1 Overview

Contents

1. [1.1 Table of Contents](#)

1 Overview

Papyrus is an environment for editing any kind of [EMF model](#), particularly supporting UML 2 ([Unified Modeling Language (UML) version 2.4.1](#)) and related modeling languages such as SysML ([System Modeling Language](#)) and MARTE ([Modeling and Analysis of Real-Time and Embedded systems](#)). Papyrus also offers very advanced support for UML profiles that enables users to define editors for DSLs (Domain Specific Languages) based on the UML 2 standard.

Papyrus is a collection of plug-ins and features on top of the Eclipse Modeling Framework. For more information about Eclipse, please go to the Eclipse web site [eclipse.org](#). Some of the terminology used in this Papyrus user guide are basic Eclipse concepts and briefly described here. To get more information about the Eclipse concepts, please visit the *Workbench User Guide* by selecting **Help > Help Contents** from within Eclipse.

1.1 Table of Contents

2 Introduction

Papyrus is built on the extensible Eclipse framework and is an implementation of the OMG (Object Management Group) specification Unified Modeling Language (UML) version 2.4.1. Papyrus is a comprehensive UML modeling environment, where many diagrams can be used to view different aspects of a system. Behind all diagrams, there is a model where all modeling elements, used in these diagrams, are kept. The model keeps the consistency between the diagrams.

UML diagrams can help system architects and developers understand, collaborate on and develop a system. Architects and managers can use diagrams to visualize an entire system or project and separate systems into smaller components for development.

System developers can use diagrams to specify, visualize, and document systems, which can increase efficiency and improve their system design. Also code can be generated from UML models.

Since UML is general-purpose modeling language in the field of software engineering, it is possible to adapt UML to specific domains. This is done by creating and applying UML profiles. Papyrus is a complete UML modeling environment, which also can be used to develop UML profiles.

2.1 Legend

In this user guide, bold text is used for menu selections, e.g. **Help > Welcome** means from the **Help** item on the main menu, select the **Welcome** item.

A context menu is the pop-up menu that appears when right clicking on something, e.g. right click on a class select **New Child > Create a new Operation**, will create a new operation on the class, using the class' context menu.

When text should be typed in, it is indicated by inline code, e.g. `this text should be typed in`.

Fields in wizards, pop-up windows, different editors, radio buttons and check boxes are indicated by italic text, e.g. set the field *Name* to `MyClass`.

3 Installation

It is a several step process to install Papyrus and its optional components. The *Eclipse Standard* must first be installed and when that is done, Papyrus is installed on top of *Eclipse Standard*.

3.1 Install Eclipse Standard

Eclipse Standard is installed from the Eclipse download page. On the download page select *Eclipse Standard <version number>* to install. Follow the install wizard to complete the installation.

3.2 Install basic Papyrus

When *Eclipse Standard* is installed, go to **Help > Install New Software** and type in `http://download.eclipse.org/releases/kepler/` in the field named *Work with:*.

Note! When this user guide was written, the Kepler release of Eclipse was the latest. Select the latest official Eclipse release.

Note! In some industrial environments, a proxy has to be used instead of this type of direct URL to the Eclipse web-site. To configure Eclipse to use a proxy is done under **Windows > Preferences** and **General > Network Connections**
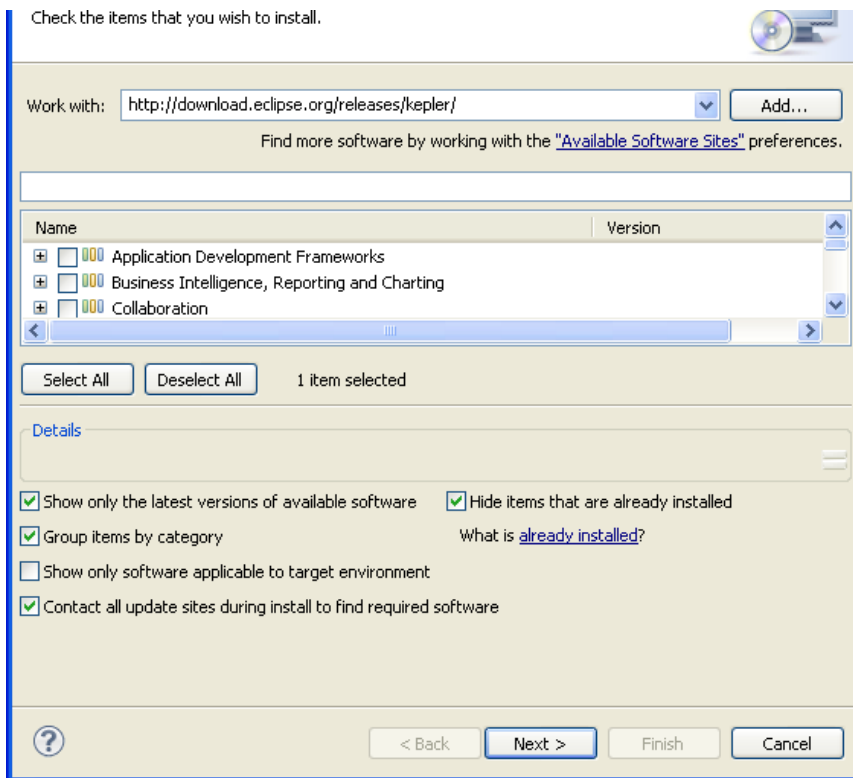
Figure 1: Install New Software wizard

In the *Name* column of the wizard, scroll down to *Modeling* and expand to the next level. Under *Modeling*, select *Papyrus UML* and follow the installation wizard to complete the installation.

When Eclipse is restarted, the environment is now ready for UML modeling.

3.3 Additional installation steps

After installation of the basic Papyrus feature, go to **Help > Install Papyrus Additional Components**. In the wizard that pops up, select the needed additional Papyrus components, e.g. to be able to do UML RT modeling, the *Real Time* component is needed. It is also recommended to install the *Diagram Stylesheets* and *Papyrus Compare* components. Follow the installation wizard to complete the installation.
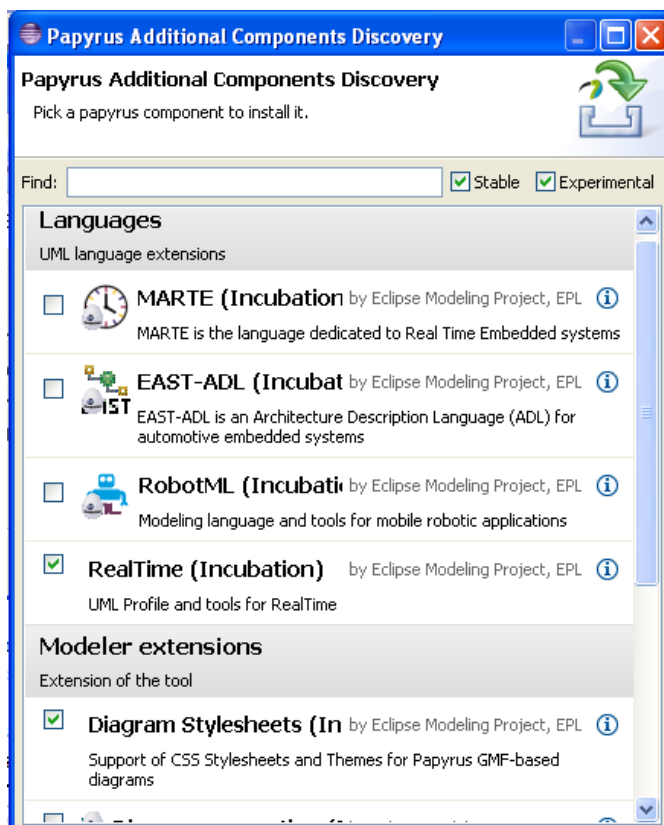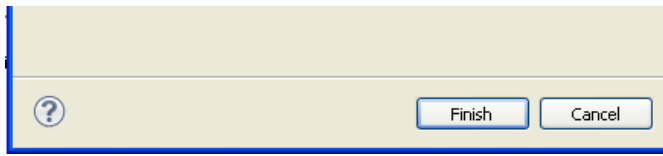
Figure 2: The Install Papyrus Additional Components wizard

4 Eclipse

Papyrus is built on the Eclipse framework, so most of its look and feel is inherited from Eclipse.

The Eclipse framework has a plug-in architecture, where plug-ins can be grouped into features. Features and plug-ins can be added to an existing Eclipse installation.

4.1 Architecture

The plug-in architecture applies also for all subsystems. A plug-in is the smallest unit of Eclipse Platform functionality that can be developed and delivered separately. Usually, a small tool is written as a single plug-in, whereas a complex tool has its functionality split across several plug-ins. Except for a small kernel known as the Platform Runtime, all of the Eclipse Platform's functionality is located in plug-ins. Plug-ins can be grouped into features.
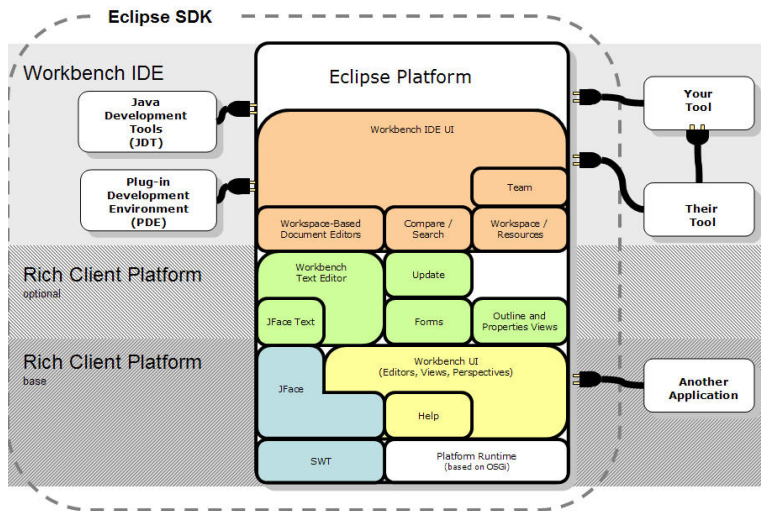
Figure 3: The Eclipse architecture

Plug-ins are coded in Java. A typical plug-in consists of Java code in a JAR (Java Archive) library, some read-only files, and other resources such as images, Web templates, message catalogs, native code libraries, and so on. Some plug-ins do not contain code at all. One such example is a plug-in that contributes online help in the form of HTML pages. A single plug-inâ€™s code libraries and read-only content are located together in a directory in the file system, or at a base URL on a server. There is also a mechanism that permits a plug-in to be synthesized from several separate fragments, each in their own directory or URL. This is the mechanism used to deliver separate language packs for an internationalized plug-in.

Each plug-in has a manifest file declaring its interconnections to other plug-ins. The interconnection model is simple: a plug-in declares any number of named extension points, and any number of extensions to one or more extension points in other plug-ins.
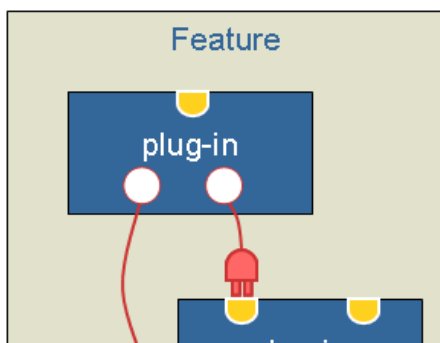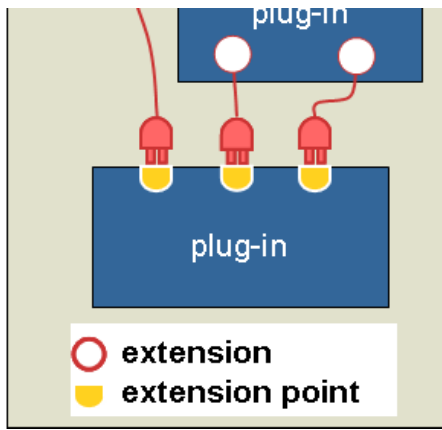
Figure 4: Plug-ins and Features

4.2 Workspace

The workspace is located in the file-system and is the place where Eclipse resources (files, folders and projects) are stored. When Eclipse is started, a pop-up window appears, where a workspace should be selected. One instance of Eclipse is connected to one workspace.
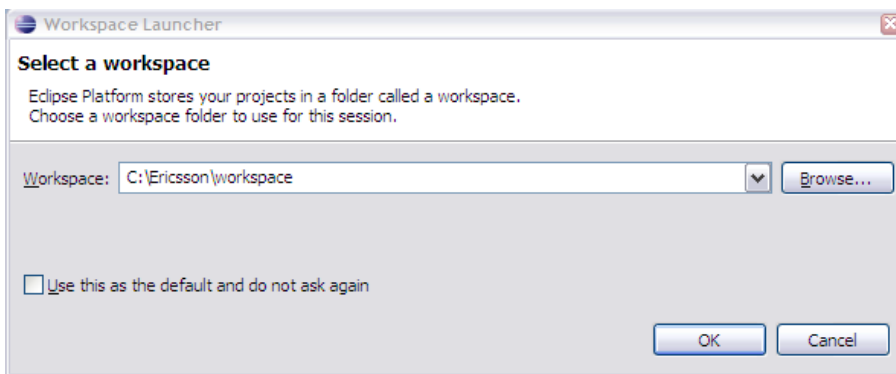
Figure 5: Pop-up window to select the workspace

In the file system all resources are stored in the selected workspace and in the same hierarchical structure as in the *Project Explorer*.

