list

Service Outils Logiciels

Saclay, 04/11/2009

REF. : DTSI/SOL/09-0231/RS

Palette Customisation in Papyrus MDT

Par

Remi Schnekenburger

DTSI/SOL/LISE (CEA)

IDENTIFICATION :   Rapport DTSI/SOL/09-0231/RS

Title :   : Customisation in Papyrus MDT

Authors :   Remi Schnekenburger
Keywords :   Papyrus, MDT, Eclipse, Palette, Customisation

Unit :   DRT/LIST/DTSI/SOL/LISE

Abstract/Context   Eclipse project

	REDACTEURS *WRITERS*	VERIFICATEUR *CONTROLLER*	CHEF DE SERVICE *SERVICE HEAD*
NOM	Remi SCHNEKENBURGER	F. TERRIER	Jan STRANSKY
DATE			
SIGNATURE			

Diffusion : Papyrus website: http://www.eclipse.org/modeling/mdt/?project=papyrus

1. Introduction

This report contains developer documentation for the implementation of diagram palettes in Papyrus.

In this document, Papyrus will always refer to the Eclipse MDT implementation (see http://www.eclipse.org/modeling/mdt/?project=papyrus).

The various step of the GMF process used to produce the GMFgen file of a diagram are not reminded in this document. This document focuses on modification directly made over the GMFgen file. For more information regarding GMF, the user should read dedicated document available on GMF website (http://www.eclipse.org/modeling/gmf/).

2. Installing the development environment

The basic steps first:

- ☐ Install the "Modelling" bundle of Eclipse (http://www.eclipse.org/downloads/)
- ☐ Install the SVN client over Eclipse (http://www.eclipse.org/subversive/)
- ☐ Get Papyrus sources from its repository

(svn+ssh://dev.eclipse.org/svnroot/modeling/org.eclipse.mdt.papyrus)

Do not forget to download gmfgen extension plugin and other development tools located in trunk/plugins/developers:

☐org.eclipse.papyrus.codegen (specific code generation)
- ☐ org.eclipse.papyrus.def (code generation templates)
- ☐ org.eclipse.papyrus.gmfgenextension (gmfgen model extension and tools)

For an easier access (directly usable in your development environment without starting a test workbench), it may be better to export a build of "codegen" and "gmfgenextension" in your development Eclipse installation:

- ☐ Open the plugin.xml file (from either "codegen" or "gmfgenextension")
- ☐ Select "export wizard" in the overview tab
- ☐ Select the plugins to export ("codegen" and "gmfgenextension" here)
- ☐ Choose "Install into host repository" as shown in the picture below.
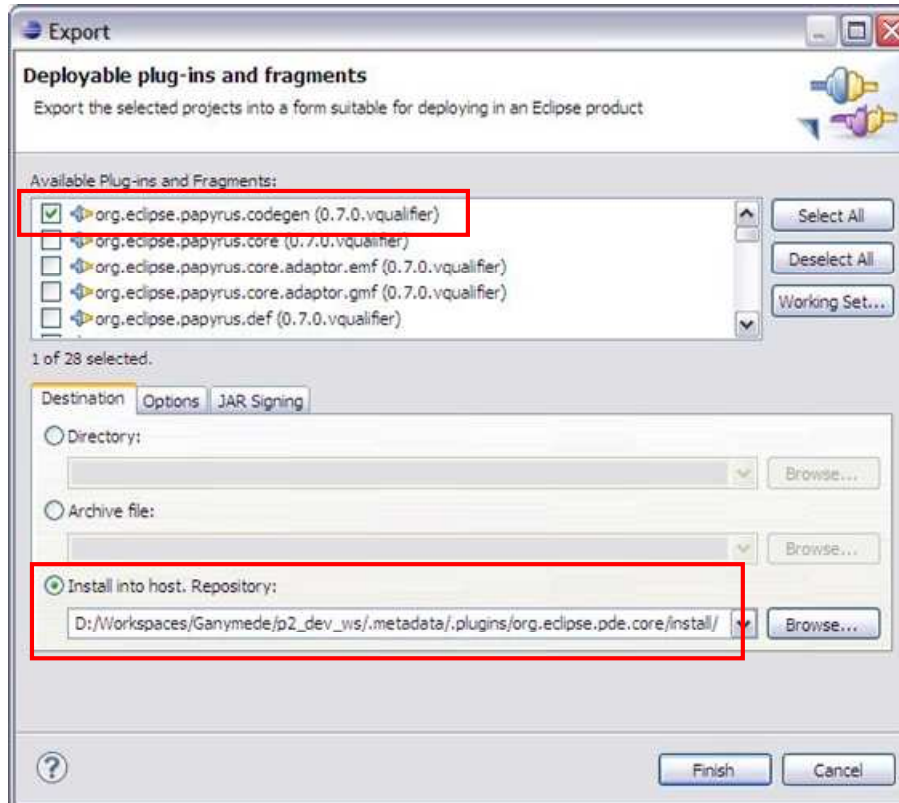
3.   Overview of the Papyrus extension for GMFGEN model

An extension point exists in GMF framework to define palettes in a plug-in. This extension point lets users complete or modify an existing palette defined in another extension. This extension point is divided into 3 aspects.

A first aspect is the "pre-definition" of tool entries. These predefinitions can be later used in other editors. The zoom + and -, the selection tool, the geoshapes are defined this way. So each gmf-based editor is able to use these entries. If you don't want these basic predefined entries, you even have to remove them using the palette extension point. They are present by default.

The second aspect is using the predefined entries to describe the palette. For example, the palette A can use a tool with a particular id predefined in a palette B, it removes another one from palette C, etc.

The last aspect is the standard definition of tool entries. It is similar to the first aspect. The only difference is a simple field in the extension point that indicates if the specification of the tool entry is only a definition, or both definition and usage.

By default, GMF-generated diagrams do not use this extension point. They generate a pseudo factory. As the editor is created, the palette root is programmatically filled with entries. This method has 2 disadvantages:

_____

- Standard palette does not have the same behaviour as other palettes. As it is not provided using a standard provider/factory couple, it is not possible to manipulate it like other palettes. It is easier considering the standard palette like other ones. This way, customization tools are directly manipulating custom elements; no special treatment has to be done for the standard palette.

