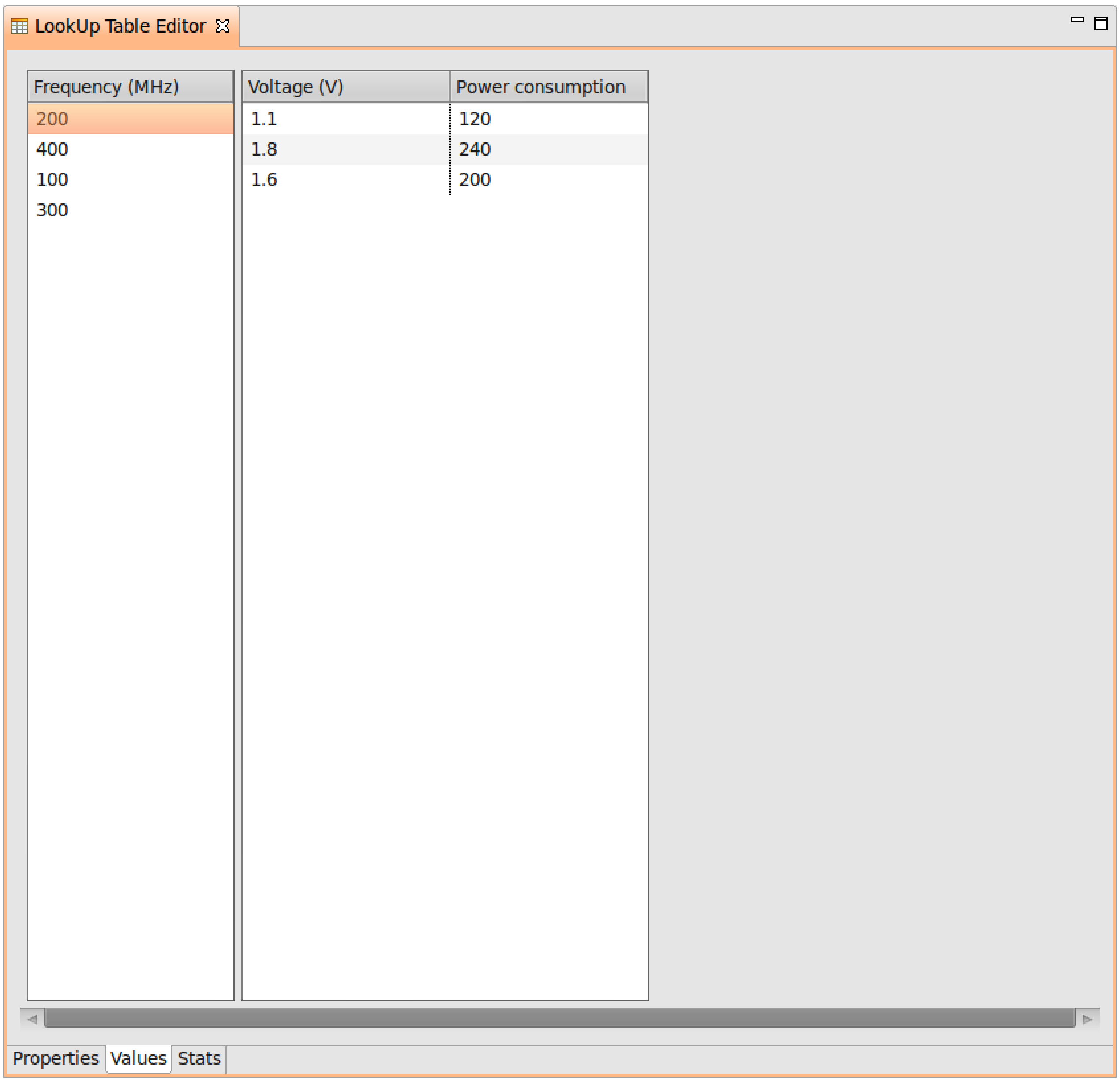
Var. #	Name	Unit	Interpolation possible?	Under domain possible?	Over domain possible?
1	Frequency	MHz	YES	NO	NO
2	Voltage	V	YES	NO	NO

Result set properties

Result column name: Power consumption Unit:

