|  |
| --- |
| CEA LIST |
| Generation of architecture from plug-in sets |
| [Tapez le sous-titre du document] |

|  |
| --- |
| TESSIER Patrick  19/11/2013 |



Sommaire

[Introduction 3](#_Toc372642750)

[Requirements 3](#_Toc372642751)

[Use Cases 4](#_Toc372642752)

[1. Import plug-ins 4](#_Toc372642753)

[2. Import Features 6](#_Toc372642754)

[3. AdvanceImportPlugins 6](#_Toc372642755)

[4. GeneratePlugins 7](#_Toc372642756)

[5. GenerateFeatures 7](#_Toc372642757)

[Solution proposal 8](#_Toc372642758)

[6. The plug-in org.eclipse.papyrus.osgi.profile 9](#_Toc372642759)

[7. The plug-in org.eclipse.papyrus.adl4eclipse.profile 12](#_Toc372642760)

[8. The plug-in org.eclipse.papyrus.adl4eclipsetool 16](#_Toc372642761)

[9. The plug-in org.eclipse.papyrus.adl4eclipsetool.assistant 16](#_Toc372642762)

[Application on Papyrus 17](#_Toc372642763)

[Perspectives 17](#_Toc372642764)

[References 17](#_Toc372642765)

# Introduction

This document present the tool used to manage Eclipse plug-ins and features from a UML based Model.

This document is a developer document. It is divided into several parts:

* Requirements
* Use cases
* Design of the tool.
* An application to Papyrus.

# Requirements

**Req001: PluginsRetroEngineering**  
 Papyrus shall help to retro engineer plug-ins structure: name plug-in properties and dependencies.

**Req002: PluginStructure**  
 Papyrus shall help to import only dependencies of chosen plug-ins or all dependencies by recursion

**Req003: ExtensionPointDependency**Papyrus shall model extension points of plug-ins as used in eclipse.

**Req004: FeatureModel**  
 Papyrus shall help to model feature elements (eclipse context)

**Req005: PluginGeneration**  
Papyrus shall generate plug-ins from the model

**Req006: FeatureGeneration**   
Papyrus shall generate Features element from the model

# Use Cases

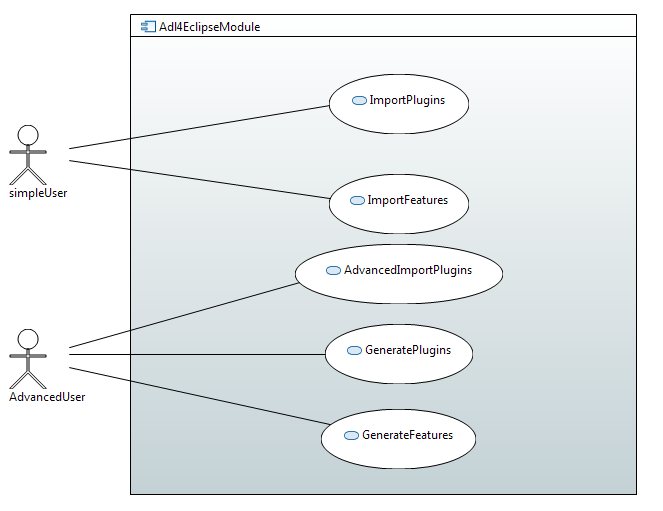


Figure 1: UseCases

## Import plug-ins

This use case refines the *Req\_001*, *Req\_002* and *Req\_003.*

The purpose of this functionality is to allow a user to model his architecture from a set of plug-ins. The model must contain:

* Plug-ins,
* Extension points defined in eclipse
* Dependencies to other plug-ins.

#### How to:

Typically, the eclipse user has got a set of plug-ins (java code and configuration files) in this workspace.

To model the set of plug-ins, first the user creates a model and its own structure in which he want to model the architecture.

Then the user runs the eclipse menu imports (Figure 2). And choose the tool “Simple Plug-in architecture snapshot” (Figure 3).