- The standard provides standard tools! In fact, it provides the basic elements to create a class, a package, a comment.... If these definitions are embedded in the code, it is not possible to use them using palette extension point. How can the user remove an element that has not be define using the standard extension point!

The goal of the document is to describe how overwrite the standard generation using Papyrus tools.

4. Modifying an existing GMFgen model

The default GMFgen generated from the various input models already contains a definition of the palette. This definition is not complete. In fact, it does not contain the icon definition for the tool entries (the group, node and link tools do not have their *smallIconPath* and *largeIconPath* filled).

An action has been added to the GMFgen editor context menu. It works on Palette/Tool Group /Tool entry elements. It generates (or tries to) the two fields for the icon path. For the palette elements, it delegates the generation to all subgroups. For the Group element, it generates the icon for the group and its sub tools. For the tool entries, it generates the icon fields for the selected element only.

It currently only works for elements directly linked to the UML model. For example, it works correctly with *Comment* Node, but not with the Comment Links. In the class diagram, Comment links and Constraint links were the only 2 elements that were not filled correctly. If an element has already been filled, it does not override previous values.
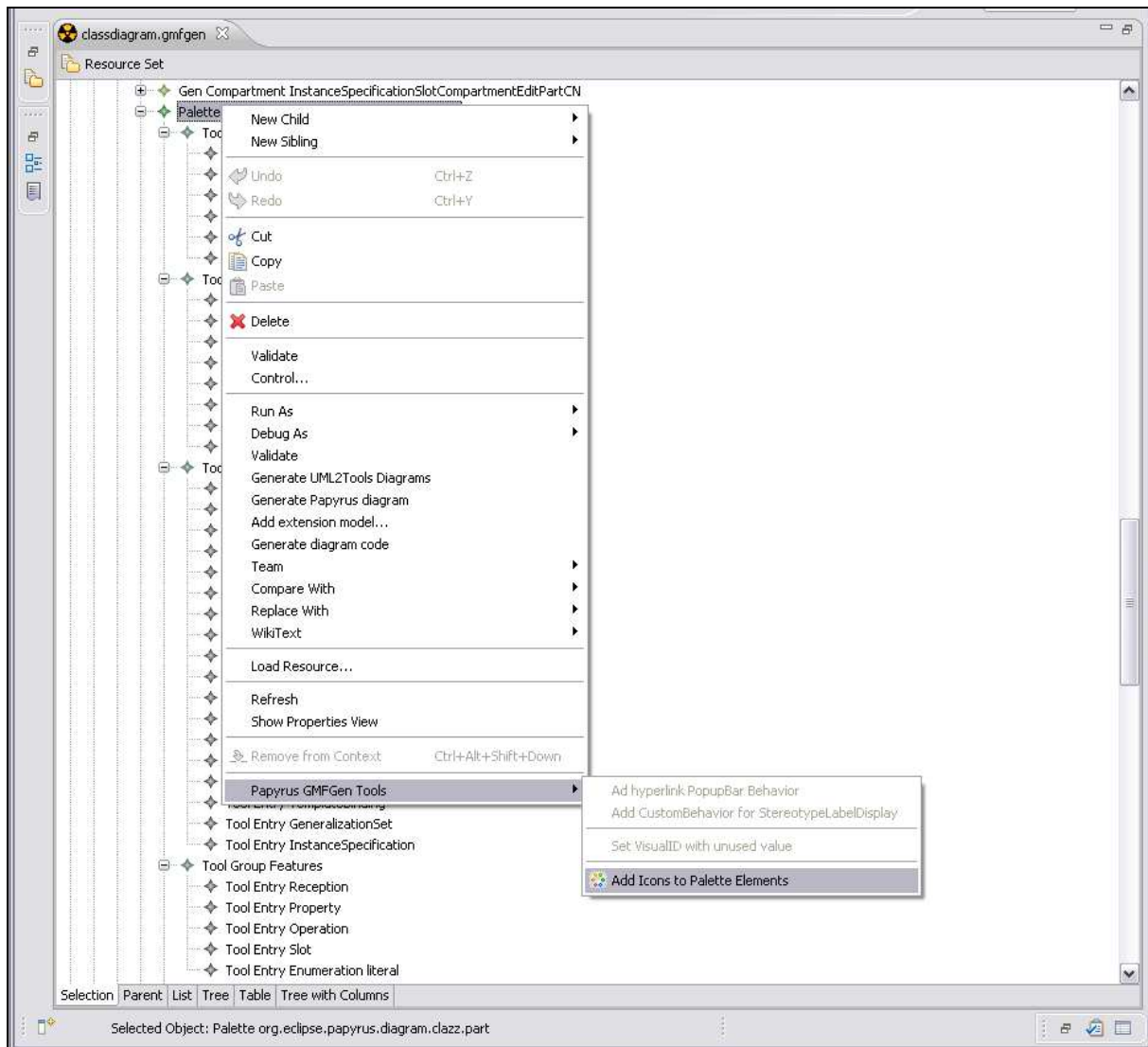
5. Generating code

After icons fields are valued, standard Papyrus code generation is made. The only modifications to apply are the removal of the already existing extension points in the plugin.xml file. In fact, xpand has sometimes problems to override previous extension points. In the class diagram, the only palette definition was the removal of the *note* tool. Once this extension has been removed, generation is generally safe (from an xpand point of view!).

Thanks to the *private* and *package* visibility used in GMF editor basic class, it was not possible to complete or modify the menu provided in the palette! So some code has been duplicated and modified in the custom class defining the class diagram editor. In fact, only one line was modified.

_____

```
<code>

viewer.setContextMenu(new PapyrusPaletteContextMenuProvider(viewer));

</code>
```

This new context menu overrides the standard palette menu provider, to add the possibility to select the list of palettes that should be visible or not.

Some custom code was also added to the diagram editor class, it has been added in the code generation instead of the custom class. This code concerns the preference listeners. In fact, the editor is listening to the palette provider, so it knows when preferences have change for the palette. If the palette display selection has changed, it can update the palette viewer.