☐ Set result precision to: 0 decimal digits

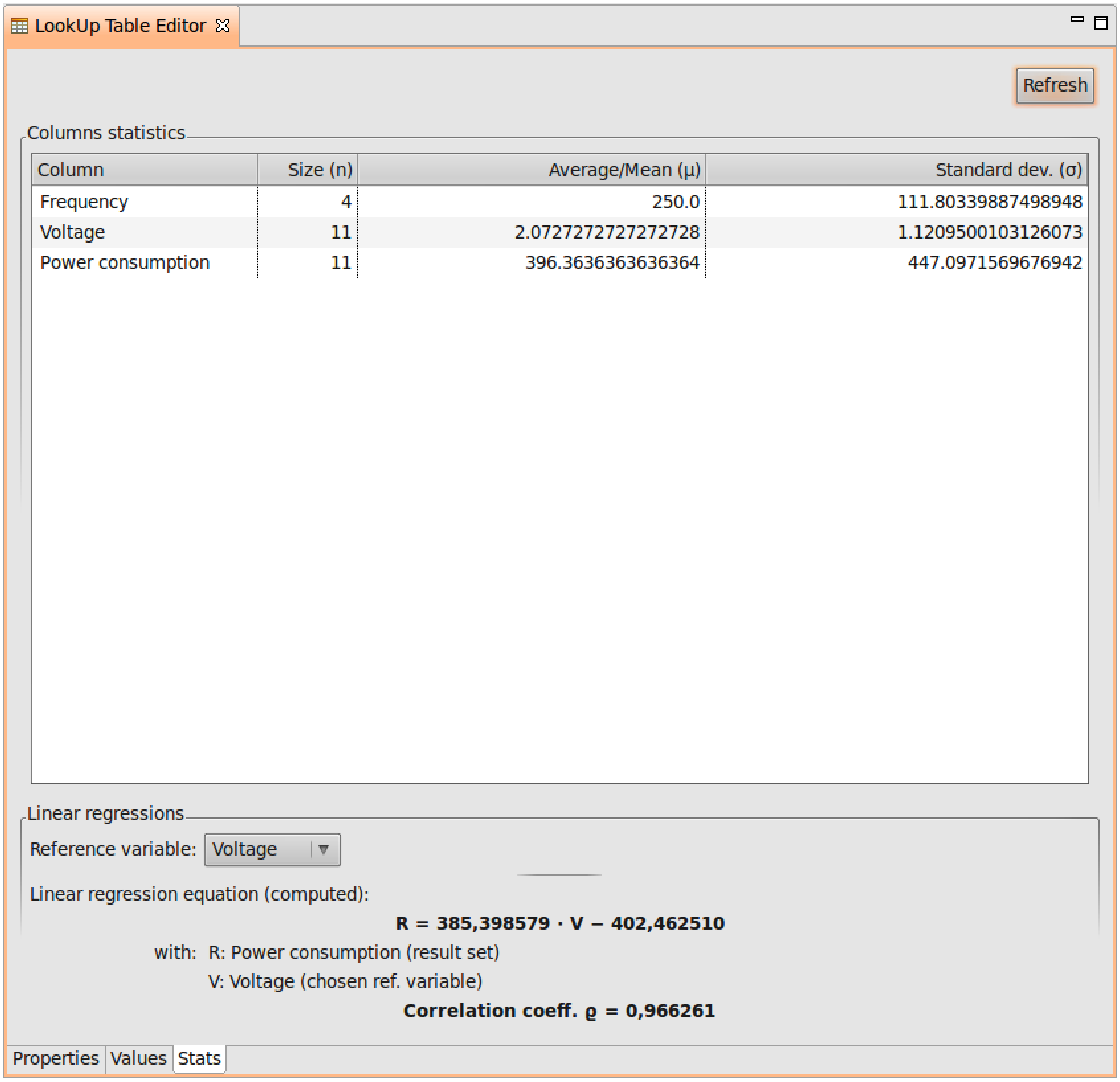
Figure 5.7: the first page of the LUT editor



Frequency (MHz)
200
400
100
300

Voltage (V)	Power consumption
1.1	120
1.8	240
1.6	200

Figure 5.8: the second page of the LUT editor



Columns statistics

Column	Size (n)	Average/Mean (μ)	Standard dev. (σ)
Frequency	4	250.0	111.80339887498948
Voltage	11	2.0727272727272728	1.1209500103126073
Power consumption	11	396.3636363636364	447.0971569676942

Linear regressions

Reference variable: Voltage

Linear regression equation (computed):

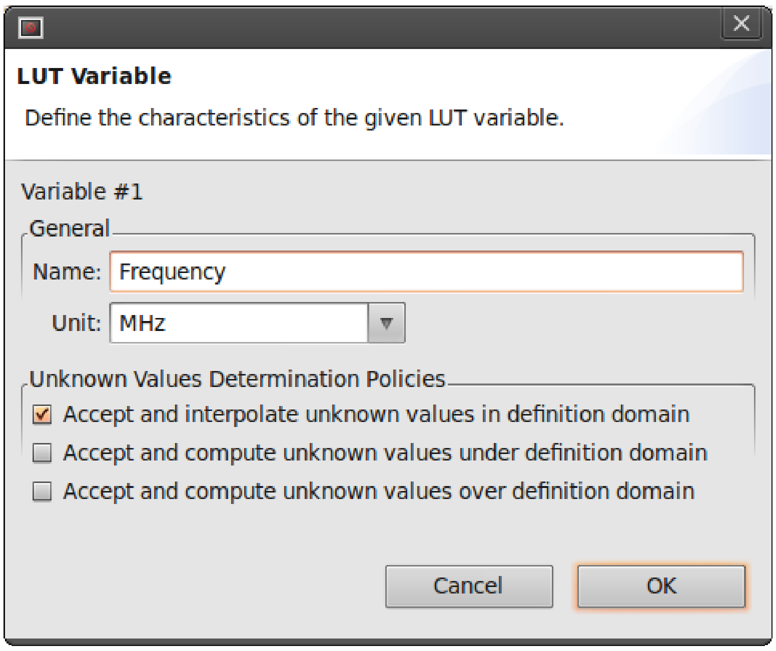
R = 385,398579 · V – 402,462510

with: R: Power consumption (result set)

V: Voltage (chosen ref. variable)

Correlation coeff. ρ = 0,966261

Figure 5.9: the third page of the LUT editor



LUT Variable

Define the characteristics of the given LUT variable.

