Dive into plugins: the PlantMPS plugin

Introduction

The PlantUML plugin developed by the mbeddr.com project seemed to me very interesting, I already knew PlantUML so I decided to study the plugin to learn how an external tool can be plugged into MPS.
As a bonus I've created a self contained plugin called PlantMPS that let one uses PlantUML without importing into MPS the whole mbeddr platform.

I've also ported the LanguageVisualization plugin to PlantMPS so one can use it without mbeddr.

The tutorial describes almost all the steps I've followed to write the plugin but it's not a step-by-step guide. There are other resources available about plugins so here I have not documented generic plugin features, rather I've focused more on specific technics and API used to implement the plugin by the mbeddr guys.

How the plugin works

First of all the plugin define a language with an interface called `IVisualizable`, concepts that implement that interface can be visualized in one or more PlantUML diagram types. To implement this the `IVisualizable` interface defines two methods:

- the first returns an array of strings that are the categories the concept supports, for instance Class diagram and Activity diagram.
- the UI let the user to choose one of the supported categories, so the second method receives the select category and fill the diagram, also recevied as parameter, with PlantUML statements required to display the selected category for that concept.

An example of these methods can be:

```
public string[] getCategories() {
return new string[]{"Class diagram", "Activity diagram"};
}
```

and

```
public void getVisualization(string category, VisGraph graph)
  overrides IVisualizable.getVisualization {
  if (category.equals("Class Category")) {
    graph.add("\nclass " + this.name);
  } else if (category.equals("Activity Category")) {
//...
}
}
```

To use the plugin your concepts should only implement this interface, the rest is handled by the plugin and include:

- history to handle next and previous diagram visualization.
- save the generated SVG image.
- copy the PlantUML source to the clipboard.
- zoom
- go-to-node feature when the user click on the SVG image.

PlantMPS also differs from the original mbeddr plugin in:

- mbeddr starts an HTTP server that exposes several services that can be requested by internal or even external processes. In the original plugin the go-to-node feature was implemented transforming the user click over the SVG into an HTTP request. PlantMPS directly select the requested node. In case you are interested in this topic take a look at app lication plugin and extension point concepts in the c.mbeddr.mpsutil.httpserver module.

Credits

- mbeddr.com project.
- LanguageVisualization plugin.

Getting started

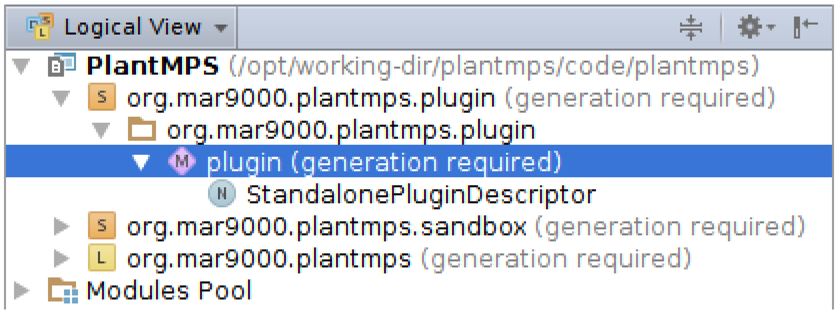
Create the project

Usually I place the MPS stuff under code, so in the project root I can place documentation, README, etc, not related with MPS. So the project one can open from MPS is in our case under `code/plantmps`. The language created with the project will hold the `IVisualisable` interface and the `VisGraph` class used to hold the diagram source.

Then we need a plugin solution as the container for our plugin:

- right click over the project node and choose New -> Plugin Solution.
- the choosed name is org.mar9000.plantmps.plugin.

The MPS IDE should now look this way:



```
PlantMPS (/opt/working-dir/plantmps/code/plantmps)
  org.mar9000.plantmps.plugin (generation required)
    org.mar9000.plantmps.plugin
      plugin (generation required)
        StandalonePluginDescriptor
  org.mar9000.plantmps.sandbox (generation required)
  org.mar9000.plantmps (generation required)
Modules Pool
```

Create the tool

Now that we have a plugin solution we can create an MPS Tool:

- right click over the plugin model and select New -> j.m.lang.plugin -> Tool.
- set the name to SVGViewer and caption to Visualization.

Add a tool icon

This is an interesting part. First the icon size used by MPS is 13x13, I copied the icon from the mbeddr project after investigating the source code of ther Idea project where the class `AllIcons` comes from. The `icons` dir has been placed at the same level of the solution, indeed `code/plantmps/solutions/org.mar9000.plantmps.plugin/icons`.

Second to specify icons you need the j.m.lang.resources language, so include it as used languge, you can do it from the Module Properties dialog:

- place the coursor over the `icon:` field and hit ctrl+space
- you should see the `IconResource` option.
- select it, a button should appear on the UI that let you select the icon file from the filesystem.
- if you look at the the inspector you should see how paths can be specified relative to module directory, that is `${module }/icons/...` .