For archive, code that has been added:

Editor now implements a new interface

```
<code>

public class UmlClassDiagramForMultiEditor extends
org.eclipse.papyrus.diagram.clazz.part.UMLDiagramEditor implements
IProviderChangeListener {

</code>
```

interface implementation

_____

```
<code>

/**

* @generated

*/

public void providerChanged(ProviderChangeEvent event) {

// update the palette if the palette service has changed

if (PapyrusPaletteService.getInstance().equals(event.getSource())) {

PapyrusPaletteService.getInstance().updatePalette(getPaletteViewer().getPal
etteRoot(), this,

getDefaultPaletteContent());

}

}

</code>
```

Listener add/remove

lines added in constructor

```
<code>

/**

* @generated

*/

public UMLDiagramEditor() {

super(true);



// adds a listener to the palette service, which reacts to palette
customizations

PapyrusPaletteService.getInstance().addProviderChangeListener(this);

}

</code>
```

dispose method updated

```
<code>

/**

* @generated

*/

public void dispose() {

// remove palette service listener

// remove preference listener

PapyrusPaletteService.getInstance().removeProviderChangeListener(this);



super.dispose();

}

</code>
```

palette menu provider

```
<code>

protected PaletteViewerProvider createPaletteViewerProvider() {

[...]

}

</code>
```

6. Customizing palettes

This concerns the local definition of palettes. Local definition means that the xml file defining the palette is stored in a predefined place on the disk. These palettes are not provided using a plugin, but are referenced in the workbench using preferences. This way, every user can define its own palette without creating a plugin, export it and install it. The local definition should be exportable using a wizard. Such wizard should extend the new plugin wizard. This way, users could define locally a palette, and could publish it with a plugin.

Another way of exchanging the palette should be an import/export XML palette definition. This would allow users to use their local palette definitions through several different workspaces. In fact, eclipse preferences are not always kept from one workspace to another. An example of this feature is the java code formatter. It is possible to exchange format profiles with people using a xml file.

A customized palette can also provide new tools. These new tools in Papyrus can be for example the creation of elements with stereotypes already applied on them. This let the user adapt the standard Papyrus tool to a particular domain of modeling. More explanation on this feature can be found in chapter 6.2

6.1. Toggling palette visibility

When selecting the "customize" menu of the palette, a new window appears. This let users select which palettes are displayed, and which one are hidden for the current kind of diagram. For example, if the user is currently using the class diagram editor, all palettes for the class diagram will be shown. User can check or uncheck the palettes to toggle the visibility of the palette.
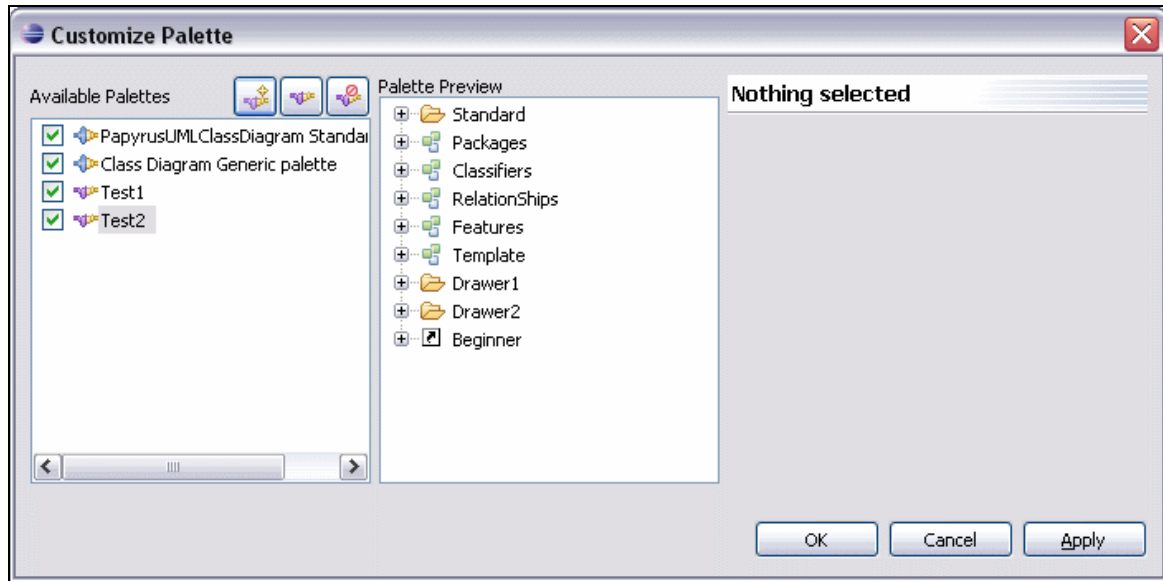
The icon describes a palette defined by the GMF standard palette extension point. In this case, 2 plugins have contributed to the palette for the class diagram. The first one is the one generated from the GMFgen file. The second one comes from a second plugin.

The icon describes local palettes. These palettes are only available in the current workspace, as they are defined using the eclipse preferences system.

The check box on the left side indicates that the palette is visible or not.
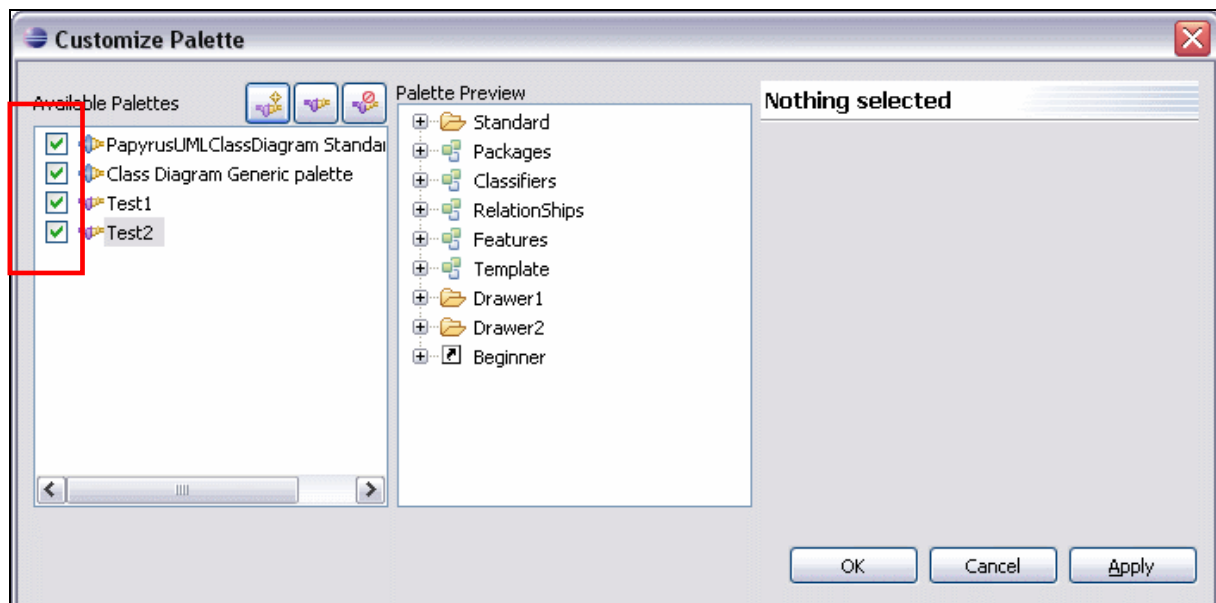
6.2. Defining a new palette