Variable #1

General

Name: Frequency

Unit: MHz

Unknown Values Determination Policies

- [✓] Accept and interpolate unknown values in definition domain
- [ ] Accept and compute unknown values under definition domain
- [ ] Accept and compute unknown values over definition domain

Figure 5.10: the LUT variable dialog box

6. Quantities and Units (QUDV) Editor

This editor provides the management functions (creation, deletion, modification) of the quantities and units[1] used by the various consumption models manipulated by the OPSWP.

6.1. Architecture

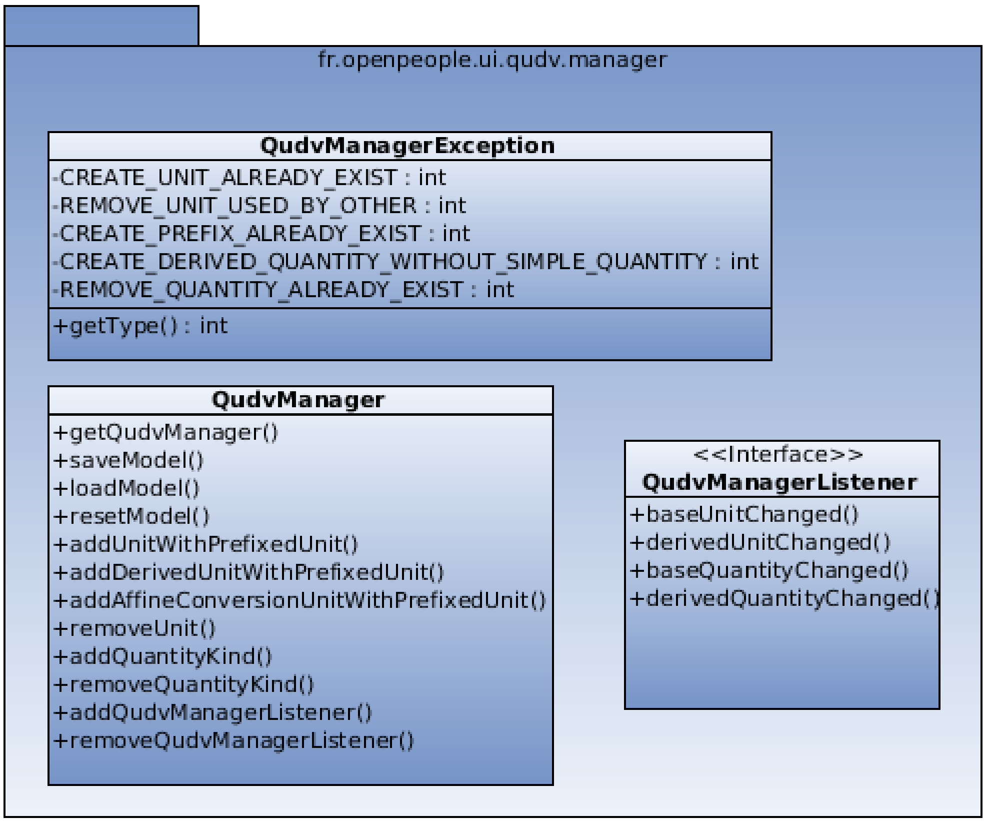
The architecture of this editor is based on Model-View-Controller (MVC) pattern.

The model use the Eclipse Modeling Framework (EMF) to persist its data, while a model manager class provides convenience methods. The model is provided by the `org.omgsysml.qudv` plugin, the model manager classes and its support classes and interfaces are in the `fr.openpeople.ui.qudv.manager` package.

The UI is divided among these packages:

- `fr.openpeople.ui.qudv.wizard`: wizard user interface
- `fr.openpeople.ui.qudv.preferences`: preferences user interface

6.2. Model manager


fr.openpeople.ui.qudv.manager
QudvManagerException
-CREATE_UNIT_ALREADY_EXIST : int
-REMOVE_UNIT_USED_BY_OTHER : int
-CREATE_PREFIX_ALREADY_EXIST : int
-CREATE_DERIVED_QUANTITY_WITHOUT_SIMPLE_QUANTITY : int
-REMOVE_QUANTITY_ALREADY_EXIST : int
+getType() : int
QudvManager
+getQudvManager()
+saveModel()
+loadModel()
+resetModel()
+addUnitWithPrefixedUnit()
+addDerivedUnitWithPrefixedUnit()
+addAffineConversionUnitWithPrefixedUnit()
+removeUnit()
+addQuantityKind()
+removeQuantityKind()
+addQudvManagerListener()
+removeQudvManagerListener()
<<Interface>>
QudvManagerListener
+baseUnitChanged()
+derivedUnitChanged()
+baseQuantityChanged()
+derivedQuantityChanged()

The `QudvManager` class provides utility functions that:

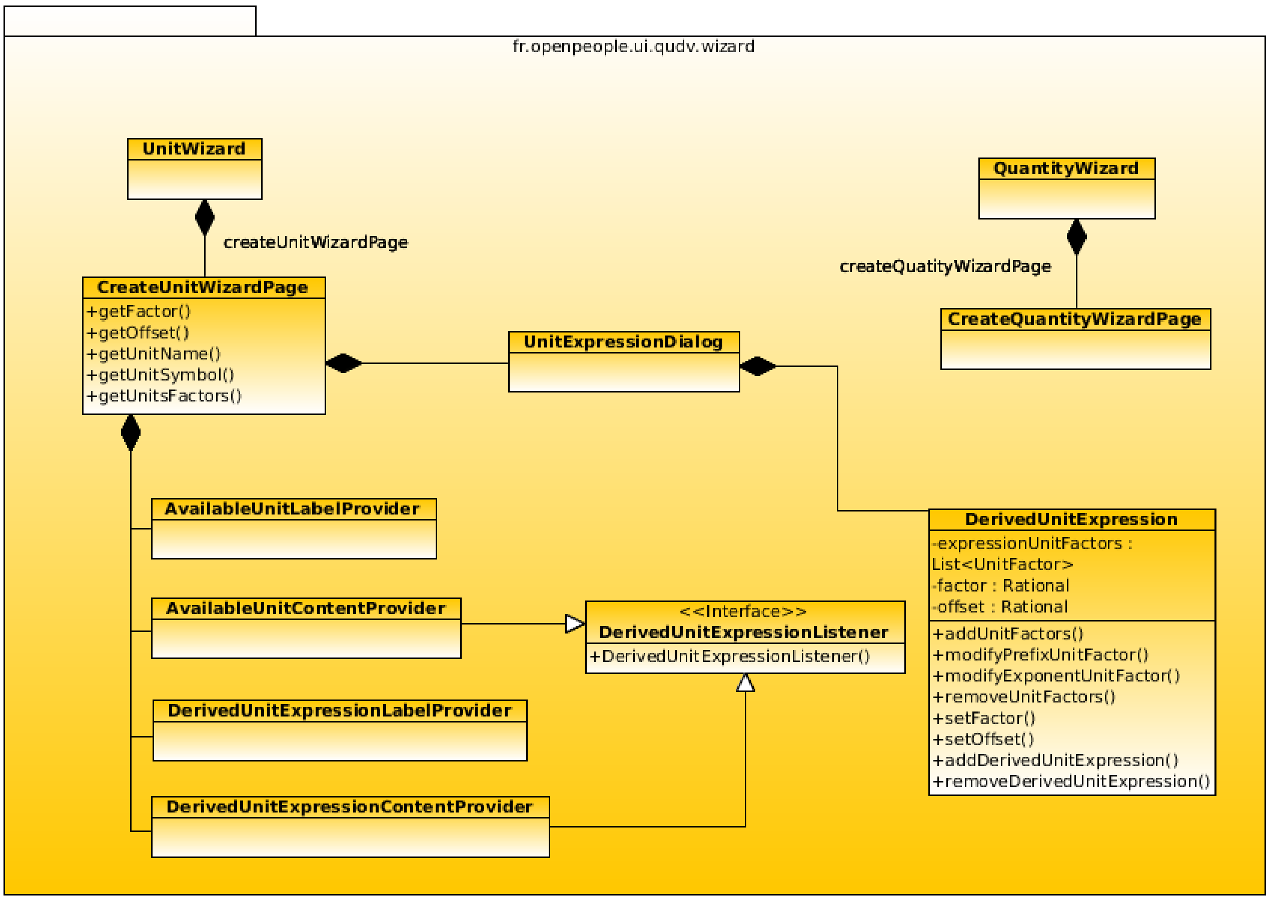
1 As a reminder, a quantity is the kind of a value (for example: a length, or a weight), whereas a unit is the reference used to quantify such a value (in our example: respectively the meter and the kilogram).

- add and remove Units and Quantities (QUDV elements),
- load and save the model.

It also sends messages to notify registered listeners when the state of a model change, thanks to `QudvManagerListener` interface.

Moreover, a new, specific kind of exception related to QUDV model manager (`QudvManagerException`) is defined.