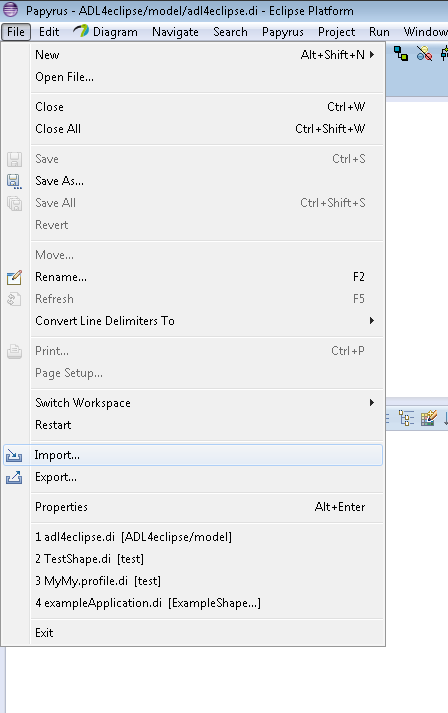


Figure 2: Import Menu

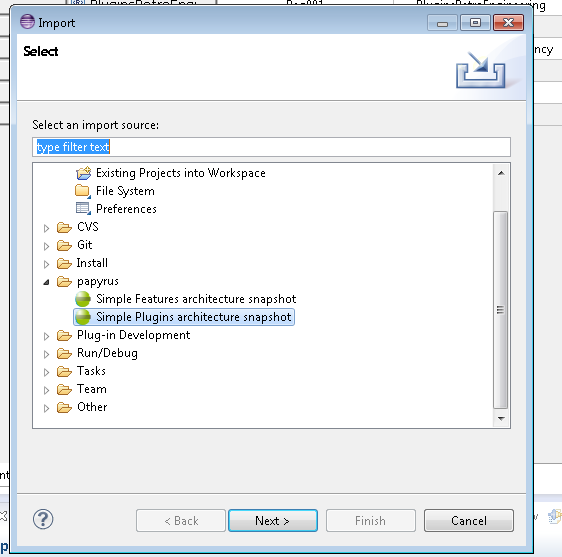


Figure 3: Tool to model the plug-in architecture

The import tool presents all possible plug-ins contained in the workspace that can be modeled. The user selects the subset of plug-ins and run the model construction (Figure 4).

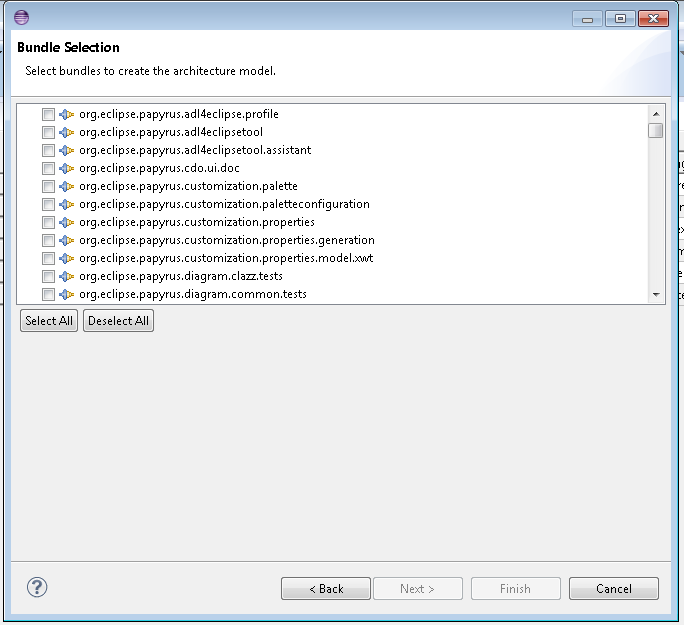


Figure 4: Plug-in selection

## Import Features

This use case refines the *Req\_004*.

The purpose of this functionality is to allow a user to model the feature [1] as a composite structure.

The feature is represented a structured composite that contained parts that references modeled plug-ins.

This use case includes the import Plug-in use case.

#### How to:

This use case is equivalent has the “import plug-in” applied for plug-ins.