Local definitions of palettes can be managed using the button above the list of available palettes. Some of these buttons are available only when a local palette is selected in the window. A local palette is the palette with the sign

Creating a new local definition of a palette

This can be achieved by selecting the button . A new wizard appears, it is divided into 2 pages.

The first page asks for information about the palette. The most important thing here is to give a name to the palette. This name is the information shown when the user wants to toggle the palette visibility, the list of available palettes, etc.
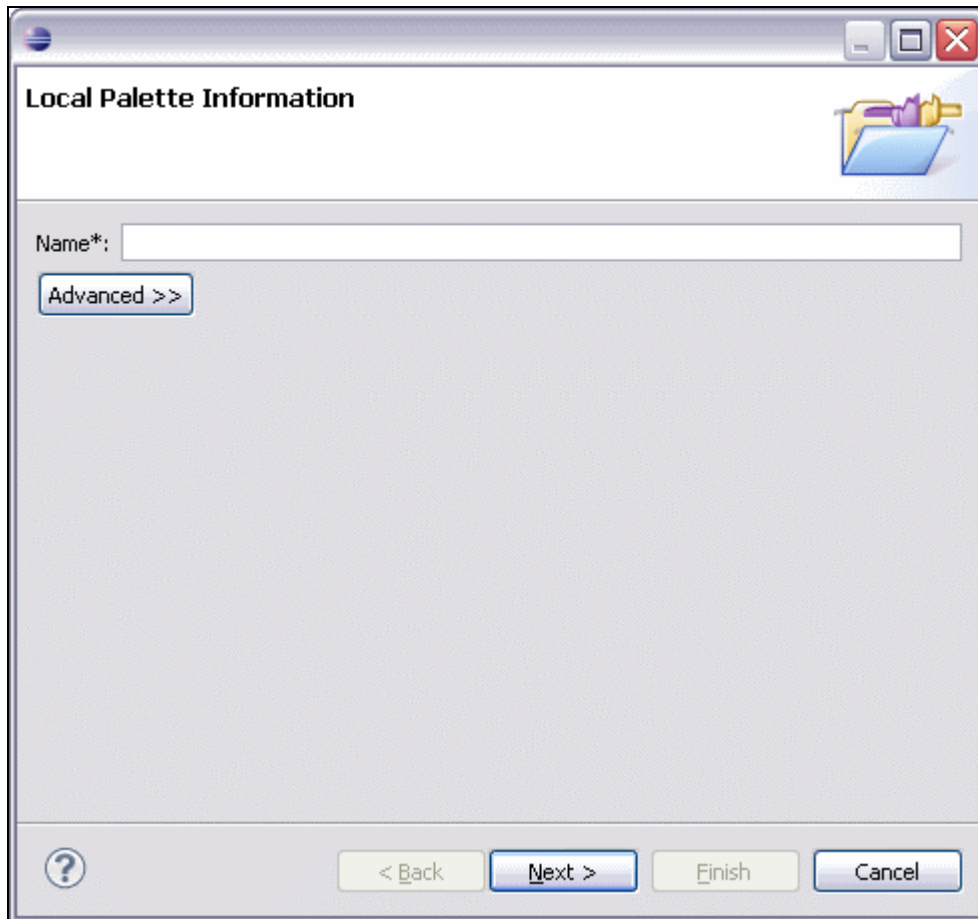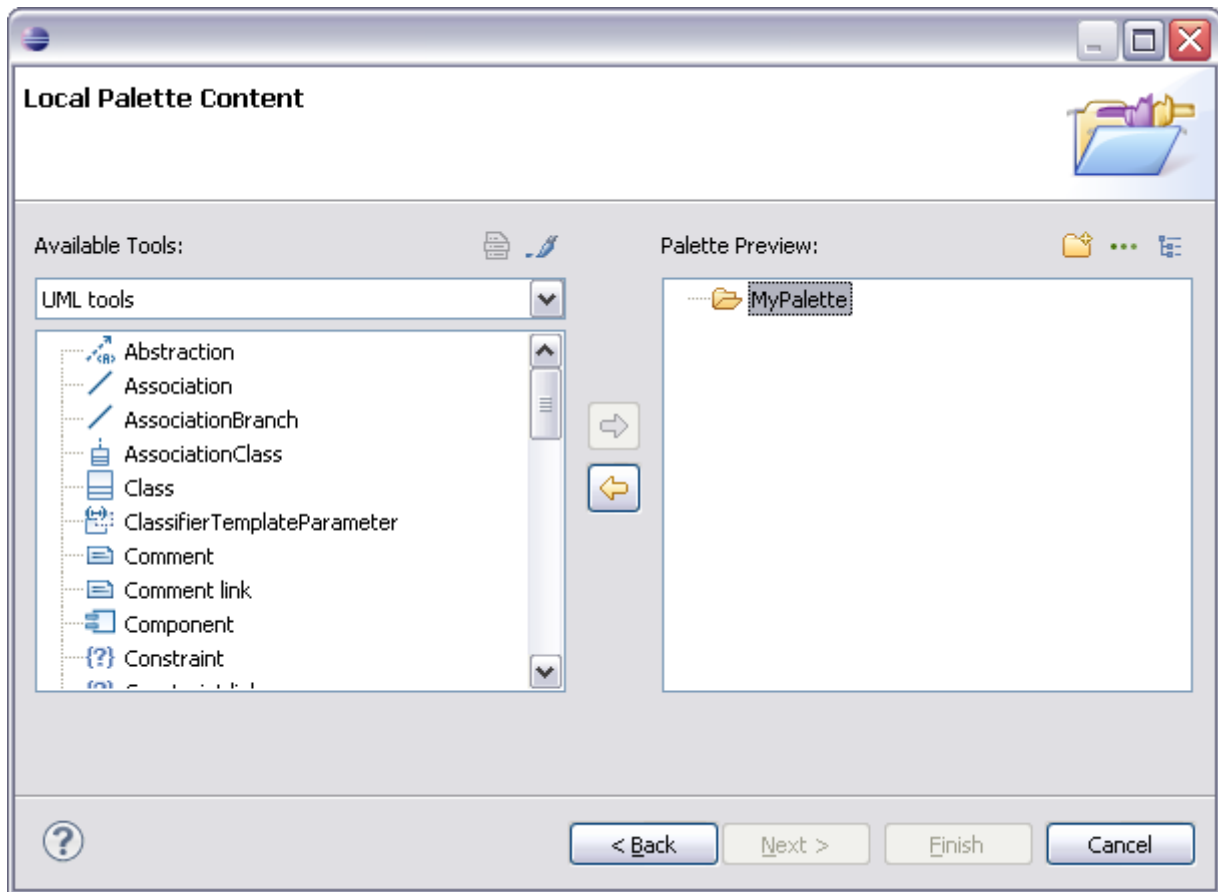
The advanced button toggles advanced information fields visiblity. Standard users should never modify these fields. The fields in this part allow fine customisation for the palette. For example, the id of this palette is defined here. A unique ID is generated, so user should not modify it.