6.3. Wizard user interface

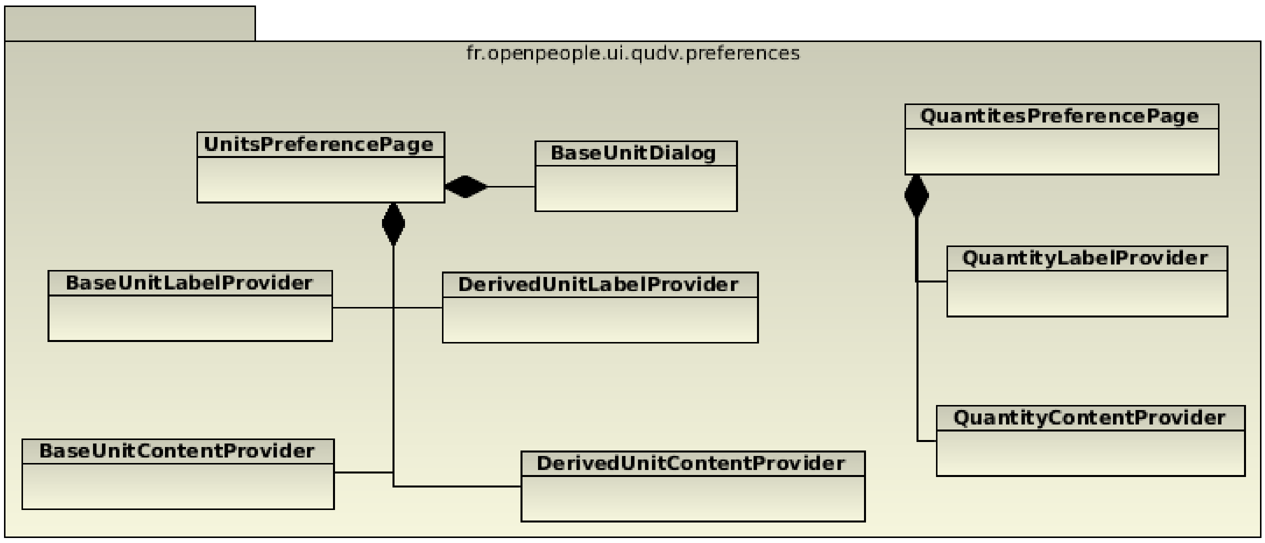

fr.openpeople.ui.qudv.wizard
UnitWizard
createUnitWizardPage
CreateUnitWizardPage
+getFactor()
+getOffset()
+getUnitName()
+getUnitSymbol()
+getUnitsFactors()
QuantityWizard
createQuatityWizardPage
CreateQuantityWizardPage
UnitExpressionDialog
AvailableUnitLabelProvider
AvailableUnitContentProvider
<<Interface>>
DerivedUnitExpressionListener
+DerivedUnitExpressionListener()
DerivedUnitExpressionLabelProvider
DerivedUnitExpressionContentProvider
DerivedUnitExpression
-expressionUnitFactors : List<UnitFactor>
-factor : Rational
-offset : Rational
+addUnitFactors()
+modifyPrefixUnitFactor()
+modifyExponentUnitFactor()
+removeUnitFactors()
+setFactor()
+setOffset()
+addDerivedUnitExpression()
+removeDerivedUnitExpression()

Two new creation wizards are contributed to the Eclipse workbench: a Unit creation wizard, and a Quantity creation wizard. These wizards both contain only one page (single-step wizards), respectively `CreateUnitWizardPage` and `CreateQuantityWizardPage`.

`DerivedUnitExpression` is a `UnitFactor` objects manager, used during the creation of derived units.

Finally, `AvailableUnitLabelProvider`, `AvailableUnitContentProvider`, `DerivedUnitExpressionLabelProvider` and `DerivedUnitExpressionContentProvider` are the support classes used by the JFace `TableViewer` control used within `CreateUnitWizardPage`.

6.4. Preferences user interface



We contribute two new preference pages, that show the list of all defined units and quantities, and allow to remove them as needed. These preference pages also contain `TableViewers`, for which ContentProvider and LabelProvider classes are defined. Finally, `BaseUnitDialog` is a creation dialog for `SimpleUnits` (i.e.: base units that are used to define `DerivedUnits` by combination).