4.3 Resources

Resources are a collective term for the projects, folders, and files that exist in the workbench. The resources are stored in the workspace, where the projects are on the first level. Inside a project, there are files and folders in the same hierarchical structure as in the *Project Explorer* and *Model Explorer*.

Files appear in the file system as files and folders are Unix directories or Windows folders and may contain other files and folders. Each time a file is saved, a copy is saved, which makes it possible to replace the current file with a previous edit or even restore a deleted file. Earlier versions of a file can be compared to the contents of all the local edits. Each edit in the local history is time stamped, i.e. is uniquely represented by the date and time the file was saved.

Projects can be viewed as the top level folder in the file system under the workspace. In Eclipse there are different types of projects, e.g. Model, C/C or Java projects and they are the top level resource in the *Project Explorer*. Projects can be closed and opened in the *Project Explorer*.

4.4 Workbench

The workbench is the Eclipse user interface and is used to navigate, view, and edit resources in a workspace, i.e. the workbench is the Eclipse IDE's application window. The workbench presents one or more editors and *views* that are gathered into adjustable groups (*perspectives*).

The first time Eclipse is started, after the installation, a *Welcome* page is presented. Take a few minutes to explore the product overview and getting started information that is located here.
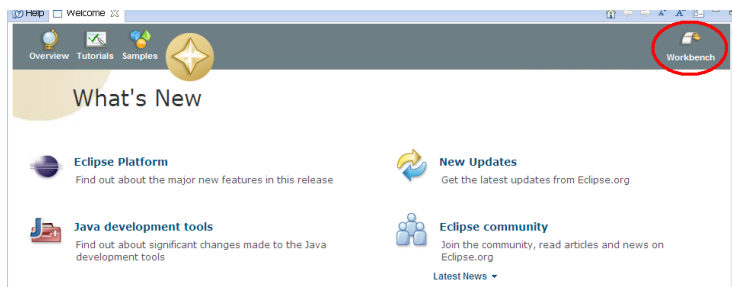
Figure 6: The Eclipse welcome page

To return to the ordinary workbench, just click on the workbench icon up to the right. When starting Eclipse, subsequent times, the workbench appears directly. To visit the welcome page at any time, just select **Help > Welcome**.

The title bar of the workbench window and the little Papyrus icon to the right indicates which *perspective* is active. In this example, the Papyrus *perspective* is in use. The *Project Explorer* and the *Model Explorer, Outline, Properties* views, etc. are open, along with a *Class Diagram* editor and its tool palette.
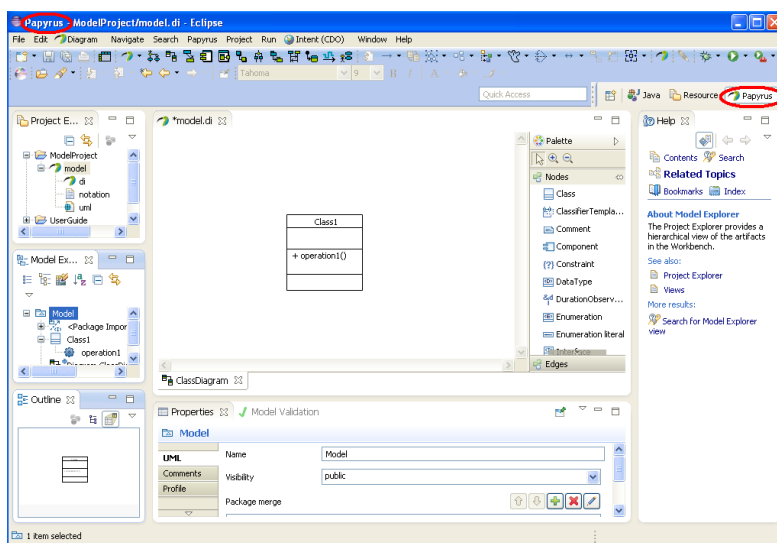
Figure 7: The Papyrus perspective in the workbench

It is easy to toggle between perspectives by clicking on some shown *perspective* in top of the right hand corner or open a new one by clicking on the *Open Perspective* icon ⬚ and browse to the *perspective* to open. It is also possible to reorganize a *perspective*, open/close *views*, customize menus, etc. and then save the *perspective* with a new name by **Window > Save Perspective As ...**.

4.4.1 Views

Views and editors are the main visual entities that appear in the workbench. Any given *perspective* can contain multiple editors and a number of surrounding *views* that provide context. *Views* provide different ways to visualize, navigate and edit the resources in the Eclipse workspace. *Views* can be single or stacked on top of each other.

Views, including editor views, can be resize, moved, detached. In addition, a *view* can be maximized to cover the entire workbench by double-clicking on its tab. By double-clicking once more, it will return to its original size. Some *views* has a view specific menu, e.g. *Project Explorer* view, where some specific view settings could be done.

To add a new *view* to the active *perspective*, use **Window > Show View**and if the desired *view* does not appear on top of the pop-up menu, select **Other**, which opens up a *view* browser, where all available *views* are organized in different categories.

4.4.1.1 Single views

In the workbench figure, above, several single views appears, e.g. *Outline, Class Diagram* editor, *Help, Project Explorer* and *Model Explorer* views. The *single view* has only one tab with the view name. By draging a *single view* tab and release it on another *single view* tab, *stacked views* are created.

4.4.1.2 Stacked views

In the workbench figure, above, also *stacked views* appear, e.g. *Properties* and *Model Validation* views. To select the one that should be on top of the stack, just click on its tab and it becomes visible. By selecting a *stacked view* and drag it besides another view in the workbench, a *single view* will appear.

4.5 Preferences

To customize the settings for the Eclipse workbench and the installed features, the preference window is used. Use **Windows > Preferences** to open it, e.g. under **General > Keys** in the preference window there are shortcuts and keys defined for the user interface. Here they can be redefined or own sets could be defined.
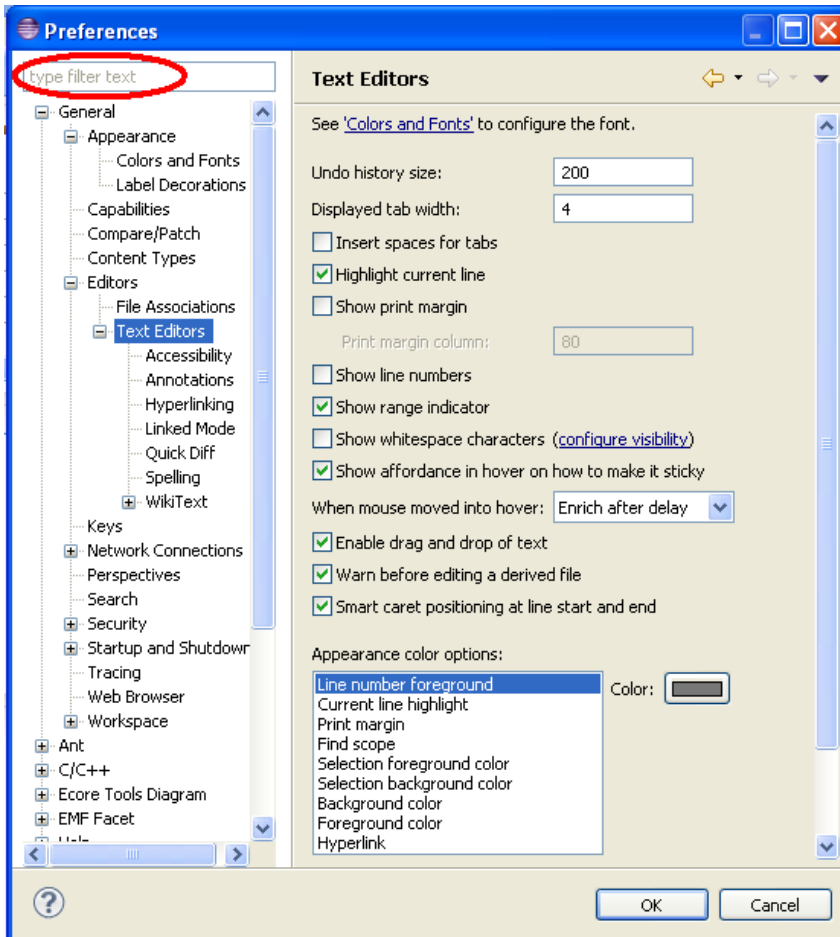
Figure 8: The preference window

The preference window pages can be searched using the filter function. To filter by matching the page title, simply type the name of the page and the available pages will be presented below.

The filter also searches on keywords. By the history controls (the left, right and drop-down arrows up in the right corner of the preference window) it is possible to navigate through previously viewed pages. To step back or forward several pages at a time, click the drop-down arrow and a list of the most recently viewed preference pages are displayed.

4.6 Import and Export

Projects can be shared between workspaces by using project import and export, which are done through wizards. To open the import wizard, use **File > Import** and in several steps select what, where from and if it should be imported as a copy or just referenced. To export resources, There are also an export wizard, which is opened by **File > Export** and select details about what should be exported, if it should be compressed and where to export it to.

Figure 10: Export wizard

Figure 9: Import wizard

When importing a project into the used workspace, it can be copied by checking the box *Copy ...* in the import wizard. If this check box is unchecked, there will just be a reference to the other workspace and when editing that project, it will be edited in its original place. Be aware of that when doing so, several instances of Eclipse may edit the same resource.

When exporting a project, browse to the place where to export it to name it and select if and how compression should be used.

5 Modeling

Papyrus is a comprehensive UML modeling environment, where diagrams can be used to view different aspects of a system. Behind all the diagrams, there is a model where the modeling elements, used in these diagrams, are stored. The model maintains the consistency between all diagrams.

A model is the collection of all the modeling elements and relationships that compose a software system. Papyrus enables the creation, viewing and manipulation of UML diagrams as specified in the UML 2 specification.

The model defines every element, representing some part of the system. Multiple model diagrams can reference an element many times. Each of the different diagrams can view a different aspect of the system.
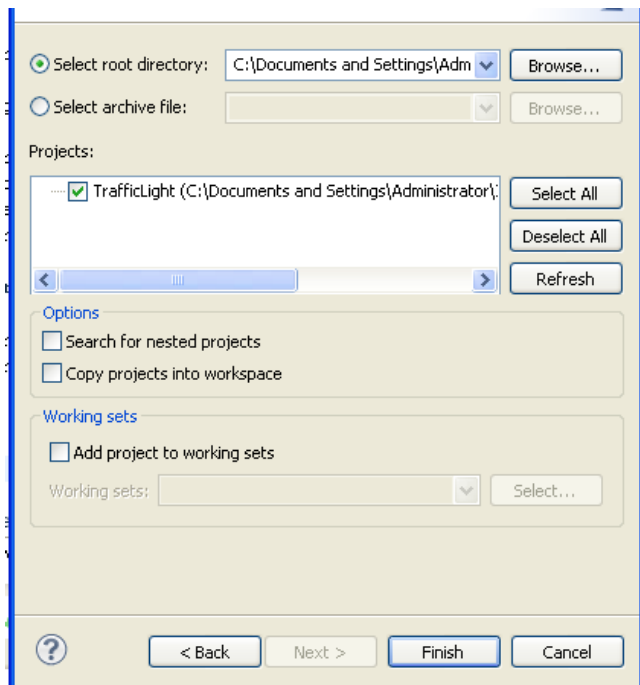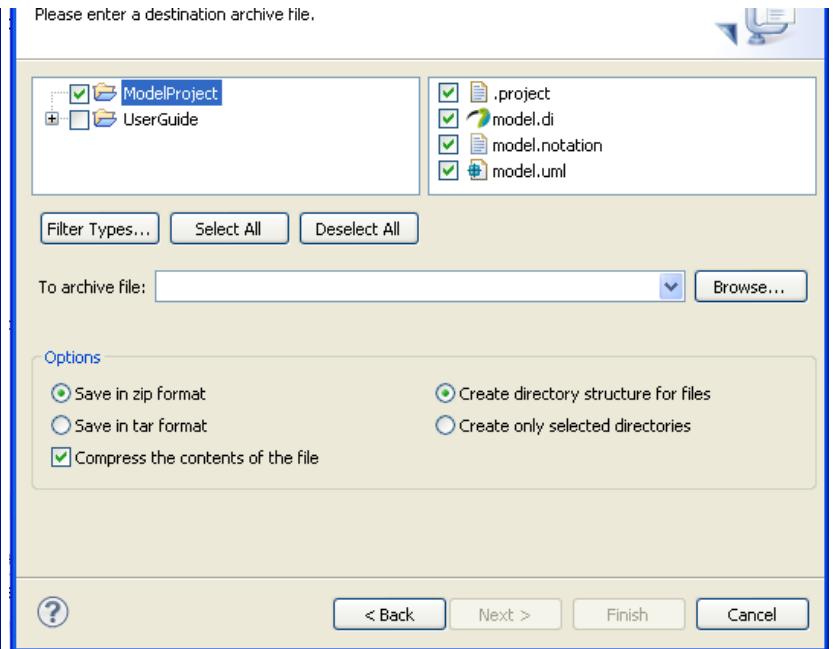
The model is the basis of the diagrams and keep the diagrams consistent. The diagrams are stored in the model's hierarchical structure. Some are owned by a modeling element like a class and some are just organized into packages and then owned by the package (a.k.a folders in basic Eclipse projects). **Note!** the top level of the model is the model package, which is a special kind of a package.

5.1 Model and diagrams

The model is the basis for all diagrams and maintains the consistency between the diagrams. The model is a collection of definitions of elements that compose the system and the relationships between them. Diagrams can be used to view subsets of the underlying model and from various view points. A model of a system may require many different diagrams to represent different views of the system for different project stakeholders.