The user creates a Papyrus model, select a package. The user selects import features from the import menu.

The tool proposes the set of feature that can be modeled. The user has only to select wanted feature and all feature and referenced plug-ins are modeled.

## AdvanceImportPlugins

This use case refines the *Req\_001*, *Req\_002* and *Req\_003*.

The functionality is an extension of the use case plug-in importation. But in this context, the user can set the deep about dependencies.

Thanks to this functionality, it is possible to model all plug-ins of an eclipse platform.

Because all plug-ins of a platform can be modeled, a role of a advanced user is associated to this use case.

#### How to:

To model the set of plug-ins, the user runs the eclipse menu imports. The import tool presents all possible plug-ins contained in the workspace and in the eclipse platform.

The user selects the subset of plug-ins and runs the model construction. He can also choose the deep of dependencies recursion. The model construction can be takes several minutes because of the big number of plug-ins and recursions.

## GeneratePlugins

This use case refines the *Req\_005*

The purpose of this functionality is to provide means to generate or update plug-ins from a model.

This use case is very interesting into to generate new plug-ins or to adapt existing plug-ins.

For example, changing the version number of Papyrus can be a model transformation then a generation than launch the execution of a script that look for all interesting files.

This use-cases is not yet implemented

## GenerateFeatures

This use case refines the *Req\_006*.

The purpose of this functionality is to generate feature from the model.

Be able to manage feature in eclipse is important to allow the deployment of a tool. In the case of Papyrus, that must be modular, managing feature is very important.

Be able to adapt the feature by taking account it modularity becomes easier by visualizing the feature model.

This use-cases is not yet implemented

# Design

In this part, the architecture of the tool is shown. This model has been obtained by running the use cases described before.

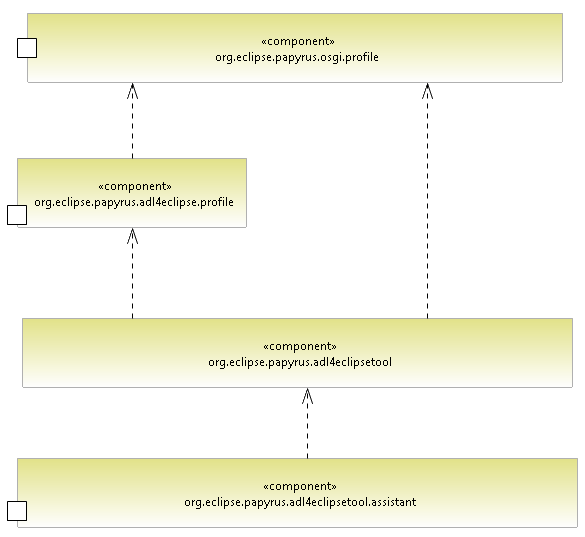


Figure 5: ADL4Eclipse architecture

This tool is divided into 4 plug-ins:

* Two profiles contained in each different plug-in: org.eclipse.papyrus.osgi.profile and org.eclipse.papyrus.adl4eclipse.profile.
* A plug-in that contains algorithm to able to execute the construction of the model.
* A plug-in that is the graphical interface to run these model algorithms.

## The plug-in org.eclipse.papyrus.osgi.profile

Eclipse is an OSGI environment called Equinox. Each plug-in in Eclipse is an extension of a bundle of OSGI [2].

For this reason, a subset of OSGI element has been modeled as a profiled. Only subset of OSGI element that was interesting for our models has been including in the profile. In our case, the module layer has been only added in the profile.

So security and life cycle layer are not present in this profile. Nevertheless, this element could be added if needed.

The profile contains 5 stereotypes.