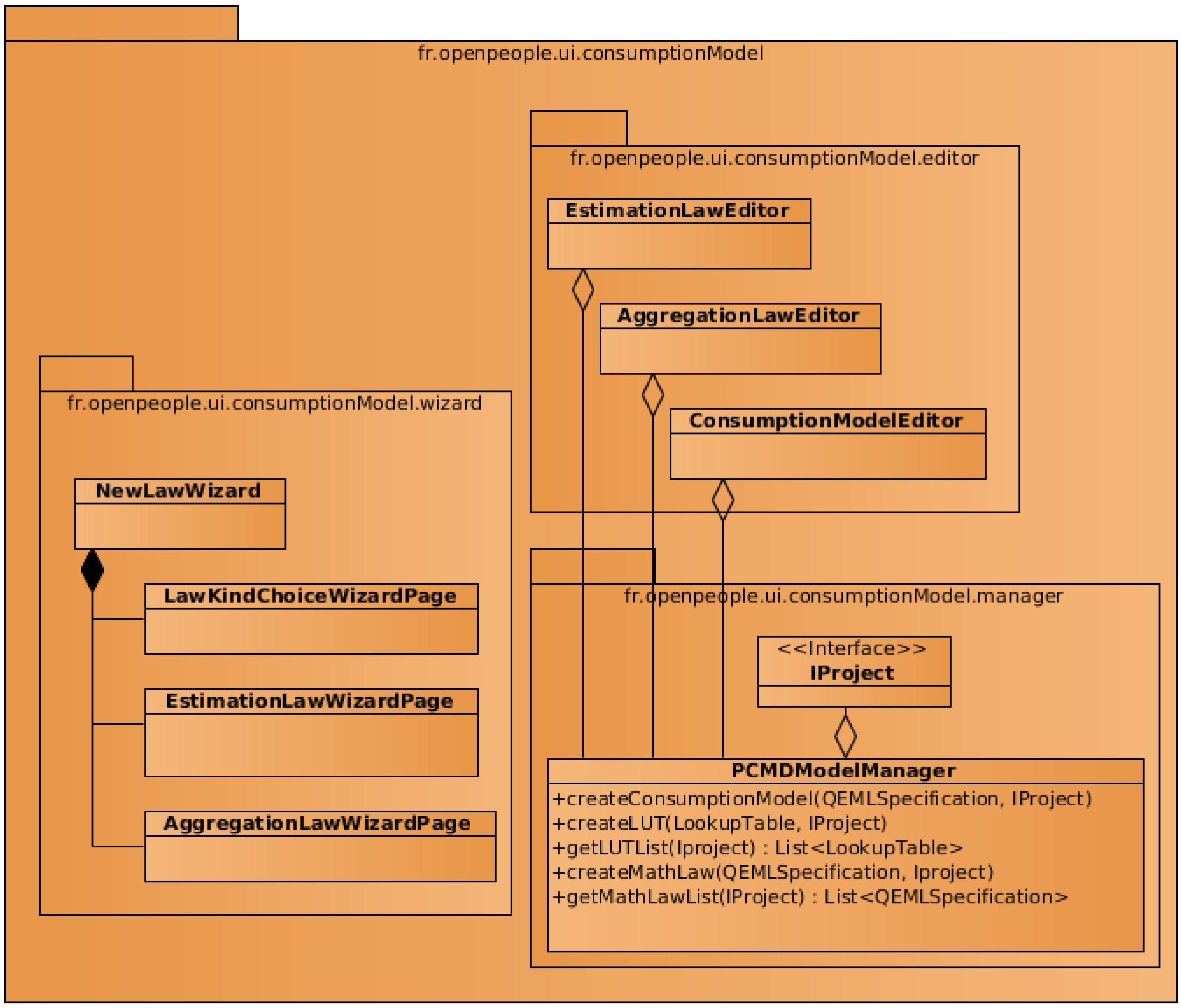
7. Model editors

These editors allow – among others – to create power consumption estimation models, that can be used directly on the local software platform or exported in remote databases.

7.1. Architecture

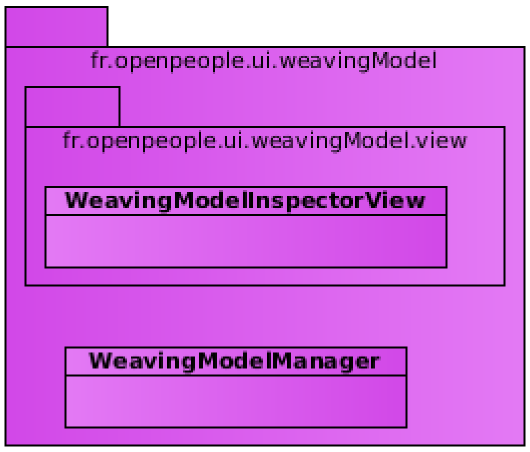
The architecture is, as usual, based on MVC paradigm. The `EstimationLawEditor`, `AggregationLawEditor`, and `ConsumptionModelEditor` classes, contained in the `fr.openpeople.ui.consumptionModel.editor` package, implement all the editors related to the PCMD features. These editors make use of an utility class, `PCMDModelManager`, that provides several useful functions, especially those related to models serialization.

The `NewLawWizard` class implements the creation wizard dedicated to mathematical estimation laws. This wizard contains three pages: `LawKindChoiceWizardPage`, `EstimationLawWizardPage`, and `AggregationLawWizardPage`: both of the two latter pages serve as alternatives for the second and last creation step, which varies according to the choice made during the first step (in `LawKindChoiceWizardPage`).


fr.openpeople.ui.consumptionModel
fr.openpeople.ui.consumptionModel.editor
EstimationLawEditor
AggregationLawEditor
ConsumptionModelEditor
fr.openpeople.ui.consumptionModel.wizard
NewLawWizard
LawKindChoiceWizardPage
EstimationLawWizardPage
AggregationLawWizardPage
fr.openpeople.ui.consumptionModel.manager
<<Interface>>
IProject
PCMDModelManager
+createConsumptionModel(QEMLSpecification, IProject)
+createLUT(LookupTable, IProject)
+getLUTList(Iproject) : List<LookupTable>
+createMathLaw(QEMLSpecification, Iproject)
+getMathLawList(IProject) : List<QEMLSpecification>

8. Weaving model edition

The user, thanks to the weaving model, can attach (“link”) a consumption model to each component of a system architecture. The WeavingModelManager class is the utility class dedicated to these weaving models, while the WeavingModelInspectorView class implements the view (in Eclipse workbench UI) that allows the user to perform the attachment operations.


fr.openpeople.ui.weavingModel
fr.openpeople.ui.weavingModel.view
WeavingModelInspectorView
WeavingModelManager

9. OPSWP global architecture

In this section, we briefly describe the OPSWP general architecture.

9.1. OPSWP design flow

Because the design of the OPSWP is based on the Model Driven Engineering (MDE) design methodology, the design flow is divided into different stages – each stage corresponding to a different kind of model – and every transition between two successive stages is actually a model transformation.