In Papyrus, diagrams are be viewed and created in the *Model Explorer* view. The *Model Explorer* shows diagrams in their logical place within the model.

The visual representation of a system that diagrams provide can offer both low-level and high-level insights into the concepts and design of a system.

6 Tutorials

The tutorials are focused on selected topics regarding the use of Papyrus and contains step by step instructions on how to create and manipulate the workbench and models.

6.1 Getting started with general Eclipse functionality

This tutorial is about to understand the workbench environment and the basic Eclipse terminology.

6.1.1 Exploring perspectives

This part demonstrates the differences between the *Papyrus* and the *Resource* perspectives and also how to customize the *Resource* perspective.

6.1.1.1 Exploring the *Papyrus* perspective

Explore the Papyrus default menus, toolbar, and views in the *Papyrus* perspective.

1. To switch to the *Payrus* perspective, click **Window > Open Perspective > Other**. Then choose the *Papyrus* perspective. (Notice the workbench title bar and perspective bar reflect that the *Papyrus* perspective is active. Notice also the main menu items, toolbar buttons, and views that is visible in this perspective.)
2. Click **File > New** and notice that the menu contains the items *Papyrus Project* and *Papyrus Model* among other items.
3. Click **Window > Show View** and notice that the menu contains the items *Model Explorer*, *Model Validation* and more.

6.1.1.2 Exploring and customizing the Resource perspective

Explore the default menu, toolbar, and views in the *Resource* perspective and customize a menu.

1. To switch to the *Resource* perspective, click **Window > Open Perspective > Other**. Then choose the *Resource* perspective. Notice that the workbench title bar and perspective bar reflect that the *Resource* perspective is active. (Notice the main menu items, toolbar buttons, and views visible in this perspective)
2. Click **File > New** and notice that there are no Papyrus items in the menu.
3. Click **Window > Show View** and notice that the menu does NOT contain the items *Model Explorer* and *Model Validation*.
4. Click **Window > Customize Perspective**. Ensure that the workbench title bar and perspective bar reflect that the *Resource* perspective is active.
5. In the *Customize Perspective - Resource* pop-up window, select the *Shortcuts* tab and select **Show View** from the *Submenus* combo box.
6. Clear all check boxes in the *Shortcut Categories* list.
7. Click *General*(not check the check box) in the *Shortcut Categories* list, select the check box next to *Project Explorer* in the *Shortcuts* list, and click *OK*.
8. Click **Window > Show View** and notice the menu now just contains the *Project Explorer* item.

6.1.2 Creating a new project, folder and files

In this section a new project, folder and files will be created in the *Resource* perspective.

6.1.2.1 Creating a new general project

Create a new project in the *Resource* perspective by completing these steps:

1. If necessary switch to the *Resource* perspective by clicking on **Window > Open Perspective > Other**. Then choose the *Resource* perspective. If the *Resource* perspective already is active, click on **Window > Reset Perspective...** to get back to its default configuration.
2. Click **File > New > Project**.
3. In the *New Project* wizard, name the project `My Project` and click *Finish*.

6.1.2.2 Creating a new folder

Create a folder in the project:

1. Right-click on *My Project* in the *Project Explorer* and select **New > Folder**.
2. Type `Text Files` in the the field *Folder name*, and then click *Finish*.

### 6.1.2.3 Creating and editing a new file

Create a file in the folder:

1. In the *Project Explorer*, right-click on the *Text Files* folder and select **New > File**.
2. In the *New File* wizard, ensure that *My Project/Text Files* is the parent folder. Type `My File` as the file name and click *Finish*. (Notice that a text editor opens in the editing view for the created resource)
3. Enter any text into the editor view for *My File*. Notice the asterisk (*) next to the file name indicates unsaved changes.
4. Press *Ctrl S* to save the work. Notice that the asterisk disappears.

6.1.2.4 Creating another file

Create another file in the *Text Files* folder, using the workbench menu this time.

1. On the workbench menu, click **File > New > File**.
2. In the *New File* wizard, expand *My Project* and then select *Text Files* as the parent folder.
3. Type `My Other File` as the file name and click *Finish*.
4. Reviewing the contents of the *Project Explorer* view, which should be like this
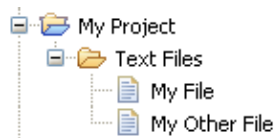
6.1.3 Exploring editors and views

This section demonstrates how to manipulate views and editors.

6.1.3.1 Maximizing and restoring an editor

Maximize one of the editors to expand the viewable area:

1. Double-click the file name on the editor tab for *My File*.
2. Double-click the file name again to restore the editor to its original size.

6.1.3.2 Tiling and restacking the editors

Currently, the editors are stacked one in front of the other. Try tiling them horizontally and vertically:

1. Click the *My Other File* editor tab and drag it to the bottom of the editor pane. Colored frames indicates how the views will be tiled. Drop the editor and notice that the editors are tiled horizontally.
2. Drag the *My Other File* editor tab to the left of the editor pane and release it. Notice that the editors are tiled vertically.
3. Restack the editors by dragging one of the editor tabs on top of the other.

6.1.3.3 Organizing views

Try moving a view:

1. Drag the title bar of the *Outline* view onto the title bar of the *Project Explorer*.
2. Experiment by dragging the title bar of the *Outline* view to various locations within the workbench.
3. Return the *Outline* view to its original place to the lower left of the workbench. It is always possible to return to the default configuration of the active perspective by clicking on **Windows > Reset Perspective...**

6.1.3.4 Using view menus

Some views has view menus, e.g. *Project Explorer* view, which is indicated by a down

Some views has view menus, e.g. *Project Explorer* view, which is indicated by a down arrow in the upper-right corner of the view. Click this down arrow of the *Project Explorer* view and review the pull-down menu options specific for this view.

6.1.3.5 Closing and opening views

If a view does not appear in the workbench, it can be open by using the **Window** menu:

1. Close the *Project Explorer* view by right-click on the *Project Explorer* view tab and select **Close**.
2. Click **Window > Show View > Other** and type `Project Explorer` in the filter text box.
3. Select *Project Explorer* and click *OK* which opens the view again.

6.1.4 Exporting and importing a project

This section demonstrates how projects can be shared between users and workspaces using the *export* and *import* feature.

6.1.4.1 Exporting a project

Export *My Project* to a compressed file:

1. Select **File > Export...** from the workbench menu.
2. In the *Export* wizard, expand **General**, and then select **Archive File**. Click *Next*.
3. Check the check box next to *My Project* in the resource list.
4. Click *Browse* to specify an export destination in the *To archive file* field. Note the available options for archive formats.
5. In the *Browse* window, select the *Desktop* as the destination folder and type `My Project` as the file name. Click *OK*.
6. In the *Export* wizard, click *Finish* to perform the export process.
7. View the desktop and notice the new compressed (.zip) file.

6.1.4.2 Removing the project from the workspace

Remove *My Project* from the workspace:

1. Right-click *My Project* in the *Project Explorer* and select **Delete** from the context menu.
2. In the *Delete Resources* pop-up window, make sure to check the

Delete project contents on disk check box. Click *OK*. Note: If this check box is unchecked, the resource will just be deleted from the Workbench, but it will still exist in the workspace. (Notice *My Project* is no longer listed in the *Project Explorer*)

6.1.4.3 Importing a project

Import *My Project* from a compressed file:

1. Select **File > Import...** from the workbench menu.
2. In the *Import* wizard, expand **General**, and then select **Existing Projects into Workspace**. Click *Next*.
3. Click the **Select archive file** radio button.
4. Click the *Browse* button.
5. In the *Browse* window, browse to the *Desktop* and select **My Project.zip** and click *Open*.

#In the

Import wizard, check the check box next to *My Project* and click *Finish* to perform the import process. (Notice *My Project* is once again listed in the *Project Explorer*)

6.1.5 Conclusion

The basic features of the Eclipse workbench have now been demonstrated.

- How to view and customize perspectives.
- How to create a project, folder and files.
- How to manipulate editors and views.
- How to export and import projects.

6.2 Creating profiles

6.3 Creating models using Papyrus

6.3.1 Use-case modeling

6.3.2 Design modeling

This is a tutorial about general class and object modeling using Papyrus.

6.3.2.1 Create a new UML project

Create a new UML modeling project as follows:

1. If necessary switch to the *Papyrus* perspective by clicking on **Window > Open Perspective > Other**. Then choose the *Papyrus* perspective. If the *Papyrus* perspective is already active, click on **Window > Reset Perspective...** to get back to its default configuration.
2. Click **File > New > Papyrus Project**.
3. In the *New Papyrus Project* wizard, name the project `My Design Model` and click *Next*.
4. Make sure that the radio button *UML* is selected and click *Next*.
5. Under the section *You can load a template*, check the box *A UML model with basic primitive types (ModelWithBasicTypes)* and click on *Finish*.

6.3.2.2 Create new packages to be used for classes

Create two packages in the model:

1. Right-click on the *model* package in the *Model Explorer*, select **New Child > Create a new Package** and select the created package in the *Model Explorer*.
2. In the *Properties* view type `Clients` in the the field *Name*.
3. Right-click on the *model* package in the *Model Explorer*, select **New Child > Create a new Package** and select the created package in the *Model Explorer*.
4. In the *Properties* view type `Server` in the the field *Name*.

6.3.2.3 Create new classes

Create two classes in the *Server* package. One is called *Server1* and the other is called *DataClass1*:

1. Right-click on the *Server* package in the *Model Explorer*, select **New Child > Create a new Class** and select the created class in the *Model Explorer*.
2. In the *Properties* view type `Server1` in the the field *Name*.
3. Right-click on the *Server* package in the *Model Explorer*, select **New Child > Create a new Class** and select the created class in the *Model Explorer*.
4. In the *Properties* view type `DataClass1` in the the field *Name*.

Create three different client classes in the *Clients* package:

1. Right-click on the *Clients* package in the *Model Explorer*, select **New Child > Create a new Class** and select the created class in the *Model Explorer*.
2. In the *Properties* view type `Client1` in the the field *Name*.
3. Right-click on the *Clients* package in the *Model Explorer*, select **New Child > Create a new Class** and select the created class in the *Model Explorer*.
4. In the *Properties* view type `Client2` in the the field *Name*.
5. Right-click on the *Clients* package in the *Model Explorer*, select **New Child > Create a new Class** and select the created class in the *Model Explorer*.
6. In the *Properties* view type `ClientRoot` in the the field *Name*.

6.3.2.4 Create new class diagrams

Create two class diagrams in the model:

1. Right-click on the *model* package in the *Model Explorer*, select **New Diagram > Create a new Class Diagram** and type `Packages` in the *Enter a new diagram name* pop-up window.
2. Right-click on the *model* package in the *Model Explorer*, select **New Diagram > Create a new Class Diagram** and type `Classes` in the *Enter a new diagram name* pop-up window.

6.3.2.5 Create new operations and attributes

1. Right-click on the class *DataClass1* and select **New Child > Create a new Property** and select the created attribute in the *Model Explorer*.
2. In the *Properties* view, type `Field1` in the field *Name* and by the *Type* field click on the ⋯ key and select *Integer* from the *UML Primitive Types*. This cause the created attribute to be named *Field1* and to be of type *Integer*.
3. Follow the above pattern to also create the attributes *Field2* of type *Integer*, *Field3* of type *String*, and *Field4* of type *String* on class *DataClass1*.
4. Create the attributes *Attribute2* of type *String*, *Attribute4* of type *Integer* and *Attribute5* of type *DataClass1* on class *Client1*.
5. Create the attribute *Attribute3* on class *Client2*.
6. Create the attribute *Attribute1* on class *Server1*.
7. Create the operation *service1* on class *Server1* by right-click on on the class *Server1* and select **New Child > Create a new Operation** and select the created operation in the *Model Explorer*.
8. In the *Properties* view, type `service1` in the the field *Name* and by the *Owned parameter* field click on the ⊕ key.
9. In the *Create a new parameter* pop-up window, type `service1return` in the *Name* field, select *return* from the *Direction* field drop down list and by the *Type* field click on the ⋯ key and select *Integer* from the *UML Primitive Types*. This causes the return type of the operation to be defined as an integer.

6.3.2.6 Create new relationships between classes

Create a *Dependency* relationship between the *Clients* and the *Server* packages:

1. Open the *Diagram Packages* diagram by double click on it in the *Model Explorer*
2. Drag the *Clients* package to the class diagram (by click on it and while holding the mouse button down, move the cursor to the editing area in the class diagram editor and release it).
3. Drag the *Server* package to the class diagram.
4. Select the *Dependency* tool from the *Edges* drawer in the *Palette*, click on the *Clients* package and then on the *Server* package in the class diagram.
5. Type `Dependency` as the name of the relationship.

Create a *Generalization* relationship between the classes *Client2* and *Client1*, i.e. make *Client2* a sub-class of *Client1*:

1. Open the *Diagram Classes* diagram by double click on it in the *Model Explorer*
2. Drag the *Client1* class to the class diagram.
3. Drag the *Client2* class to the class diagram.
4. Select the *Generalization* tool from the *Edges* drawer in the *Palette*, click on the *Client2* class and then on the *Client1* class in the class diagram.