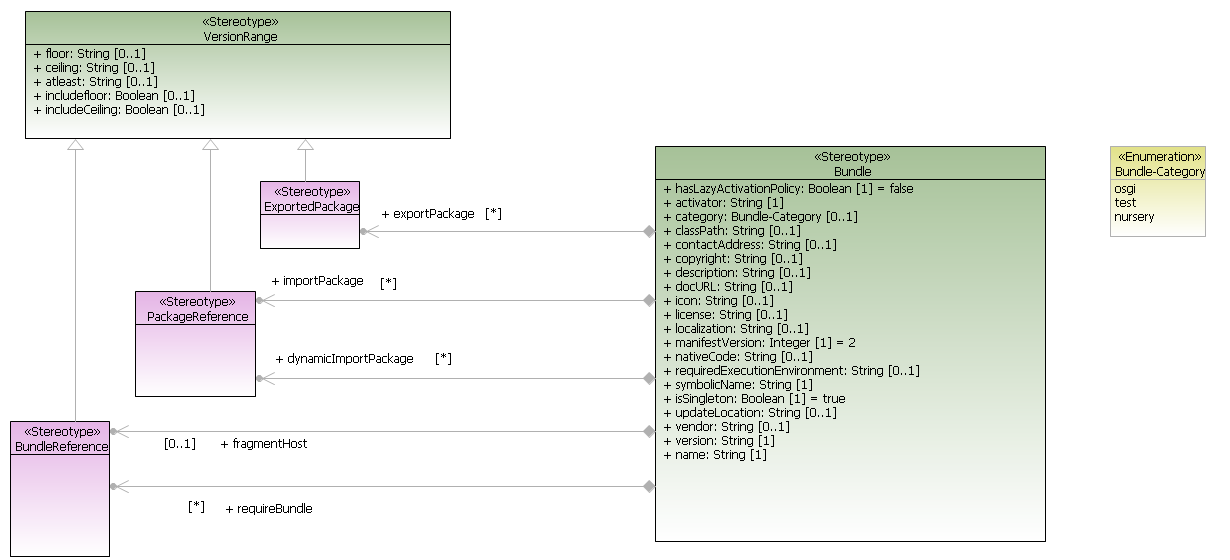


Figure 6: stereotypes of OSGI profiles

In this part, we detail each stereotype. Some content comes from the OSGI norm[2]

#### Bundle stereotype:

“The Framework defines a unit of modularization, called a bundle. A bundle is comprised of Java classes and other resources, which together can provide functions to end users. Bundles can share Java packages among an exporter bundle and an importer bundle in a well-defined way. In the OSGi framework, bundles are the only entities for deploying Java-based applications.”[2]

Because this element can contains java classes, packages or properties, the bundle stereotype extends the Component meta-class (Figure 7).

This stereotype contains several properties:

* ***hasLazyActivationPolicy : Boolean=false***

It specifies how the framework should activate the bundle once started.

* ***activator String***

The Bundle-Activator header specifies the name of the class used to start and stop the bundle.

* ***category Bundle-Category***

The Bundle-Category header holds a comma-separated list of category names.

* ***classPath String***

The Bundle-ClassPath header defines a comma-separated list of JAR file path names or directories (inside the bundle) containing classes and resources.

* ***contactAddress String***

The Bundle-ContactAddress header provides the contact address of the vendor.

* ***copyright String***

The Bundle-Copyright header contains the copyright specification for this bundle.

* ***description String***

The Bundle-Description header defines a short description of this bundle.

* ***docURL String***

The Bundle-DocURL headers must contain a URL pointing to documentation about this bundle.

* ***icon String***

The optional Bundle-Icon header provides a list of URLs to icons representing this bundle in different sizes.

* ***license String***

The Bundle-License header provides an optional machine readable form of license information.

* ***localization String***

The Bundle-Localization header contains the location in the bundle where localization files can be found.

* ***manifestVersion Integer 2***

The Bundle-ManifestVersion header defines that the bundle follows the rules of this specification.

* ***nativeCode String***

The Bundle-NativeCode header contains a specification of native code libraries contained in this bundle.

* ***requiredExecutionEnvironment String***

The Bundle-RequiredExecutionEnvironment contains a comma-separated list of execution environments that must be present on the OSGi framework.

* ***symbolicName String***

The Bundle-SymbolicName header specifies a non-localizable name for this bundle.