Each one of these model transformations consists in adding more and more details to the treated model:

The first part – processing of the Platform-Independent architecture Model (PIM) – consists in defining the application, and choosing the general architecture that will execute this application;

The second part (Deployment Model) consist in choosing the components (IP blocks) of the underlying architecture;

The third part (Estimated Model) consist in associating (“weaving”) each consumption model of the application with the related chosen component of architecture.

The last part finally consist in translating the obtained generic model in any needed third-party model (VHDL, SystemC, etc).

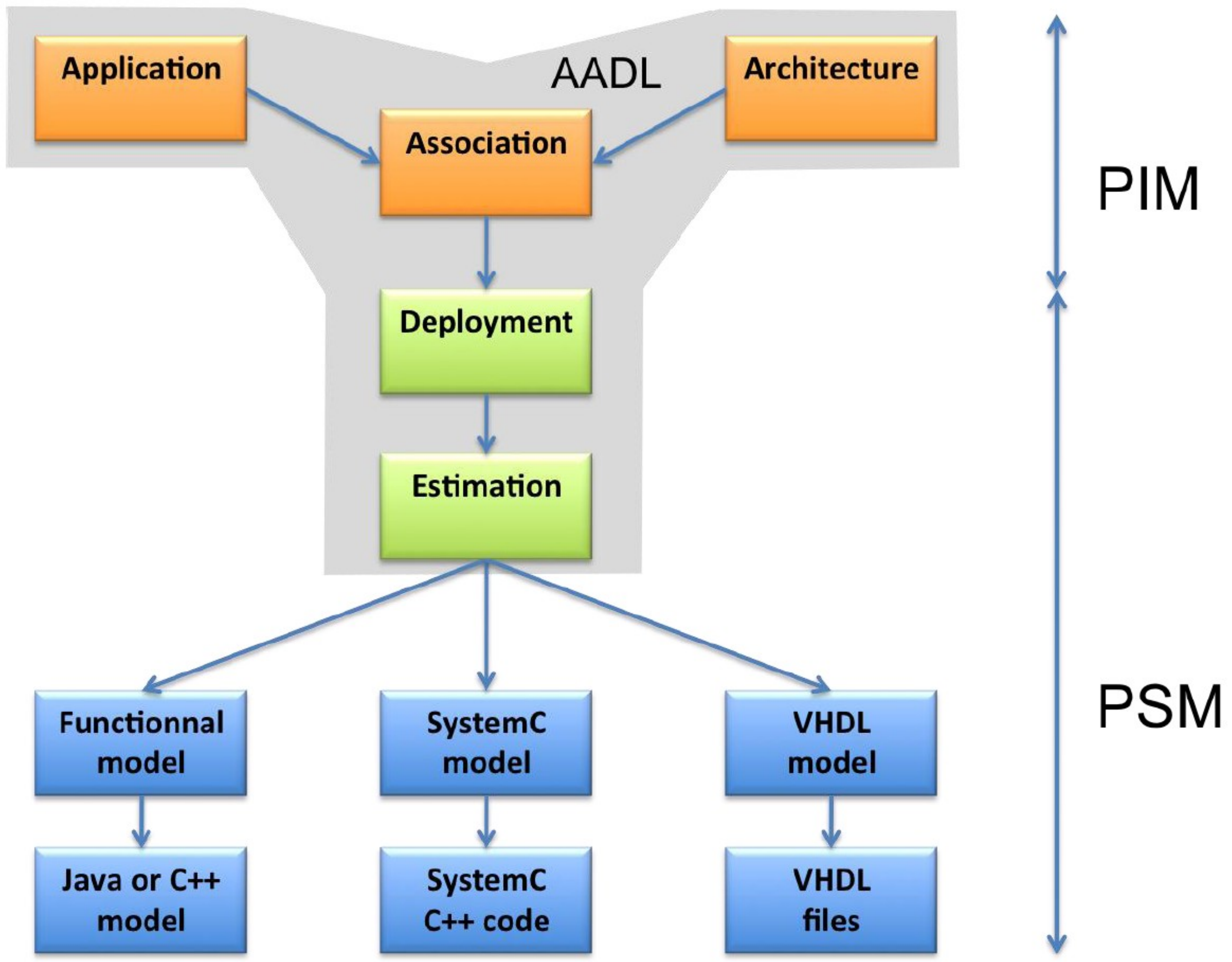


Figure 9.1: The design flow for multi-target platform power estimation in the Open-PEOPLE project