Create a *Composite* relationships between the classes *ClientRoot* and *Client1* plus *ClientRoot* and *Client2*:

1. Continue with the already opened class diagram *Diagram Classes*.
2. Drag the *ClientRoot* class to the class diagram.
3. Select the *Association* tool from the *Edges* drawer in the *Palette*, click on the *ClientRoot* class and the on the *Client1* class in the class diagram.
4. Select the created association in the class diagram and in the *Properties* view, rename the association to `cr-c1`.
5. Also in the same *Properties* view, at the member end, named *client1*, by the *Aggregation* field, select *composite* from the drop down list.
6. Select the *Association* tool from the *Edges* drawer in the *Palette*, click on the *ClientRoot* class and the on the *Client2* class in the class diagram.
7. Select the created association in the class diagram and in the *Properties* view, rename the association to `cr-c2`.
8. Also in the same *Properties* view, at the member end, named *client2*, by the *Aggregation* field, select *composite* from the drop down list.

6.3.2.7 Create a new package to be used for objects

Create a new package in the model:

1. Right-click on the *model* package in the *Model Explorer*, select **New Child > Create a new Package** and select the created package in the *Model Explorer*.
2. In the *Properties* view type `Objects` in the the field *Name*.

6.3.2.8 Create new objects

Create objects (instances of classes) in the *Objects* package:

1. Right-click on the *Objects* package in the *Model Explorer*, select **New Child > Create a new InstanceSpecification** and select the created object (InstanceSpecification) in the *Model Explorer*.
2. In the *Properties* view type `clientObj1` in the the field *Name* and by the *Classifier* field click on the ⊕ key. In the *Classifier* window that pops up, browse to the *Client2* class and click on the ⇨ key, which specifies the object's class as shown in figure 11.
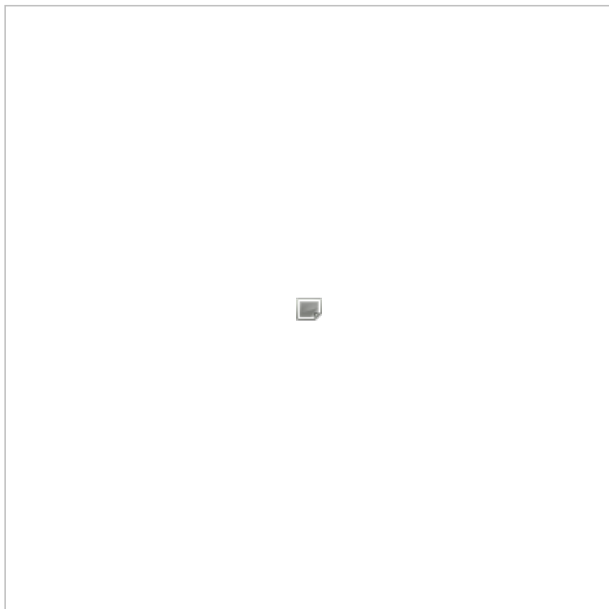

Figure 11: The Classifier pop up window

Follow the same pattern to create:

1. An object called *clientObj2* in the *Objects* package based on class *Clent2*.
2. An object called *serverObj1* in the *Objects* package based on class *Server1*.

6.3.2.9 Create a new class diagram

Create a class diagrams in the model to depicts the created objects:

1. Right-click on the *model* package in the *Model Explorer*, select **New Diagram > Create a new Class Diagram** and type `Objects` in the the *Enter a new diagram name* pop-up window.

6.3.2.10 Create new relationships between objects

Create a *Dependency* relationship between the clients and the server objects:

1. Open the *Diagram Objects* diagram by double click on it in the *Model Explorer*
2. Drag the *clientObj1* object to the class diagram (by click on it and while holding the mouse button down, move the cursor to the editing area in the class diagram editor and release it).
3. Drag the *clientObj2* object to the class diagram
4. Drag the *serverObj1* object to the class diagram.
5. Select the *Dependency* tool from the *Edges* drawer in the *Palette*, click on the *clientObj1* object and then on the *serverObj1* object in the class diagram. Leave the default name on the relationship.
6. Also create a *Dependency* relationship between the *clientObj2 and the* serverObj1''.

6.3.2.11 Conclusion

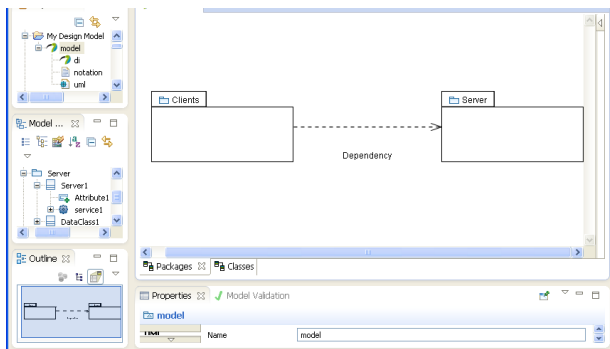In this tutorial the following model was created:
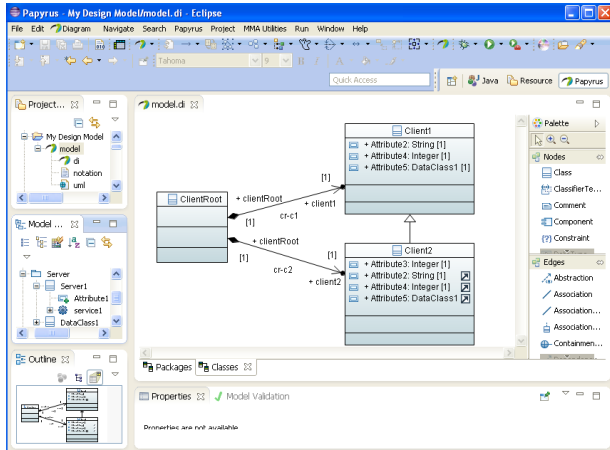
Figure 12: Two packages on top in the model

Figure 13: Relationships between the client classes

Note! The inherited attributes depicted in figure 13 on the *Client2* class.
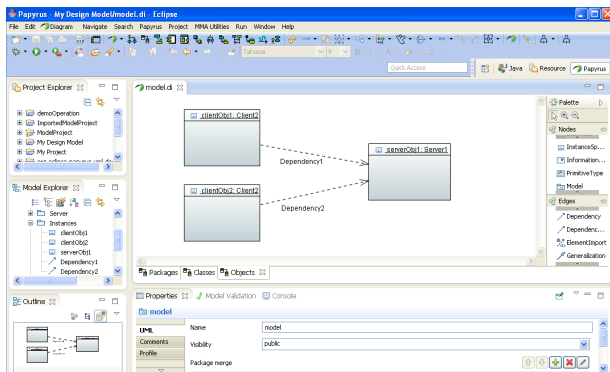

Figure 14: Objects in the model and their relationships

6.3.3 RT modeling

7 Papyrus

Papyrus can be used to do

- General UML modeling.
- UML RT modeling, which is described in the section 6.3.3_RT_modeling when the UML RT profile is applied.
- SysML modeling when the SysML profile is applied.
- MARTE modeling when the MARTE profile is applied.
- UML profiles, which is described in the section UMLprofiling .

In Papyrus, different UML profiles can be applied. When installing Papyrus, as described in section Installation , also the UML, UML RT, SysML and MARTE profiles can be added. When creating a new Papyrus project, the type of Papyrus project is selected. Project types to chose from are SysML, Profile and UML.

7.1 Papyrus resources in the workspace

When modeling in Papyrus, three types of resources are stored in the workspace.
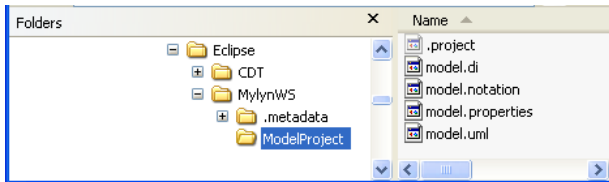
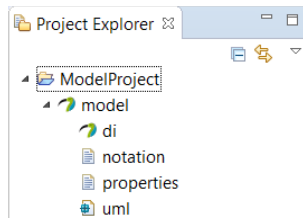Figure 15: Resources in the file system

Figure 16: Resources in
the Project Explorer

- **.di** file persists the status of the workbench, i.e. which diagrams and views are opened, etc.
- **.notation** file persists the information about the diagrams in the model.
- **.uml** file persists the UML model.
- **.properties** file persists the UML labels.

Note! In this case the model is contained in three files. When working in an industrial context, the model may need to be split up into several fragments in order for several designers to work concurrently with the same model. This is described in section Papyrus in a team environment .

7.2 The Papyrus perspective

When Papyrus is installed a predefined perspective called *Papyrus* is made available. This is the perspective to use when modeling with Papyrus. The Papyrus perspective can be customized to the user needs and saved as new perspective (see section Workbench).

7.2.1 Project Explorer view

The *Project Explorer* view is used to browse, select and manipulate resources in the workspace. Projects or working sets are the top level in this view. From the *Project Explorer's* (right click on the white space) context menu, e.g. new projects can be created.
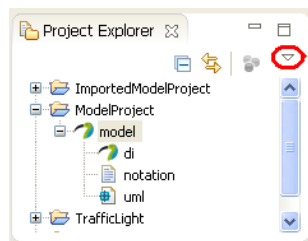

Figure 17: The Project
Explorer

In some views, as in this case, there is a view specific menu (indicated in Figure 13 with a red ring). Here some settings can be applied for the view, e.g. if the top level should be working sets or projects.

7.2.2 Model Explorer view

In the *Model Explorer* view, the model that has been opened in the *Project Explorer*, can be browsed and edited. Model elements can be added by using the context menu of any existing modeling element, including the model package and packages. Diagrams can also be added by using the context menus. Existing diagrams can be opened in an appropriate editor by just double clicking on the diagram in the *Model Explorer*.
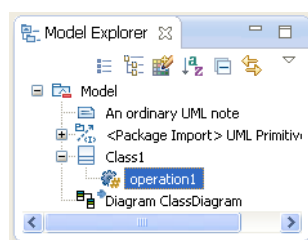

Figure 18: The Model
Explorer

7.2.3 Editing view

The *Editing View* is in the middle part of the workbench and here opens different types of editors, depending on the type of resource to edit, e.g. if a class diagram is opened, the class diagram editor will be visible in the *Editing View*.
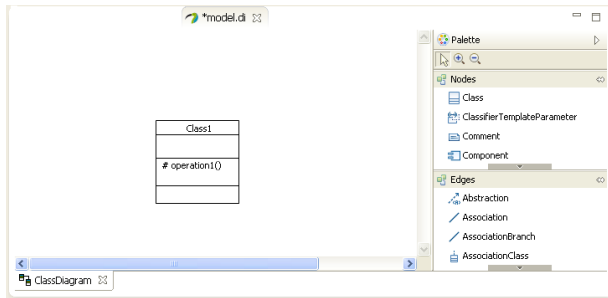

Figure 19: The Editing View

How to use editors is described in section Editors in Papyrus.

7.2.4 Outline view

The *Outline View* is connected to the *Editing View* and gives an overview of what is open in the *Editing View*. The *Outline View* may be used to pan the *Editing View* or to select some information that will be highlighted in the *Editing View*. The shaded area is the area that is visible in the *Editing View*.
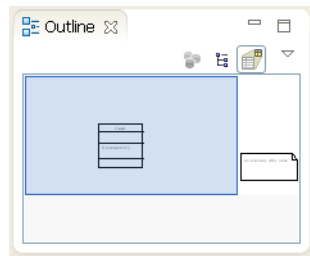

Figure 20: The Outline View

7.2.5 Properties view

The *Properties view* is a stacked view which is located at the bottom of the workbench and shows the properties of a selected modeling element. The modeling element can be selected in the Model Explorer or in a diagram. The properties are categorized under different tabs located to the left in the *Properties view*.
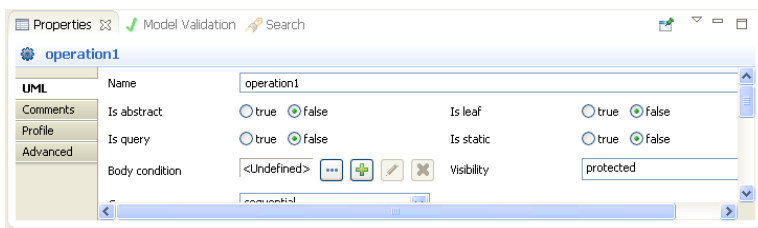

Figure 21: The Properties View

In this view the properties may be viewed and edited, e.g. rename the operation and change the visibility of the operation.

7.2.6 Model Validation view

From the context menu in the *Model Explorer* it is possible to validate the entire model or parts of it (for more details see section Model validation . All warnings and errors appear in the *Model Validation View*, which is a stacked view together with the *Properties View* and the *Search View* at the bottom of the workbench.
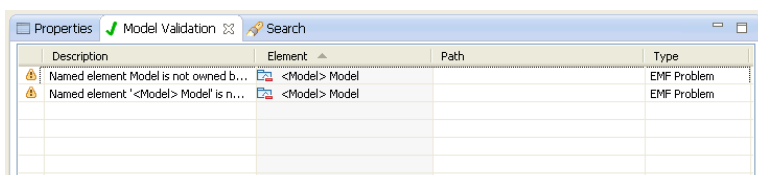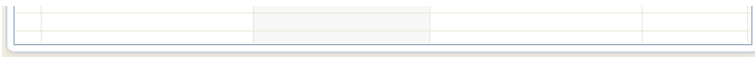
Figure 22: The Model Validation View

The model validation constraints are customizable and how to work with it is described in section Model validation .