* ***isSingleton Boolean true***

It specifies if this plug-in must appear only on time in the platform.

* ***updateLocation String***

The Bundle-UpdateLocation header specifies a URL where an update for this bundle should come from. If the bundle is updated, this location should be used, if present, to retrieve the updated JAR file.

* ***vendor String***

The Bundle-Vendor header contains a human-readable description of the bundle vendor

* ***version String***

The Bundle-Version header specifies the version of this bundle

***Associations:***

* ***dynamicImportPackage PackageReference***

The DynamicImport-Package contains packages that should be dynamically imported when needed

* ***exportPackage ExportedPackage***

The Export-Package property references a list of exported packages

* ***fragmentHost BundleReference***

The Fragment-Host header defines the host bundles for this fragment.

* ***importPackage PackageReference***

The Import-Package header declares the imported packages for this bundle.

* ***requireBundle BundleReference***

The Require-Bundle specifies the list of bundles that must be imported, effectively requiring the public interface of another bundle

#### VersionRange stereotype:

A version range describes a range of versions using a mathematical interval notation.

If the property atLeast is filled, the property must be not filled.

The property floor and ceiling indicate aversion interval. The property includefloor and includeCeiling indicates if the interval is opened at left or right.

* ***floor String***

The floor property indicates the minimum version of the interval,

* ***ceiling String***

The ceiling property indicate the maximum version of the interval,

* ***atleast String***

The floor property indicates the minimum version

* ***includefloor Boolean***

If true the version indicated in the floor is included in the interval

* ***includeCeiling Boolean***

If true the version indicated in the ceiling is included in the interval

#### ExportedPackage stereotype:

This stereotype is applied on package. It represents java packages that can be exported by the plug-in. It inherits form VersionRange in order to indicate the version number for each if needed.

#### PackageReference stereotype:

A Bundle can reference package from another bundle. So this stereotype is applied on dependencies. It inherits form VersionRange in order to constraint the version of the package that is required.

#### BundleReference stereotype:

A Bundle can reference another bundles. So this stereotype is applied on dependencies. It inherits form VersionRange in order to constraint the version of the bundle that is required.

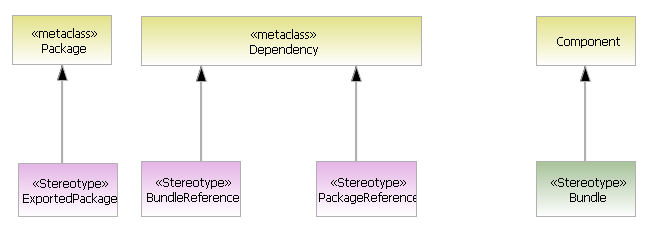


Figure 7: Extension of OSGI stereotypes

.

## The plug-in org.eclipse.papyrus.adl4eclipse.profile

This plug-in contains a profile that extends the OSGI profile. It defines the notion dedicated to the eclipse world.

Each stereotype is detailed. Some description comes directly from eclipse specification [1].

#### Plug-in stereotype

* ***isIncubation Boolean***

If true , the plug-in belong to an eclipse project that is in the incubation phases.

* ***lazyStart Boolean***

The Eclipse-LazyStart header is used to specify if a bundle should be started before the first class or resource is accessed from that bundle

* ***lazyStartException Package***

When Eclipse-LazyStart is true, the 'exceptions' attribute specifies a list of packages that will not cause the bundle to be activated when classes are loaded from them.

* ***platformFilter String***

The Eclipse-PlatformFilter is used to specify a platform filter for a bundle.

* ***buddyPolicy BuddyPolicyEnumeration***

The Eclipse-BuddyPolicy header is used to specify the buddy classloading policies for a bundle.the value of the enumeration can be dependent global registered app ext boot parent

* ***extensibleAPI Boolean***

The Eclipse-ExtensibleAPI is used to specify whether a host bundle allows fragment bundles to add additional API to the host.

* ***bundleShape BundleShapeEnumeration***

The Eclipse-BundleShape header is used to specify the deployment shape. The value is an enumeration: ***jar or dir***.