The second page is the definition of the palette content.

On the left side of the window, all predefined tools are available. For example, each tool from the standard class diagram is available for selection in this window.

_____

The combo menu provides the list of already applied profiles on the current model - remember we use an opened editor to customize the palette. This list also displays the standard UML tools for the given diagram.
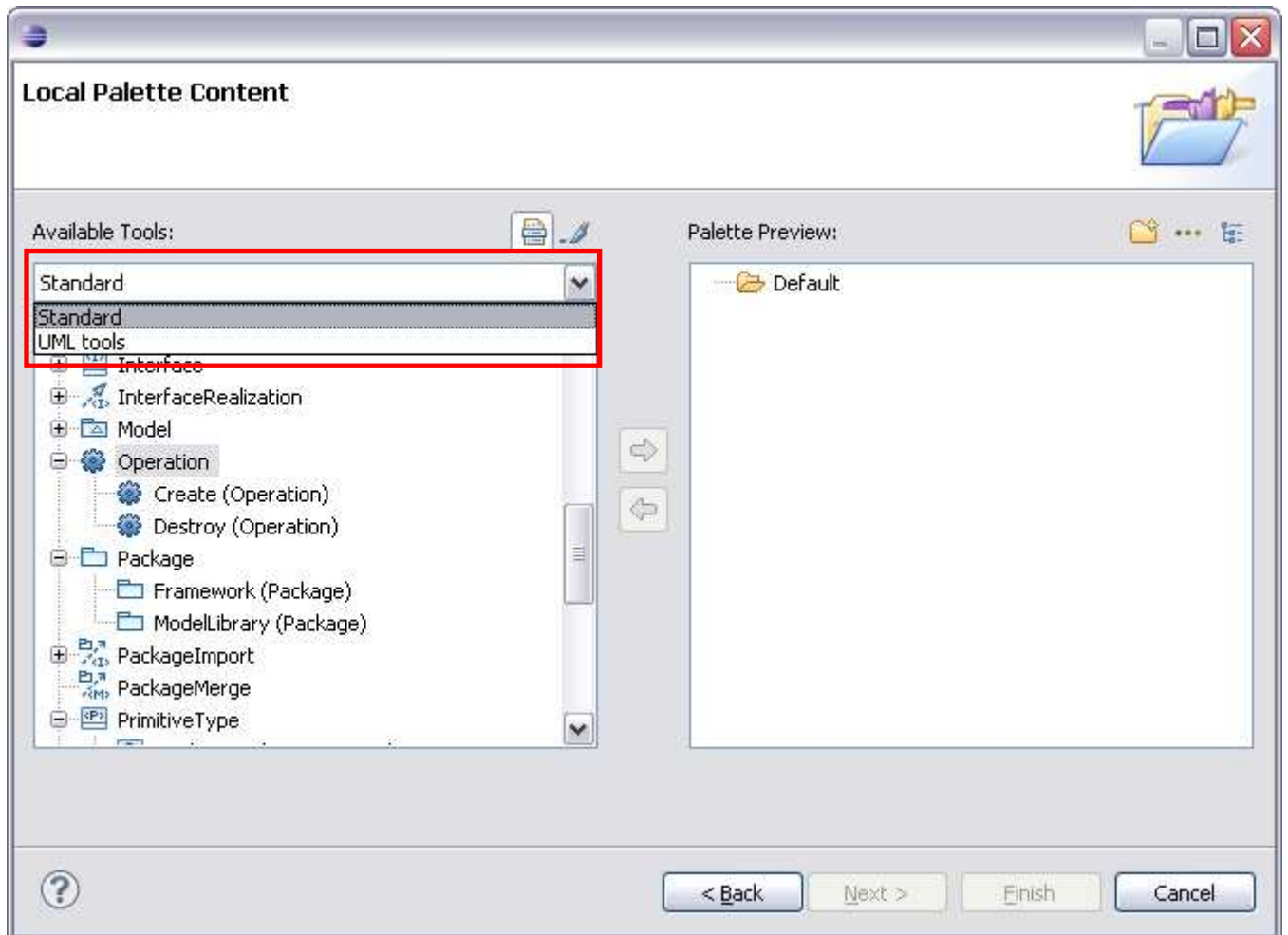
The "UML tools" displays all UML elements that have a creation tool for the diagram.