7.2.7 Search view

It is possible to do searches on a selected resource in a specific project or in the entire workspace. When the search is finished, the result appears in the *Search View*. Details about specifying searches is described in the Searching section.
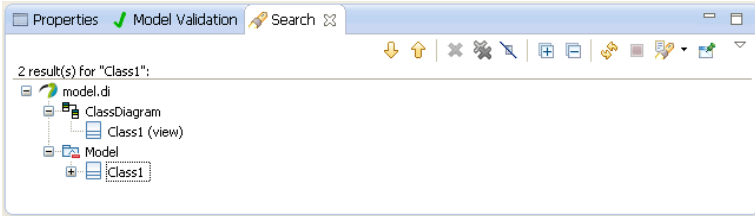

Figure 23: The Search View

The example in Figure 23 shows the result of a model search for *Class1* in the entire workspace.

7.3 Diagram editing in Papyrus

To edit diagrams different editors are available in Papyrus. They have the same basic look and feel. When double clicking on some diagram in the *Model Explorer*, the diagram opens in the editing view. An outline view and a tool palette are also opened. Creating a new diagram in the *Model Explorer* will also open up a diagram editor together with its tool palette and outline view.

7.3.1 Diagram editors

When a diagram editor is opened in Papyrus, three views are opened:

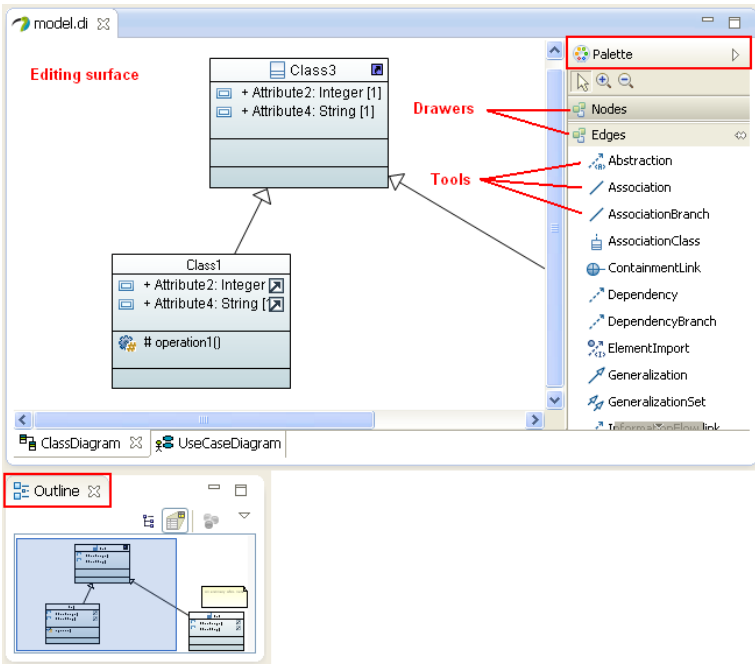- Editing surface
- Palette
- Outline

Figure 24: Parts of a diagram editor (as an example, the class diagram editor is used)

Figure 24 shows the different parts of a diagram editor. In this case the class diagram editor has been chosen as an example. The *Editing surface* is where the diagram editing is taken place. The *Outline view* gives an overview of the entire diagram. The blue shaded part in the *Outline view* shows what is visible in the editing surface. The *Palette* contains *Drawers* and in each drawer there are *Tools* to be used

to add different things into the diagram. In Figure 22, the *Nodes* drawer is closed and the *Edges* drawer is opened. In the *Edges* drawer there are *Tools* to create different types of edges. By clicking on a drawer, it toggles open and close drawer.

7.3.2 Basic tool techniques

Diagrams can be created in different places in the model such as they can be owned by model elements like classes or packages. Diagrams can also be placed on top of the model directly under the model package.

7.3.2.1 Creating diagrams

To create a diagram, right click on the model element that should be the owner of the diagram and select **New Diagram** from the context menu. A new level of menu appears, displaying all types of diagrams that are available to create in this place, e.g. Figure 21 shows the available diagram types that can be created directly on top in the model package.
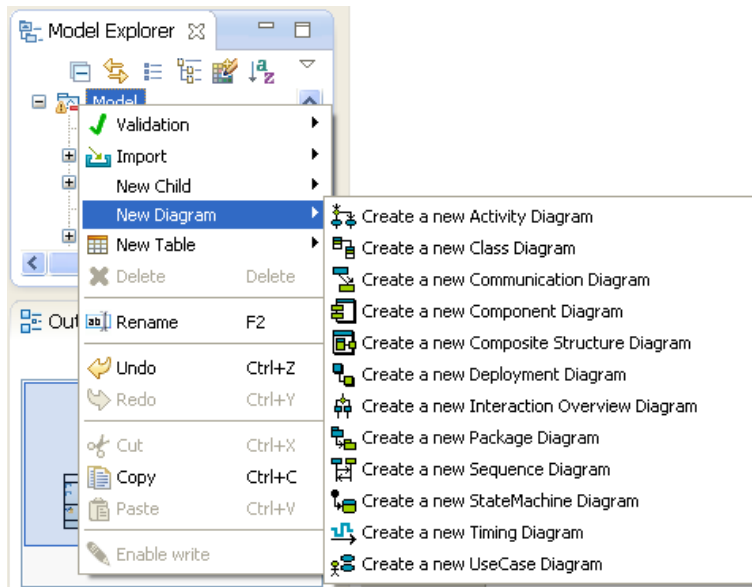

Figure 25: Available diagram types that can be created
directly under the model package

Note! In Figure 25, no adaptation of Papyrus has taken place, hence all diagram types according pops up.

7.3.2.2 Scrolling and panning in diagrams

Scrolling and panning in diagrams can be done by either:

- Use the outline view and click (hold down) and drag the blue shaded area around, which simultaneously pans the editing surface.
- Use the vertical and horizontal scroll bars in the editing surface.

7.3.2.3 Creating an element in a diagram

Elements can be created in a diagram directly, by using a tool from the palette, e.g. to create a class

1. Open a class diagram
2. Open the nodes drawer
3. Click on the class tool
4. Click somewhere in the editing surface
5. Name the class

If an element already exists in the model, just click on (hold down) the element in the *Model Explorer* and drag it to the editing surface. When releasing the mouse key, the modeling element appears in the diagram.

7.3.2.4 Delete and hide

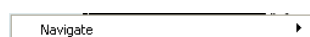In a diagram, elements can be deleted or hided.

Figure 26: Context menu
of an element in a
diagram

Right click on an element in a diagram and do

- **Delete Selected Element** will delete the element from the entire model and also from all diagrams where it was present.
- **Hide Selected Element** will hide the element just in this diagram.

Note! These menu items have short cuts, i.e. instead of using the context menu, just select the element in the diagram and use the **Delete** or **Shift-Delete** keys

7.3.2.5 Formating and validating diagrams

Diagrams can be adjusted and graphically edited to get a nicer look also using the element context menu. From the same context menu it is also possible to validate the model or specific parts of the model.
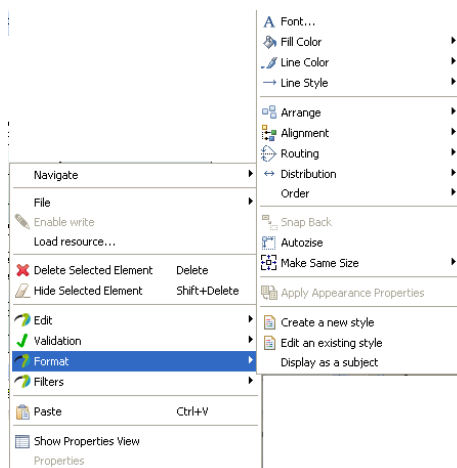

Figure 27: Example of the Format sub
context menu

In Figure 27 several menu items are shown, e.g.:

- **Validation** to validate the model partly or fully
- **Format** to do some advanced formating of the diagram like adjustments, routing, etc. as shown in the sub context menu in Figure 25
- **Filter** to select/unselect parts of symbols that should be visible or not

7.4 UML modeling

With Papyrus, UML models can be created. This is done using different types of diagrams. Modeling elements can be created in these diagrams or directly in the *Model Explorer*. Diagrams are created in the *Model Explorer* and when doing so a diagram editor together with its tool palette and outline view are also opened. Section Diagram editing in Papyrus describes how to work with diagram editors.

The most common modeling elements are:

- Package
- Use-case
- Actor
- Class
- Object

7.4.1 Package

A *package* is a general UML grouping element, comparable to a folder in Windows or a directory in Unix. It is used to bring order in the model. A *package* may have a semantical meaning (e.g. representing a subsystem) and then a UML stereotype, defined in a applied UML Profile, may be added to it (e.g. << subsystem >>). To create a new *package*, right click on the owning element, e.g. the model package and from the context menu select **New Child > Create a new Package**
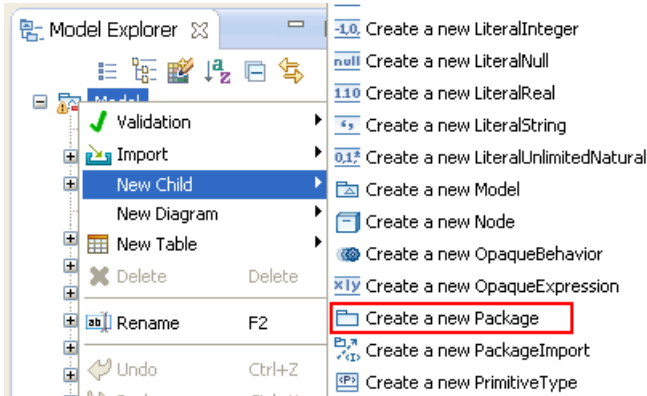

Figure 28: Create a new Package

7.4.2 Use-case

A *use-case* is a functionality in the system. A *use-case* is a model of the dialogue between actors and the system. It should return a result of measurable value to at least one actor. A *use-case* is initiated by an actor to invoke a certain functionality in the system. A *use-case* is a complete and meaningful flow of events. Taken together, all *use-cases* constitute all possible ways of using the system.

To create a new *use-case*, right click on the owning element, e.g. a package and from the context menu select **New Child > Create a new UseCase**
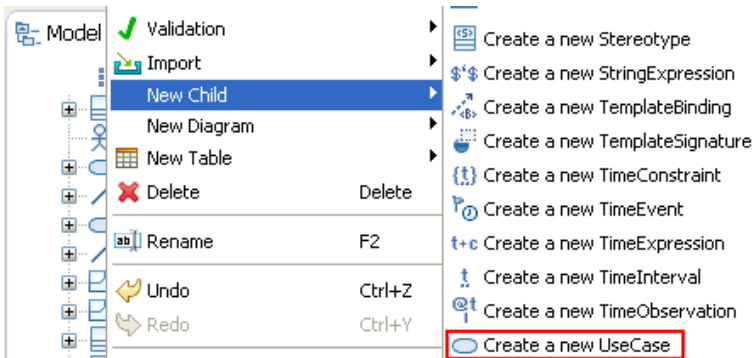

Figure 29: Create a new use-case

7.4.3 Actor

An *actor* is something external to the system, but interacts with it. An *actor* may be a human being or another system. It may be active or passive. An *actor* interacts (active *actor*) or receive (passive *actor*) information from one or several use-cases.

To create a new *actor*, right click on the owning element, e.g. a package and from the context menu select **New Child > Create a new Actor**
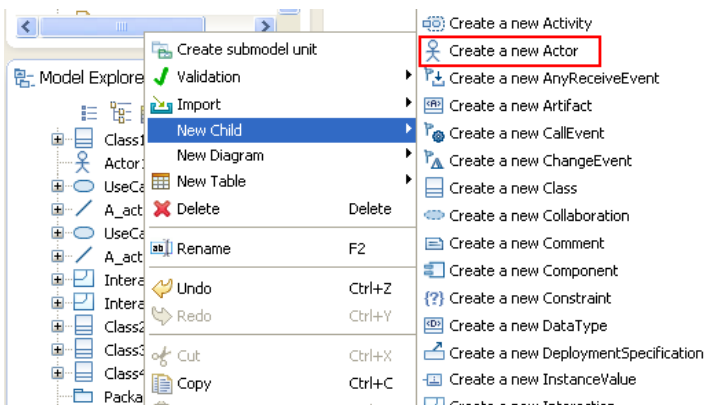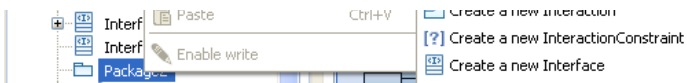
Figure 30: Create a new actor

7.4.4 Class

A *class* is an extensible template for creating objects, providing initial values for state (member variables, attributes) and implementations of behavior (member functions, methods, operations).

Collectively attributes define the structure of a *class*. A class may have any number of attributes or none. Attributes are typically implemented as variables. An attribute has a type, which tells us what kind of attribute it is. Typical types of attributes are integer, Boolean, real, and enumeration. These types are called primitive types. More complex types are defined by other *classes*.

Collectively operations define the behavior of the *class*. A *class* may have any number of operations or none. Operations are implemented as functions or procedures.