* ***capability String***

The Eclipse-GenericCapability header is used to specify a generic capability of a bundle. Generic capabilities can be used to describe features of your bundle which can be required by other bundles in the system (using the Eclipse-GenericRequire header).

Even if this property is becomen deprecated, we have add in the profile in order to model all plug-ins

* ***requireCapability String***
* The Eclipse-GenericRequire header is used to specify a requirement on a generic capability which is offered by another bundle (using the Eclipse-genericCapability header)
* ***genericCapabilityAlias String***

#### EclipseExportedPackage

* ***IsInternal Boolean***

It is used to specify whether the package is an internal package. The Plug-in Development Environment will discourage other bundles from using an internal package.

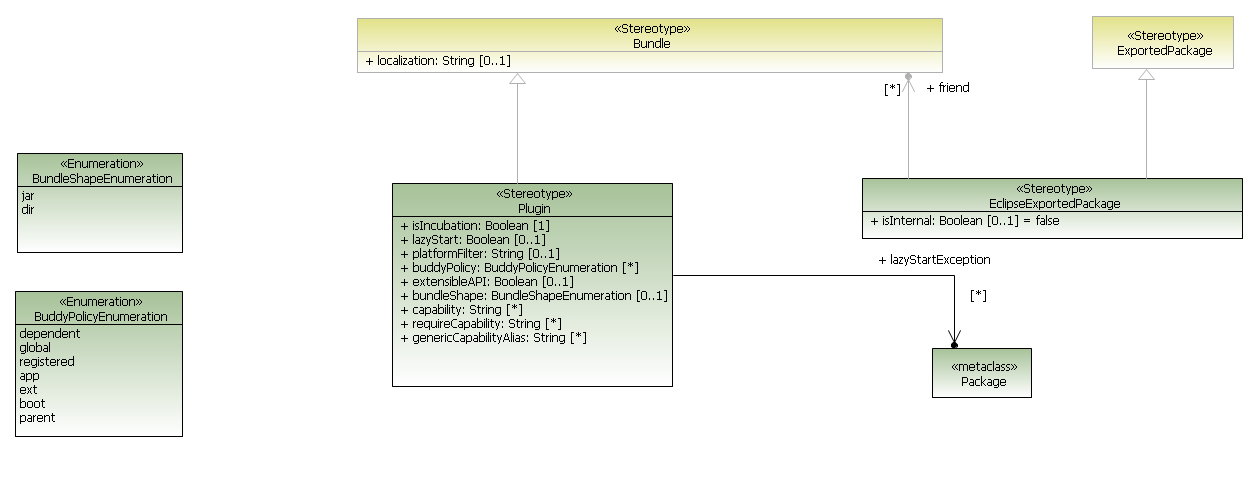


Figure 8: Plug-in stereotype

#### Feature

* ***description String***

Brief component description as simple text. Intended to be translated.

* ***copyright String***

Feature copyright as simple text. Intended to be translated.

* ***license String***

License as simple text. Intended to be translated

* ***provider String***

Optional display label identifying the organization providing this component. Intended to be translated

* ***image String***
* ***url String***

optional URL specifying site(s) contain feature updates, or new features

* ***plugins Plugin***

List of packaged plugins in the feature

* ***includedFeatures Feature***

Optional reference to a nested feature that is considered to be part of this feature.

* ***importedFeatures Feature***

It express in terms of plug-in dependencies.