As one profile is selected in the combo, a new set of tools is presented. We present the customization of the palette using the *Standard* profile in this case. It is important to note that this generation is automatic; this means that the user does not have to manipulate some code or install plugins to have access to these tools. They are generated as soon as the user wants to customize his palette. These palette containing generated tools for a given profile disappear when the profile is removed, and are shown as soon as the profile is applied.

It is also interesting to see that a single palette can contain several tools that create elements stereotyped from different profile! There is no limitation for a single profile. If the palette is dependant of several profiles, the palette will be hidden as soon as one profile is not applied. All profiles must be applied if the user wants the palette to be shown.
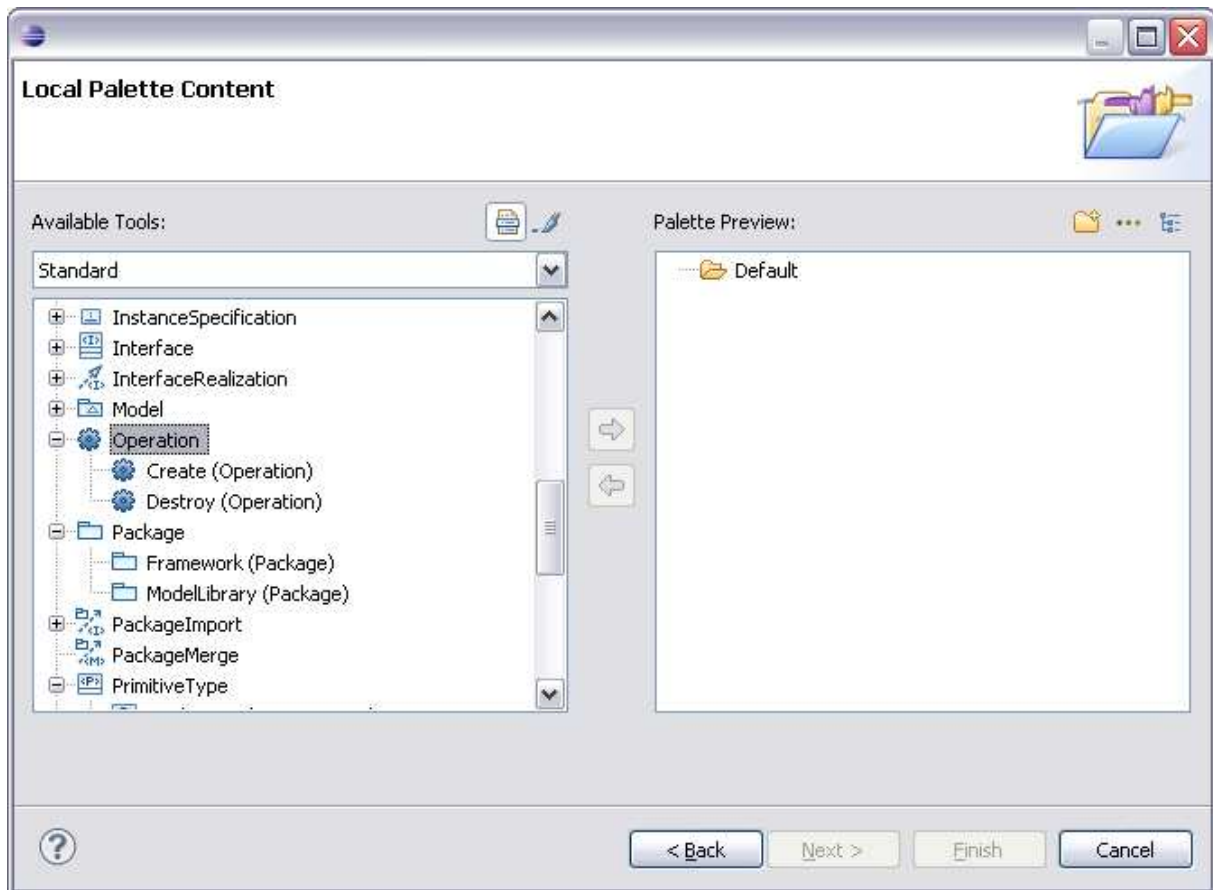
As we can see in the *available tools* list, a set of elements is displayed. They correspond to the list of metaclasses having a creation tool for the current diagram type (the same list displayed when the *UML Tools* item is selected in the combo). In the example above, the tool creating an *Operation* is selected. The sub-items of this tool are new tools. These new tools also create an *Operation*, but with a stereotype already applied on it. In the case of the *Standard* profile, we can create an *Operation* stereotyped *Create* or *Destroy*.

The available tools list can be ordered differently, using the ▣ icon on top right of the available tools viewer. This icon toggles the kind of display for the tool.

- The first kind of view is the structure described above: a tree composed by UML tools. The tools have leafs, which are tools that create the same kind of UML element, but with a particular stereotype on them.

- The second kind of view is a tree structure also. The first level is composed by the set of stereotypes present in the profile. The leafs of these stereotypes are the set of tools that can create a stereotypable element. For example, there will be the stereotype Create, with leafs for creating an operation stereotyped Create, a reception, a usage, etc.