To create a new *class*, right click on the owning element, e.g. a package and from the context menu select **New Child > Create a new Class**
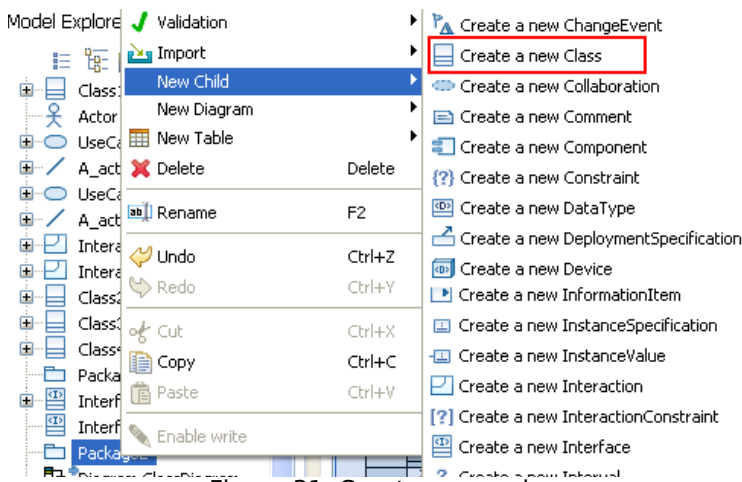

Figure 31: Create a new class

7.4.4.1 Attributes on classes

When a *class* is created, *attributes* can be added to it by using the context menu of the *class*. To create a new *attribute* on the a *class* select **New Child > Create a new Property** from its context menu.
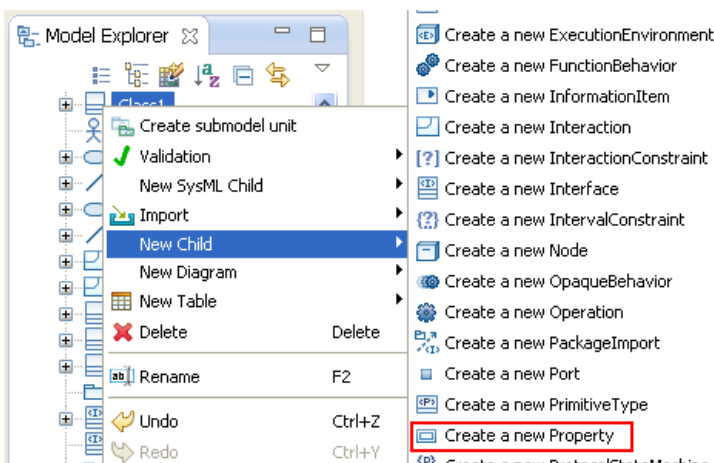

Figure 32: Create a new attribute

The visibility, type and default value of the attribute are set in the properties view when the attribute is selected.

7.4.4.2 Operations on classes

When a *class* is created, *operations* can be added to it by using the context menu of the *class*. To create a new *operation* on the a *class* select **New Child > Create a**
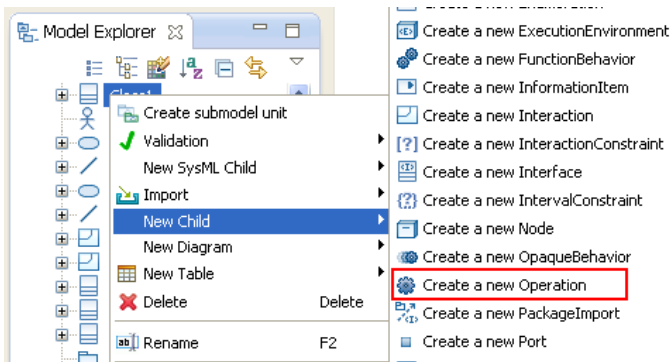
new Operation from its context menu.

Figure 33: Create a new operation

The visibility, arguments and return type of the operation are set in the properties view when the operation is selected.

Regarding the arguments and return type of the an operation, select the ⊞ key by the *Owned parameter* field.
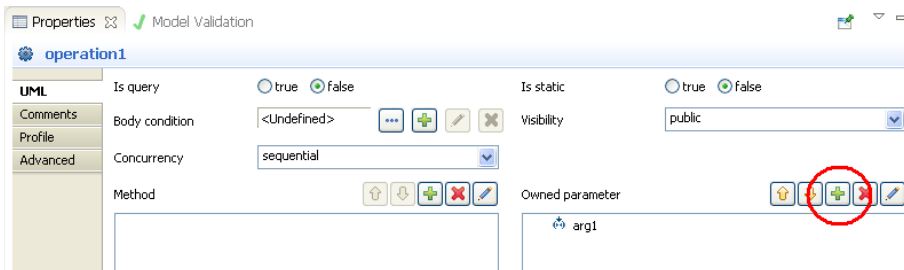

Figure 34: Create a new argument

Then the following window pops up and from the drop list in the *Direction* field, select the direction of the argument. In the *Name* field the name of the argument is written and the type is defined in the *Type* field.
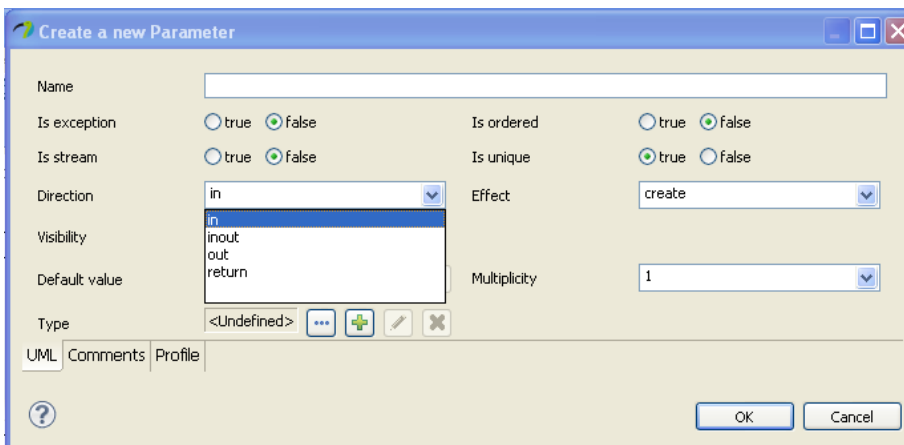

Figure 35: Select the argument�s direction

The direction *return* defines the return type of the operation. Only one argument can have the return direction.

7.4.5 Object

An *object* is an instance of a class. In UML it is called and *InstanceSpecification*, which is a more general term since it can be used for instances of other classifiers than classes.

To create a new *object*, right click on the owning element, e.g. a package and from the context menu select **New Child > Create a new InstanceSpecification**
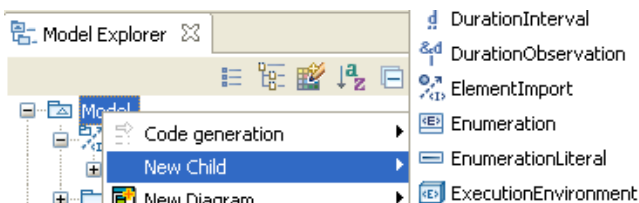
Figure 36: Create a new object

The class to be instanciated is selected by clicking on the ⊕ key by the *Classifier* field in the *Properties* view of the *InstanceSpecification*. This will open the *Classifier* pop-up window (figure 37), where the class to be used is selected.
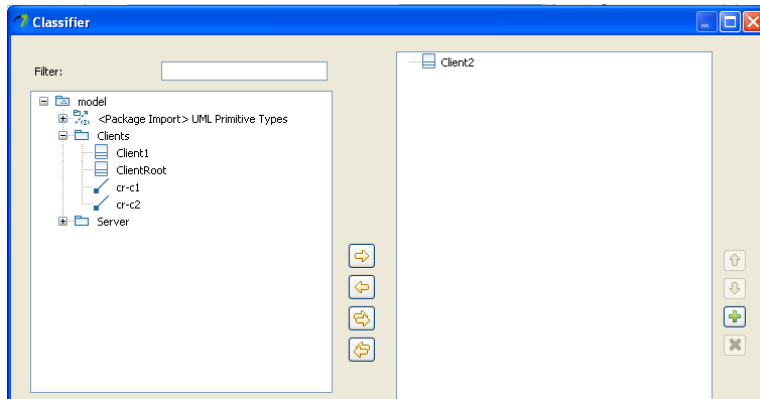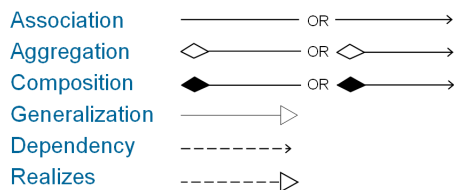

Figure 37: Classifier pop-up window

7.4.6 Relationships

There are different types of relationships that can be used in diagrams, hence in the model between different modeling elements.

Navigability can be unidirectional or bidirectional for *Association*, *Aggregation* and *Composition*.

Association specifies peer-to-peer relationships between model elements, e.g. if a Class-x has an attribute of type Class-y, it can be viewed in a class diagram as and *Association* between Class-x and Class-y.

Aggregation is used to model a whole/part relationship between model elements. The part element can exist without the whole. *Aggregation* causes the generated code to contain the aggregate either by reference or by value, depending on the details of the relationship. E.g. to model an aggregation, the aggregate (Department) has an aggregation association to its constituent parts (Employee). A hollow diamond is attached to the end of an association path on the side of the aggregate (the whole) to indicate aggregation.

Composition is an aggregation with strong ownership, i.e. when the container is destroyed, all of its composite objects are destroyed as well.

Dependency is a relationship in which one model element uses another. *Dependency* may exist between classes if a message is sent from one class to the other or if one class mentions the other as a parameter to an operation. *Dependency* may exist between packages if one package is dependent on another.

A *Dependency* relationship causes a class to be generated with inclusions or references to another class.

A *Generalization* relationship causes a class to be generated as a subclass of another class.

The *Realizes* relationship specifies that, e.g. an implementation realizes a specification. The *Realizes* relationship does not affect the code.

To create a relationship between two modeling elements, use the tool palette in the diagram editor, e.g. to create an *Association* between two classes, select the *Association* tool in the tool palette, click on the source element and then click on the destination element as described in figure 38.
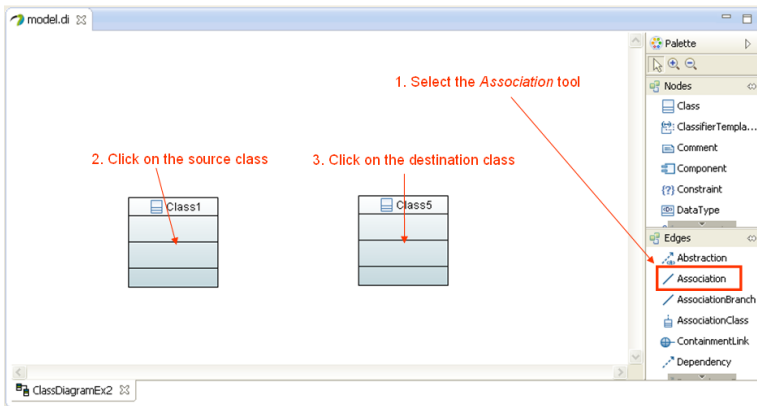

Figure 38: Create a new Association

In the *Edges* drawer in the tool palette, all available relationships are shown. To create a *Generalization* relationship, select the *Generalization* tool from the tool palette and follow the same procedure as described.

The *Aggregation* and the *Composition* relationships are a special kind of an *Association* relationship. To create any of these, an *Association* relationship needs first to be created. Then select the created *Association* and in the properties view, change the *Aggregation* field at the appropriate end of the *Association* to *shared*(if an *Aggregation* is desired) or to *composite* (if a *Composition* is desired). Figure 39 shows how to do it.
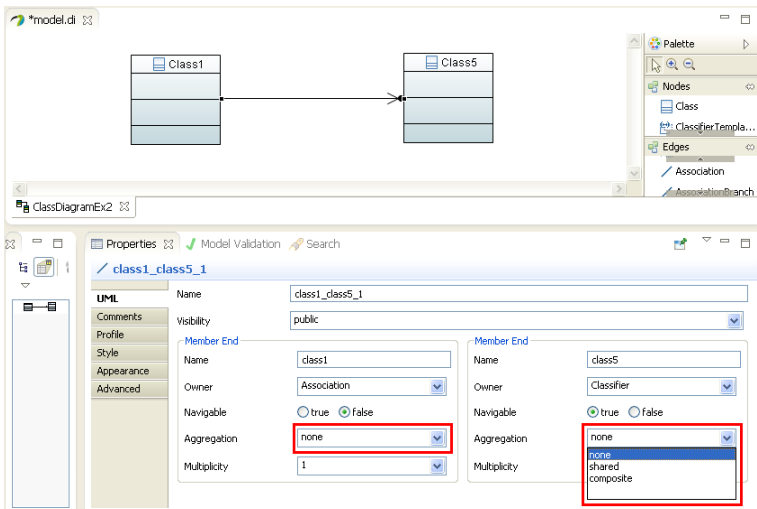

Figure 39: Create a new Association

When the *Association* is created, the *Aggregation* field is set to *none* by default. When doing the change at the destination end (as in figure 35), the diamond shows up at the source end of the relationship.

7.4.7 Diagrams

UML has many different types of diagrams to capture all different aspects of a system. To capture and refine requirements, diagrams related to use-cases are used. To specify the architecture and design, diagrams related to classes and packages are used. To specify the implementation, state and activity diagrams are used, etc.

The different diagrams in UML 2 are shown in figure 40 and here they are structured after diagram type. In the following of this section they are organized how they are used.
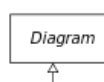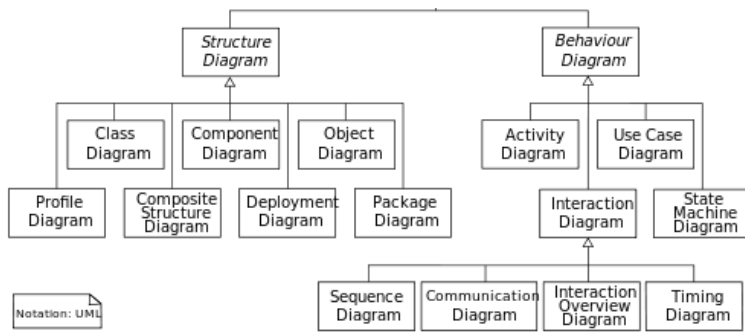
Figure 40: UMl 2 diagram types

Note! In Papyrus class diagrams are also used as object diagrams.