* ***label String***

Optional displayable label (name). Intended to be translated

* ***id String***

required feature identifier

* ***version String***

required component version

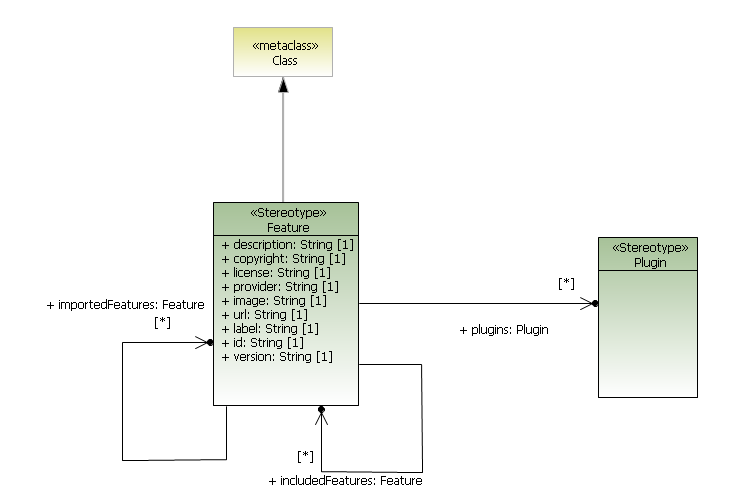


Figure 9: Feature stereotype

## The plug-in org.eclipse.papyrus.adl4eclipsetool

This plug-in contains algorithm to allow the creation of model from the structure of plug-ins or feature.

It contains a class that owned a generic algorithm to construct the model. This class is specialized to two classes: one dedicated to construct model of plug-ins and one dedicated to the construction of the feature.

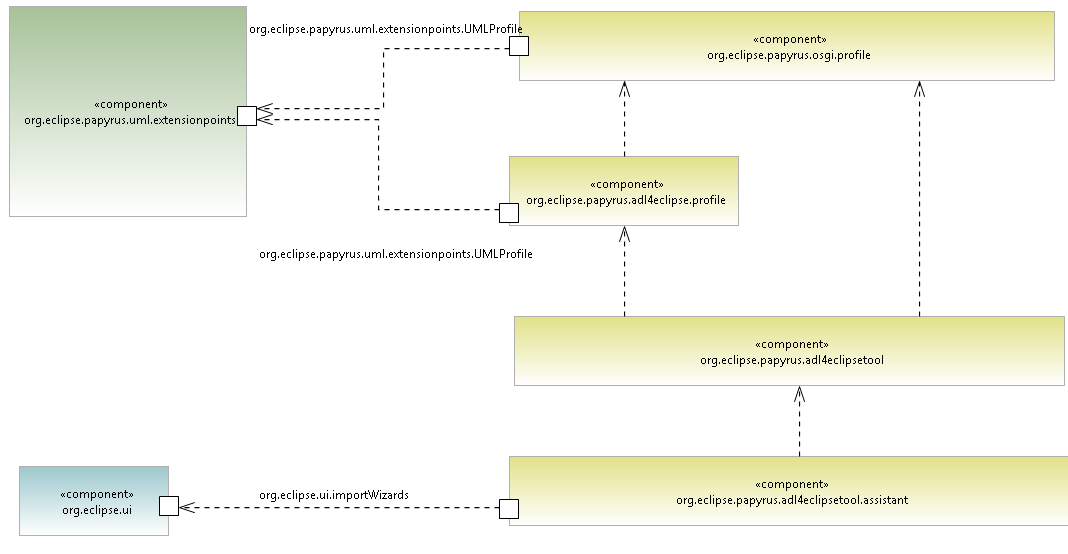
Moreover this plug-in contain delegation pattern in order to retrieve information from plug-in stored in the workspace or in the platform.

## The plug-in org.eclipse.papyrus.adl4eclipsetool.assistant

This plug-in contains UI to launch these algorithms of model creation.

Thanks to this if the Eclipse API change, this algorithm remains valid. It only needed to replace this plug-in.

In order to interface this tool inside the eclipse framework, the extension point org.eclipse.ui.importWizards has been implemented.