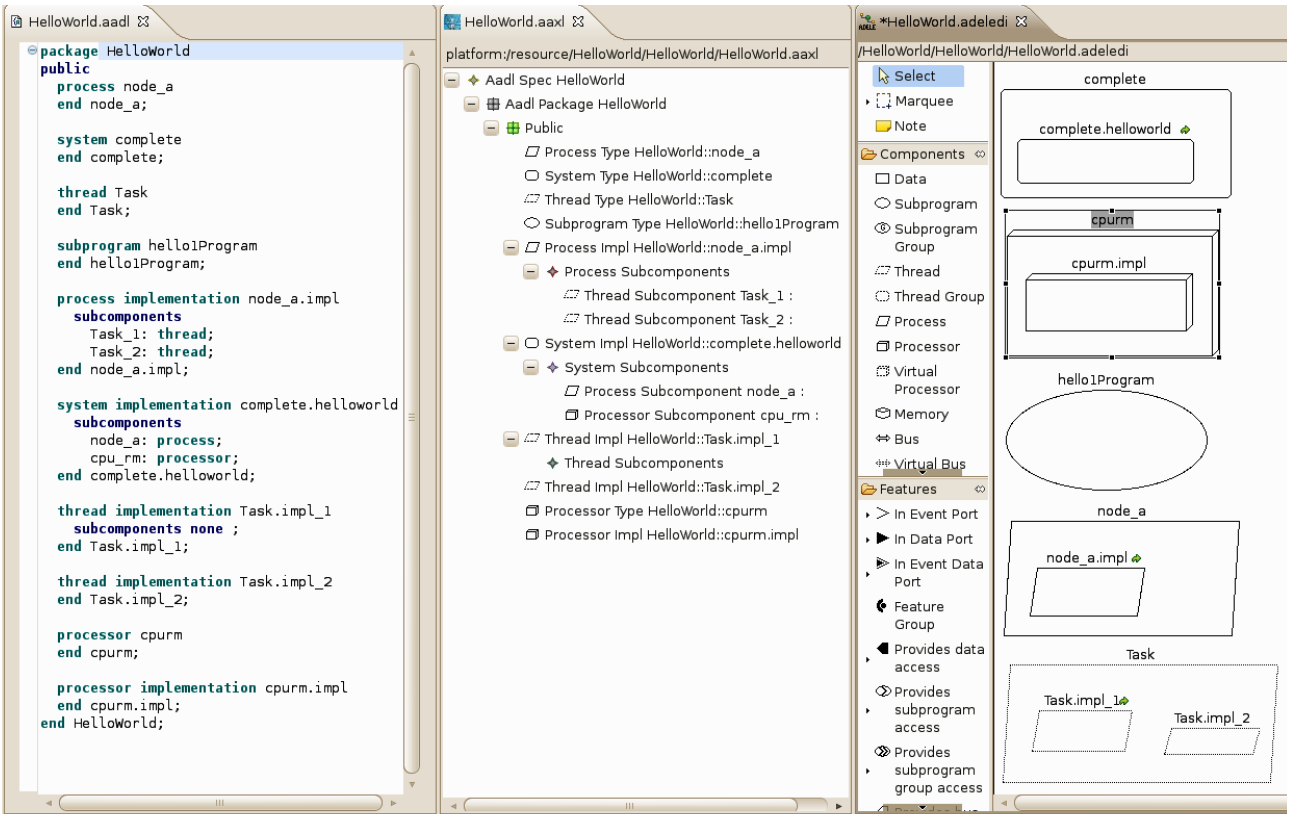
9.2. Application, architecture and association

As we said before, this part consists in defining the application, the underlying system architecture, and associating both of them. This, in the OPSWP, is to be done in the reference platform language: AADL.

There are three available ways to build and edit the AADL description of an architecture:

- a purely textual way, by editing directly the AADL source file;
- an object-oriented way, using a standard, tree-based Eclipse editor;
- and a graphical way, using a dedicated graphical editor.

In the picture hereafter, we can see the three possible representations of the same AADL architecture: textual on the left, object-oriented in the middle, and graphical on the right.



HelloWorld.aadl

```
package HelloWorld
public
  process node_a
  end node_a;

  system complete
  end complete;

  thread Task
  end Task;

  subprogram hello1Program
  end hello1Program;

  process implementation node_a.impl
    subcomponents
      Task_1: thread;
      Task_2: thread;
  end node_a.impl;

  system implementation complete.helloworld
    subcomponents
      node_a: process;
      cpu_rm: processor;
  end complete.helloworld;

  thread implementation Task.impl_1
    subcomponents none ;
  end Task.impl_1;

  thread implementation Task.impl_2
  end Task.impl_2;

  processor cpurm
  end cpurm;

  processor implementation cpurm.impl
  end cpurm.impl;
end HelloWorld;
```

HelloWorld.aaxl

platform:/resource/HelloWorld/HelloWorld/HelloWorld.aaxl

```
Aadl Spec HelloWorld
  Aadl Package HelloWorld
    Public
      Process Type HelloWorld::node_a
      System Type HelloWorld::complete
      Thread Type HelloWorld::Task
      Subprogram Type HelloWorld::hello1Program
      Process Impl HelloWorld::node_a.impl
        Process Subcomponents
          Thread Subcomponent Task_1 :
          Thread Subcomponent Task_2 :
      System Impl HelloWorld::complete.helloworld
        System Subcomponents
          Process Subcomponent node_a :
          Processor Subcomponent cpu_rm :
      Thread Impl HelloWorld::Task.impl_1
        Thread Subcomponents
      Thread Impl HelloWorld::Task.impl_2
      Processor Type HelloWorld::cpurm
      Processor Impl HelloWorld::cpurm.impl
```

Tools allowing to edit AADL representations, using these different ways, already exist. Thus, Osate will provide us with textual (source) and object-oriented ADDL editors; whereas the Topcased Adele project offers a graphical AADL editor. Note, however, that Adele is not ready for production use yet (as of summer 2011), because its ergonomics are vastly perfectible, and – especially! – because we can't create a graphical diagram directly by importing a pre-existing AADL source file.