7.4.7.1 Diagrams related to use-cases

When working with requirement capture and refinement, use-case modeling is used and any or all of the following diagrams can be used:

- Use-case diagram
- Activity diagram
- Interaction diagram
    - Sequence diagram
    - Collaboration diagram

A *use-case diagram* describe how different *Actors* use different functionality of the system. Implicitly, it also define the system boundary, since it shows what should be performed by the system and what exists outside the system. The elements used in *use-case diagrams* are:

- **Actors** interact with, but are outside the system.
- **Use-cases** are some functionality that are performed by the system.
- **Relationships** between elements.

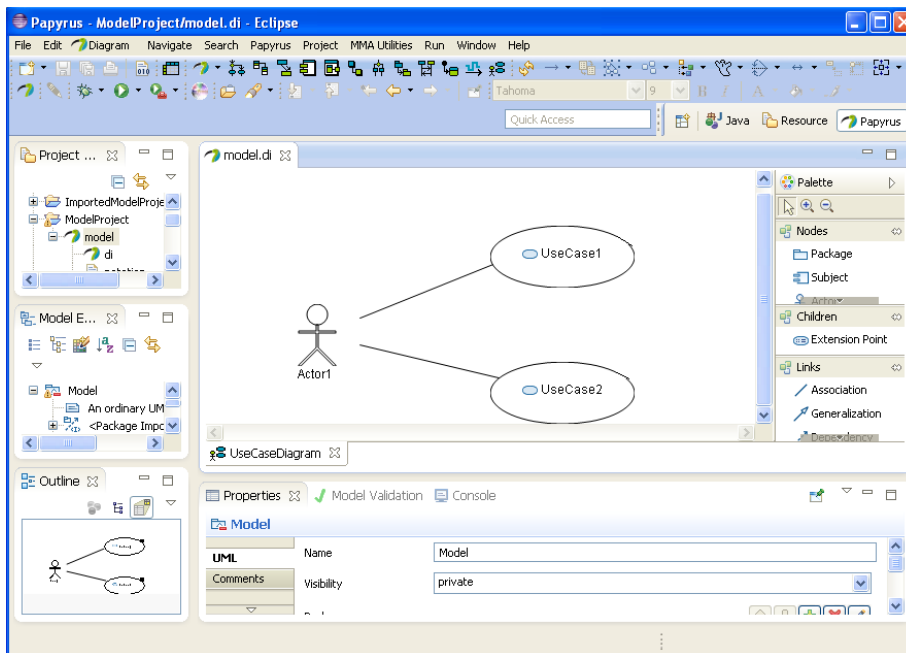A diagram may depict all or some of the use-cases of a system.

Figure 41: Use-case diagram

A *use-case* interacts with an *actor* and perform something useful for that *actor*. A *use-case* exist because of its main flow, but all odd cases and error situations have to be specified. A *use-case* has a black box and a white box view. The black box view is preferably described in plain text or by using activity diagrams. The white box view is described by one or several sequence diagrams.

All *use-cases* together span the entire functionality of the system. *Actors*, *use-cases* and *use-case diagrams* are owned by packages (general UML packages or model packages).

To create a *use-case diagram*, right click on the owning package and select **New Diagram > Create a new UseCase Diagram** from its context menu.

Diagram > Create a new UseCase Diagram from its context menu.

An *activity diagram* is a kind of behavioral diagram and shows flow of control from activity to activity. It is used to specify a use-case black box view. It can also be used to specify a flow chart for a class operation.

The main elements in an *activity diagram* are:

- Initial and end states
- Activities
- States
- Transitions
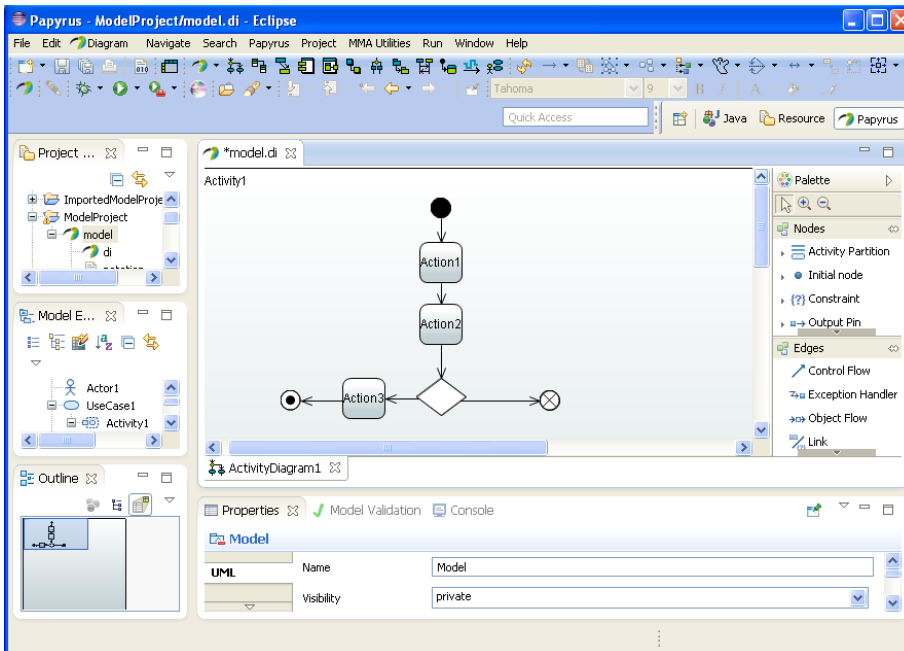- Synchronization lines
- Decisions
- Partitions

Figure 42: Activity diagram

The *activity diagram* is preferably used to specify the black box behavior of a use-case instead of using plain text. It may also be used to specify a flow chart for a class operation. *Activity diagrams* are owned by use-cases or classes.

To create an *activity diagram*, right click on the owning use-case or class and select **New Diagram > Create a new Activity Diagram** from its context menu.

Interaction diagrams are used to specify how different modeling elements interacts. Here two types are described, the *sequence diagram* and the *communication diagram*.

A *sequence diagram* describes the interactions between elements as a time ordered set of messages. One or several *sequence diagrams* are used to specify the white box view of a use-case.

Sequences involving collaborating elementsThe main elements in a *sequence diagram* are:

- Class instances (objects)
- Life lines
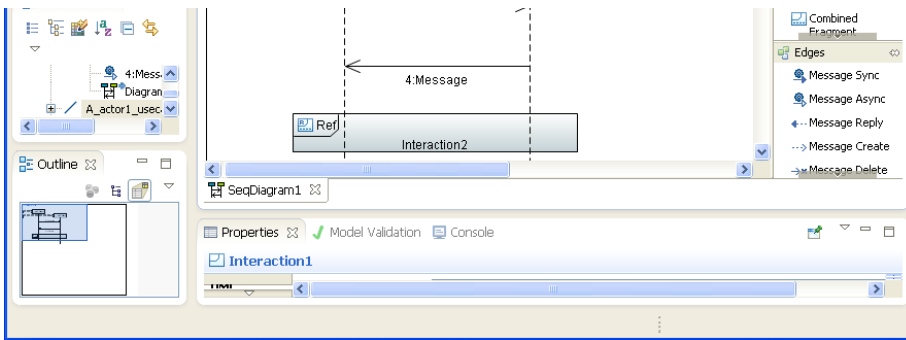- Messages
- Combined fragments
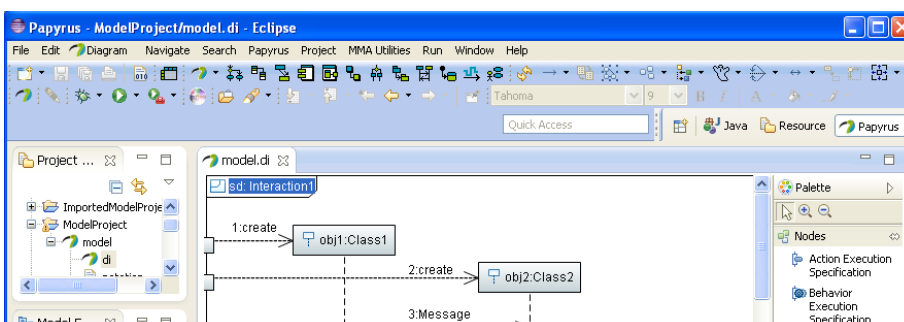
Figure 43: Sequence diagram

The example (Figure 43) describes Interaction1, two objects (instances of Class1 and Class2) are created and interacts by messages. The time goes down along the life lines. In the bottom, there is a combined fragment of type "Ref" which is a reference to another interaction, Interaction2, meaning that the sequences in that interaction are executed.There are a lot of combined fragment types, e.g. type "Loop" specifies a loop, type "Alt" specifies alternatives, etc. All combined fragment types are defined in Unified Modeling Language (UML) version 2.4.1

Sequence diagrams are owned by use-cases or communication diagrams (see below).

To create a *sequence diagram*, right click on the owning use-case or communication diagram and select **New Diagram > Create a new Sequence Diagram** from its context menu.

Communication diagrams show the lines of communication among a set of objects to accomplish a specific purpose. They act as the framework for sequence diagrams and define access paths between elements. i.e. *communication diagrams* are used to specify a use-case's white box communication channels between elements in the system.

The main elements in a *collaboration diagram* are:

- Class instances (objects)
- Access paths
- Messages
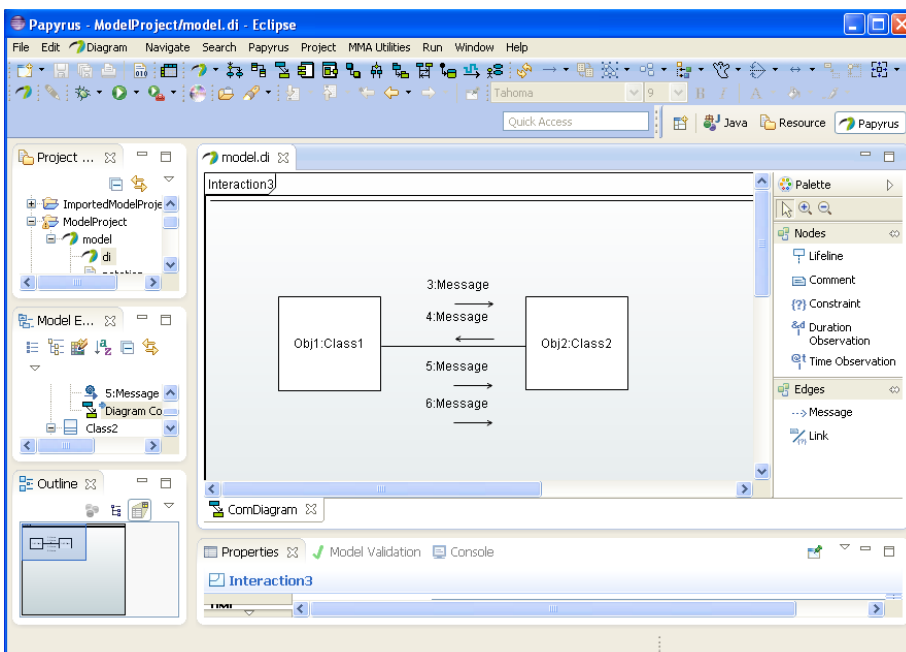

Figure 44: Communication diagram

The example (Figure 44) describes access paths between objects and which messages are passed in these paths. *Communication diagrams* are owned by use-cases.

To create a *communication diagram*, right click on the owning use-case and select **New Diagram > Create a new Communication Diagram** from its context menu.

7.4.7.2 Diagrams related to classes

When modeling classes, any or all of the following diagrams may be used:

- *Class diagrams*
- *Composite structure diagrams*
- *State diagrams*

As described above, activity diagrams may also be used to specify a flowchart for a class operation.

Class diagrams depict static views of the system. A *class diagram* may represent all or part of the class structure of a system. Typically there are many *class diagrams* in a model. Usually one or many *class diagrams* are used to specify the inheritance structure in the system. *Class diagrams* may also be used to define dependency rules between packages.

The main elements in a *class diagram* are:
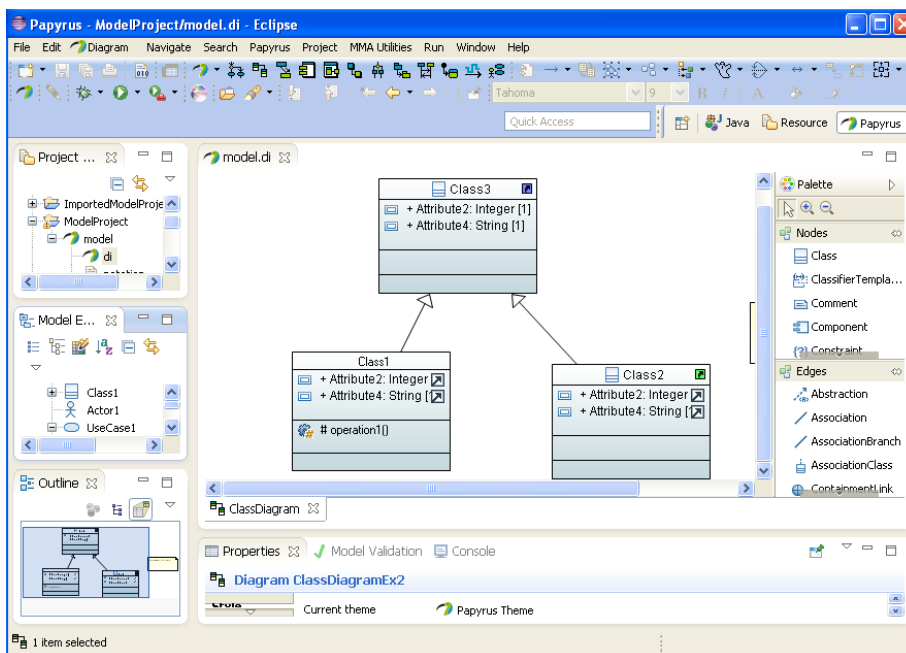
- Packages
- Classes
- Relationships

Figure 45: Class diagram

The example (Figure 45) shows a *class diagram* used to specify an inheritance structure between classes. Note: The *Generalization* relationship and the indication of the inherited attributes in Class1 and Class2.