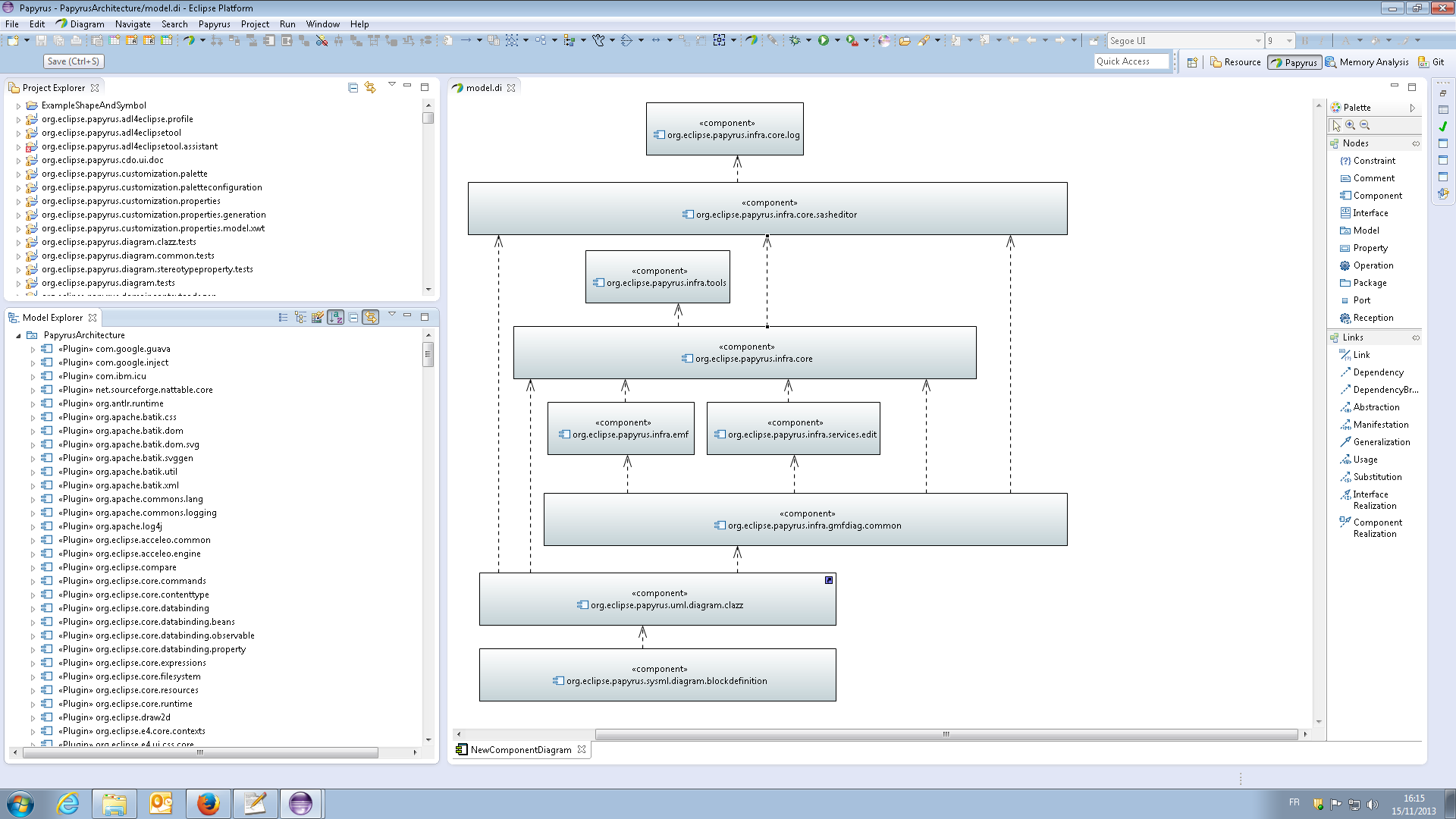


# Application on Papyrus

We have applied this tool to model the papyrus architecture:

Nb plugins 260 + 180 required plugins

Nb dependencies 3708



# Perspectives

Link to model requirements model done in AGESYS

Perform the generation of plug-in and feature. Allow round-trip.

Compare with wanted architecture: detect unwanted dependencies…

# References

[1] “Help - Platform Plug-in Developer Guide - Eclipse Platform.” [Online]. Available: http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fmisc%2Fbundle\_manifest.html. [Accessed: 19-Nov-2013].

[2] O. Alliance, “OSGi Service Platform, Core Specification, Release 5,” aQute, Mar. 2012.