_____
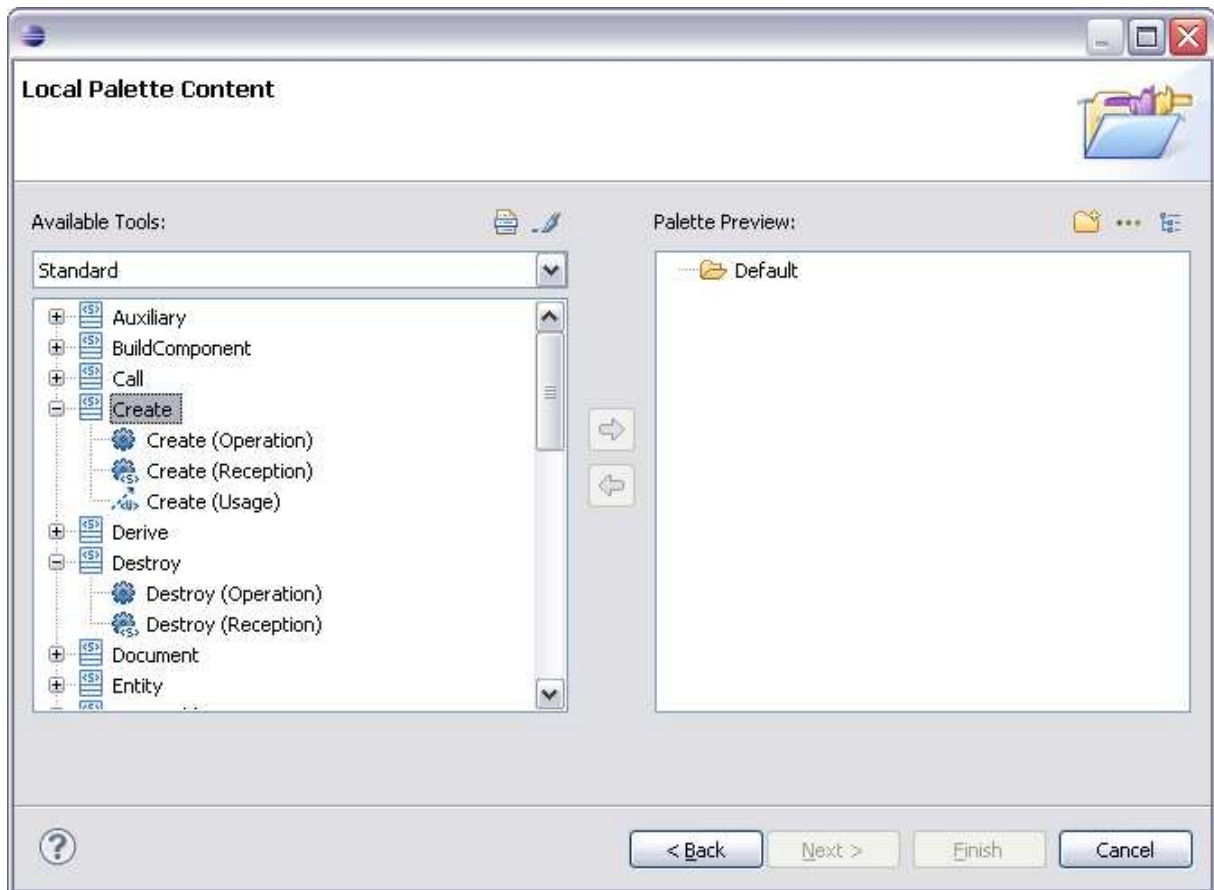
On the right side, a preview of the palette is shown. The tools on the top right corner build new elements in the palette. For example, the first icon ⬜ creates a new "Drawer", the second one ⬚ a new "separator" and the third one ⬚ a new "Stack".
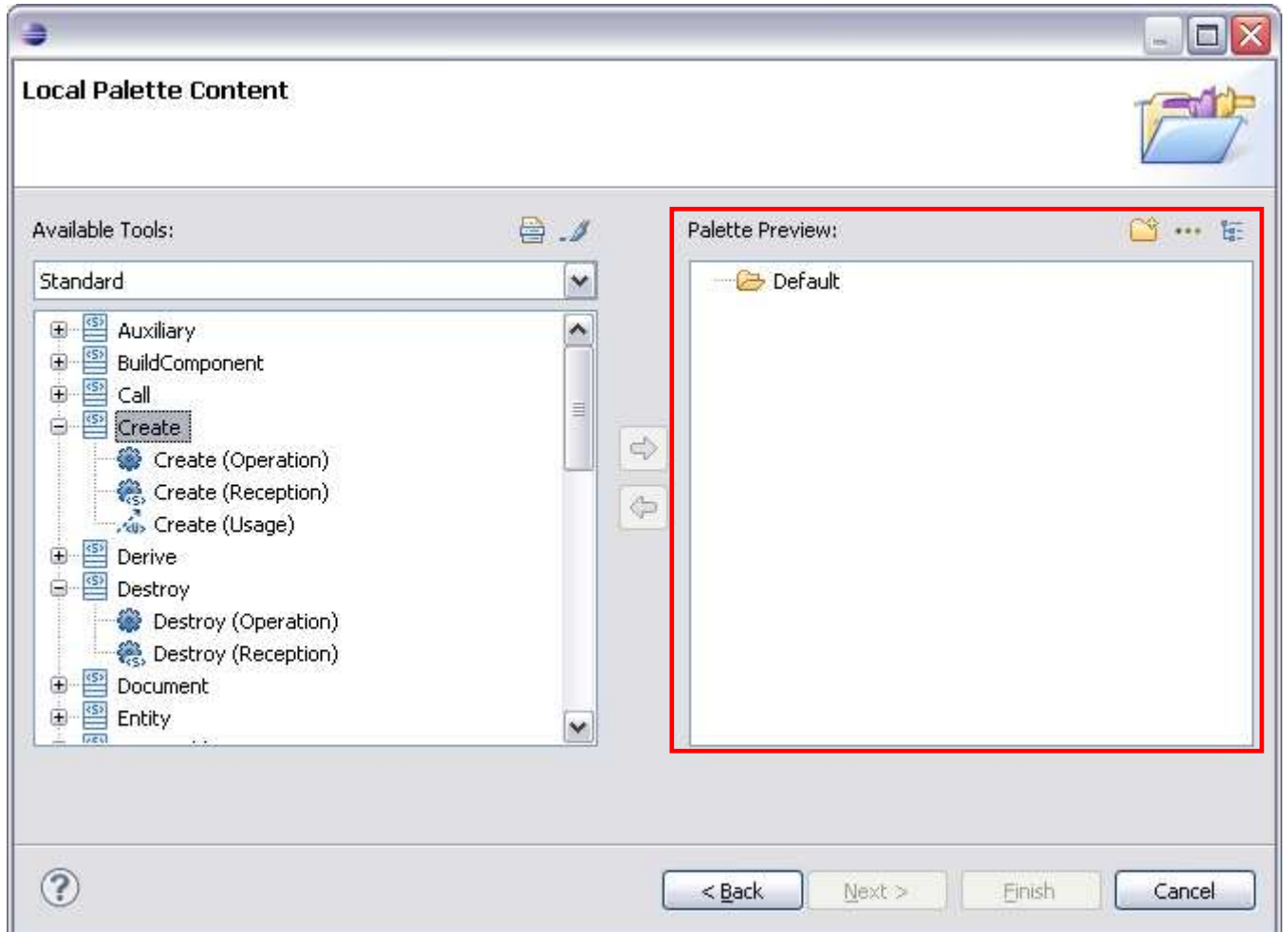
Drag and drop from the list of available tools to the palette preview is possible, as the drag and drop inside the palette preview. When dragging an element from the available tools, it creates a tool in the palette preview. When dragging inside the palette preview, it moves the element from one place to another.