Class diagrams are owned by ordinary UML packages or model packages.

To create a *class diagram*, right click on the owning package and select **New Diagram > Create a new Class Diagram** from its context menu.

The *composite structure diagram* specifies structure classes contents, i.e. how the class uses *roles* (instances from other classes) and how they are connected to fulfill its responsibility.

The main elements in a *composite structure diagram* are:

- Contained roles (instances of classes)
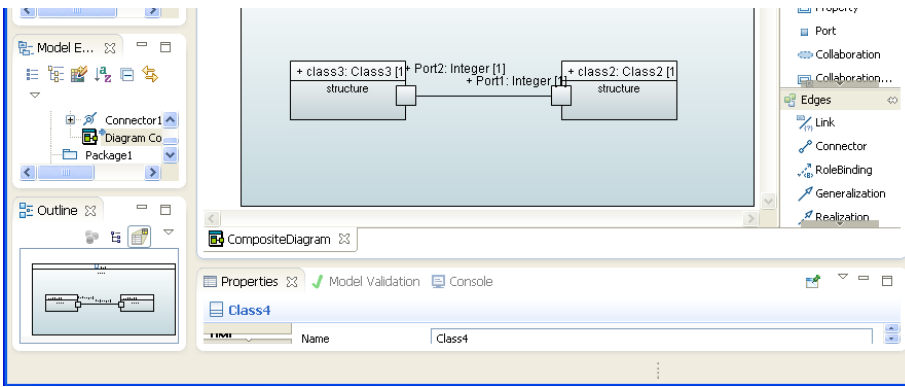- Ports (interface objects)
- Connectors
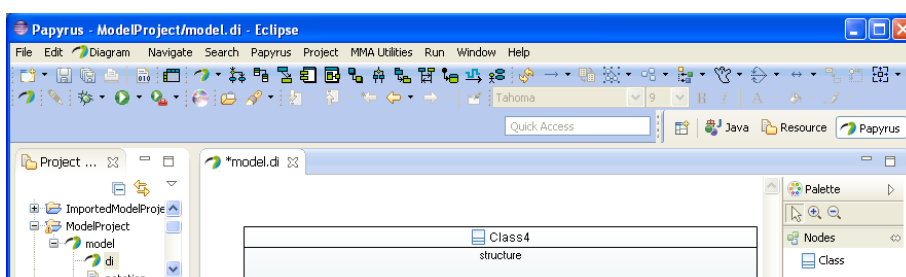
Figure 46: Composite structure diagram

The example (Figure 46) shows a *composite structure diagram* used to specify the structural contents of Class4. Note: Class4 uses one instance (class2) of Class2 and one instance (class3) of Class3 and they are connected between Class2/Port1 and Class3/Port2.

Composite structure diagrams are owned by structured classes.

To create a *composite structure diagram*, right click on the owning class and select **New Diagram > Create a new Composite Structure Diagram** from its context menu.

The *state machine diagram* specifies the behavior of a class. It is used when the class is state rich, i.e. has an event driven behavior. If the class has no states, activity diagrams can be used.

The main elements in a *state machine diagram* are:

- States
- Transitions
- Effect code
- Triggering events
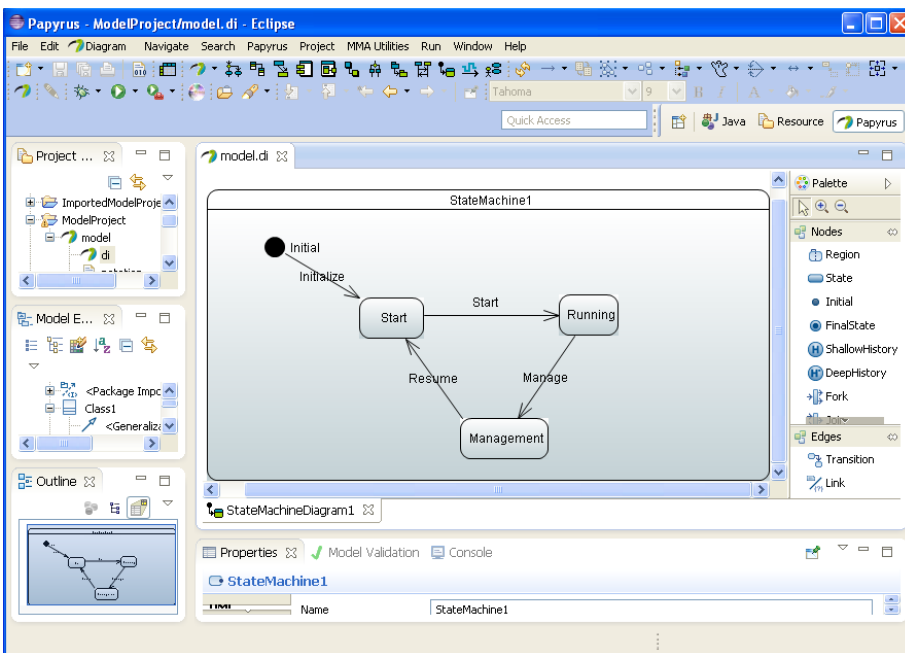- Pseudo states, e.g. initial, final and choice points

Figure 47: State machine diagram

The example (Figure 47) shows a *state machine diagram* that has an initial pseudo state, three states and transitions between them. On each transition (except for initialize), a *triggering event* is specified, which defines the event that makes the transition to be taken. Transitions and states may have effect code, which specify detailed behavior to be executed when an associated transition is taken.

State machine diagrams are owned by classes.

To create a *state machine diagram*, right click on the owning class and select **New Diagram > Create a new State Machine Diagram** from its context menu.

7.5 Internationalize a model

The internationalization is the way to manage your model with a specific language by labels. Like Eclipse, the internationalization is managed by the 'properties' file which is defined depending on the language as following (with for example, english language in US country):

- *modelName*_en_US.properties
- *modelName*_en.properties
- *modelName*.properties

This 'properties' file is managed directly by the Papyrus model and created (if not existing) when a label is modified.

7.5.1 How to use internationalization in model

The internationalization can be used by checking a Papyrus model preference. The language of the internationalization is initialized by the application language but can be modified as well.
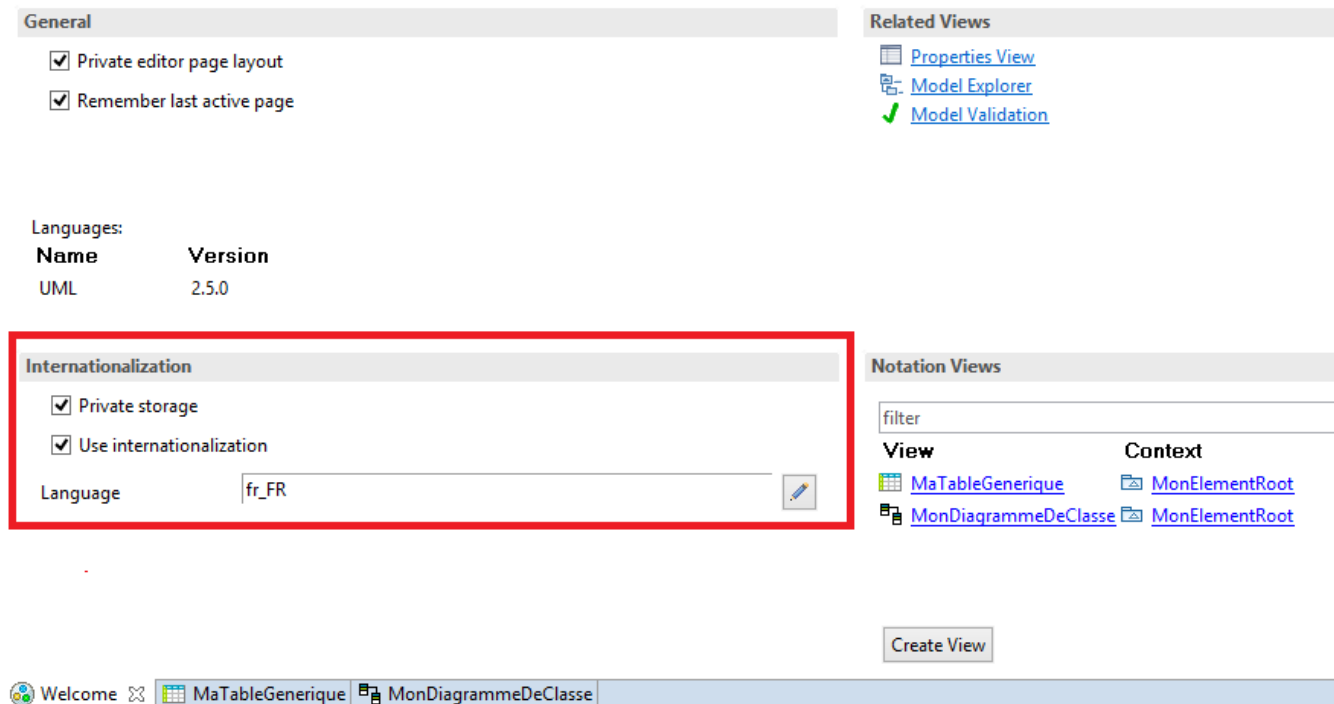

Figure 48: Internationalization preferences management

7.5.2 How to edit labels

The internationalization of UML elements, diagrams and tables can be edited by the properties view with the 'Label' field:
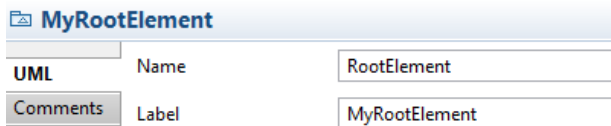

Figure 49: 'Label' field in properties view

If the internationalization preference is active and object to modify is internationalized, it can be modified directly by the direct editor too as following:
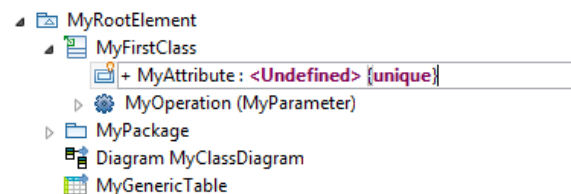

Figure 50: Edit label by direct editor

7.5.3 How the labels are managed in properties file

Each element which have a label must be written in the properties file like following:

- If this is a diagram: **_labelDiagram_** _DiagramOwnerQualifiedName_ **_label_** _DiagramName = DiagramLabel_
- If this is a table: **_labelTable_** _TableOwnerQualifiedName_ **_label_** _TableName = TableLabel_
- If this is an UML NamedElement: **_label_** _NamedElementQualifiedName = NamedElementLabel_

7.5.4 Sub-models

The sub-models are managed with the properties files. Indeed, when an object is created as sub-model, the key representing this object or its descendance in the properties files must be moved in sub-properties files corresponding to the object of the sub-model. During this sub-model creation, the qualified names of the keys, that are moved, change, the new qualified names are calculated from the root of the sub-model.

7.6 UML RT modeling

When creating models UML is used. Since UML is general-purpose modeling language in the field of software engineering, it is possible to adapt UML to specific domains. This is done by creating and applying UML profiles. When using UML for RT modeling with Capsules and Protocols, the UML RT profile is applied. When a profile is applied we can say that Papyrus has been specialized. There is a specific use-case in Papyrus to develop UML profiles and when doing so a domain specific modeling language is defined. This use-case is described in section UML profiling in this user guide.

7.6.1 Additional modeling elements

7.6.1.1 Capsule class

7.6.1.2 Protocol class

7.6.2 Using C++ in a model

7.6.3 C++ service library

7.6.3.1 Sending messages

7.6.4 Transformation from model to code

7.6.5 Edit the generated code

7.6.6 Compiling and linking the generated code

7.6.7 Using external libraries

7.6.8 Running the system

7.7 Papyrus in a team environment

7.7.1 Model fragmentation

7.7.2 Source configuration management

7.7.3 Compare and Merge

7.8 Model validation

7.8.1 Object Constrain Language (OCL)

7.8.2 Defining constraints using OCL

7.9 Searching

7.10 Sample models

-

In the Papyrus installation directory There are several sample models

7.10.1 Class model with inheritance

7.10.2 Send and receive data

7.10.3 Interprocess communication

7.11 UML profiling

TBD include the information in the user guide "About UML profiling"

8 Support

To report bugs, suggest improvements, view the status of the Papyrus project, discuss different Papyrus subjects, etc. please use the following references:

- The Papyrus project home page
- The Papyrus discussion forum
- Proposals for Papyrus improvements
- Bugzilla_to_report_bugs

9 References

1. eclipse.org
2. EMF model
3. Eclipse download page
4. Unified Modeling Language (UML) version 2.4.1
5. System Modeling Language
6. Modeling and Analysis of Real-Time and Embedded systems