There are a few rules to respect when building a palette:

- Drawers must always be direct children of the palette root. Thus a drawer can not be placed in a drawer.

- Tools and separators must be placed in a container – a stack or a drawer, they can not be placed directly on the palette root. *This means that the first step when creating a palette is to create a new Drawer.*

- A stack must contain only tools, no separator and no stacks.

- If a tool is present twice in the same drawer or stack, it will be present only once in the real editor palette.

6.3. Updating a new local palette

It is very hard, if not impossible, to create a good palette from scratch. There is always a misplaced element, a missing one, etc. Thus, the local palette definitions can be edited and updated. This is done when selecting a local palette in the customize dialog window, and pushing the "edit" button .

This opens a wizard similar to the previously presented one, the only difference is the fact that fields and palette preview are already filled with existing configuration.

6.4. Deleting a local palette definition

When a user wants to remove a local palette definition from its workspace, he can delete the definition he wants. He can select a local palette and select the "delete" button .

WARNING: This removes definitely the palette. It is impossible to recover this palette configuration!

6.5. Export a palette

Not implemented yet... the PDE wizard for a new plugin project should be extended for this functionality.

6.6. Import a palette

Not implemented yet... an XML parser should be build upon the plugin.xml file, to build a local palette from palette defined by a plugin.

A second import action should be done to exchange local palettes. The xml file is currently stored in the ".metadata" hidden folder in the workspace. This file can be copied to export a local palette. A way to import such XML files should be done in the new palette wizard.

7. Defining palette using plugins

As said in previous chapters, the goal of the gmf generation modification was to allow the reuse of predefined elements. These predefined elements, like the tool that creates a *Class*, can be used in a palette definition using the GMF palette extension point. The standard palette generated by the gmfgen itself currently uses this kind of extension.

Thus, a tool developer or a methodology provider can build and publish his own palettes. For example, several palettes could be furnished for the class diagram.

Two possibilities for the palette selection can be developed.

- Selection by the user

   This first solution is already present. User can select through a context menu in the palette which palettes he wan to display.

_____
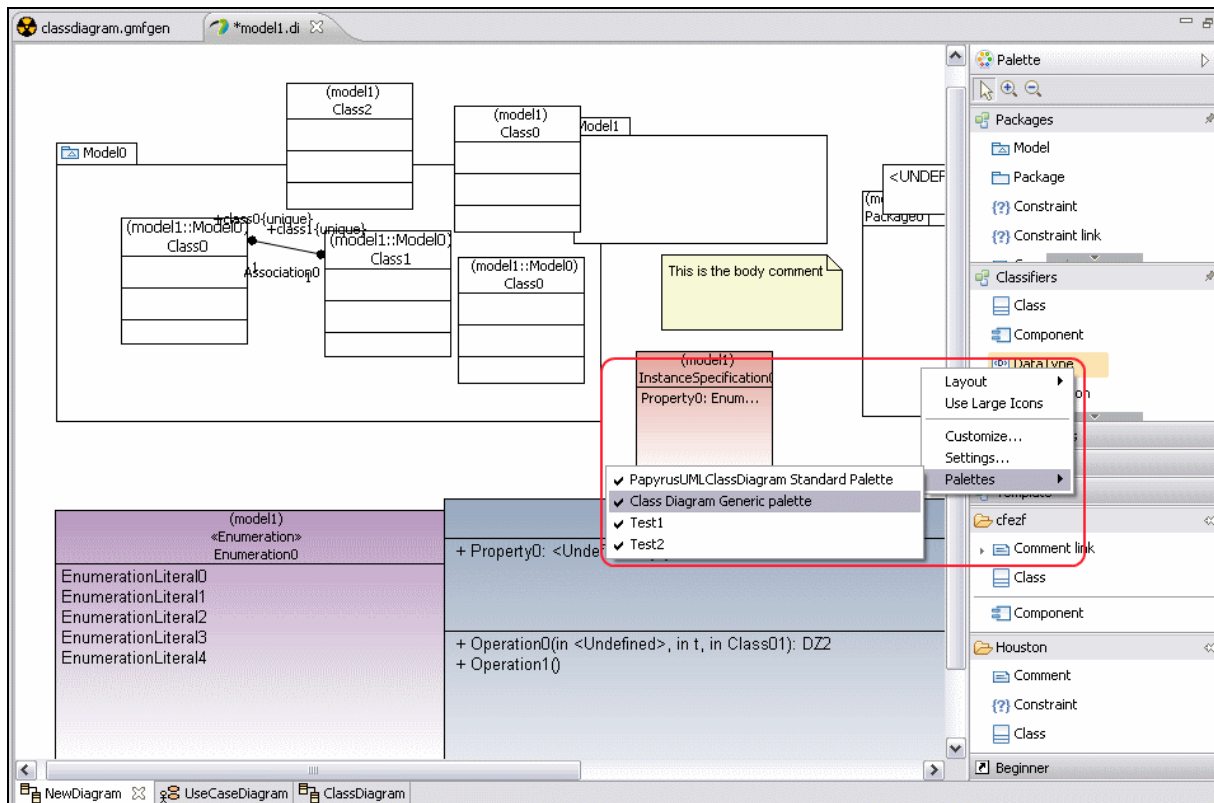
- Selection by configuration

  A tool like a model-based configuration or eclipse activities should be able to configure the tool given the current step in the development process. The tool can be configured; activities or configuration should now be implemented to have such behaviour. The user would not have the right to modify his palette in this kind of situation.