Complete a dummy tool

The only thing to complete to open our first tool is the getComponent() method, for the moment just return a JLabel(). To do this you need to import `JDK/javax.swing@java_stub`. The fastest way to import this model is:

- hit `ctrl+R`.
- write `JLabel`.
- one of the suggested option should be the model indicated above, select it.
- now you can use `new JLabel()` as java statement.

The tool should look this way:

```
tool SVGViewer {
caption: Visualization
number: <no>
icon: <here you have the added icon>
position: right
[snip]
getComponent()->JComponent {
new JLabel("Your first Tool.");
}
}
```

Tool life cycle

Remember that a Tool is instantiated only once when you open it or in case of modifications while you are developing it, indeed the `getComponent()` method is called once. Actions when invoked alter the state of the Tool, in our case set the node to be displayed, and call `openTool()` method that has the effect to set the Tool visible on screen.

Adding actions

An action is required to open our tool and an action group is required in order to display the action menu somewhere in one of the MPS menus. First create the action:

- right click on the plugin model and select New -> j.m.lang.plugin -> Action.
- set the name and other trivial parameters.
- `action context parameters` are parameters passed to the action at runtime. They are specified from fields of classes of the MPS or Idea API.
  Here we need the key `PROJECT` contained into `CommonDataKeys`, include its model with ctrl+R.
  Another important container of the keys is `MPSCommonDataKeys`, we are going to use it in this tutorial.
- now you should be able to select `PROJECT` into the drop down menu that appears with ctlr+space.
- set the `mnemonic:` parameter to "V", it does not bind any key to our menu only display the first letter of Visualize under scored.

To complete the `execute()` method as below your should add `MPS.Platform` as dependency of the plugin solution. The tool should look now this way:

```
action VisualizeAction {
mnemonic: V
execute outside command: false
also available in: << ... >>

caption: Visualize
description: <no description>
icon:<our icon>

construction parameters
<< ... >>

action context parameters ( always visible = false )
Project project key: PROJECT required

<update block>

execute(event)->void {
tool<SVGViewer> svgViewer = this.project.tool<SVGViewer>;
svgViewer.openTool(true);
}
}
```

Last thing to do to open the tool is to add our action somewhere into the MPS menus:

- right click on the plugin model and select New -> j.m.lang.plugin -> ActionGroup.
- call it `VisualizeActionGroup`.
- you should see a small red/mandatory field, here select element list, another field where you can select actions should appear.
- select our `VisualizeAction`, should be the first option.

The `modifications` section is used to specify where the action menu should appear, as in the case of `CommonDataKeys` we need to import a model in order to specify interesting places:

- `EditorPopup` contained in `j.m.ide.editor.actions` is the menu that appears when you right click over the editor area.
- `NodeActions` contained in `j.m.ide.actions` is the menu that appears when you right click over a node in the tree area.
- search `ActionGroupDeclaration` to find more, another one is `ModelActions`.

Your action group should be:

```
group VisualizeActionGroup
is popup: false
contents
VisualizeAction
modifications
add to EditorPopup at position <default>
```

It's time to open our tool:

- rebuild the solution, ctrl+F9.
- either from a node on the left tree or from the editor you should the menu Visualize.

Keys binding with Keymap

As stated above the `mnemonic:` parameter does not realize any binding rather it's just to visually underscore one of the characters of the menu.
Keys binding is declared with instance of `KeymapChangesDeclaration` from the j.m.lang.plugin language:

- after creation add a simple item.
- then select `VisualizeAction` into the action field.
- indicate `ctrl+alt+VK_V` as binding.

Rebuild with `ctrl+F9` and you should be able to open the tool using `ctrl+alt+V`.

Use the defined actions programmatically

As we saw actions can be defined that displayed somewhere in the MPS menus. But actions and action groups can also be instantiated programmatically, actions will be displayed like buttons and action groups like toolbars.
See SVGViewer.getComponent() where the Tool toolbar is built, take a look at the `actionGroup<>` and `toolbar<>` creators. For this you need to import j.m.workbench.action@java_stub. All actions into the toolbar package are very simple they get the Tool from the project field and call one or more methods to update its state. The Save action has also the `FRAME` context parameter.

Tune actions

The action we've created is always available but we want to show it only when we are on a node `IVisualizable`. To do it add a context parameter to the VisualizeAction:

```
action context parameter (always visible = false)
Project project key: PROJECT required
node<IVisualizable> elementToVisualize key: NODE required
```

Also note the `event.getPresentation().setText()` to dynamically set the text displayed as menu. If you clone and open the PlantMPS plugin you can verify that the Visualize action appears only for `IVisualizable` nodes, try the `DummyVis1` and `DummyVis2` nodes.

Dynamic actions

As we have just seen the `DummyVis2` nodes can be visualized in several categories. This is implemented with dynamic actions. In particular when a node has only one category it's handled by the `VisualizeAction` action, see `VisualizeAction.isApplicable(event)`. When a node as more than one category this action become disabled and `VisualizeActionParametrized` become enabled. This ladder action is almost equal to the first but it has a construction parameter, is this case a string named `cat`. If we have the category we want to display the `execute()` method can display this category through `VisGraph` instead of the first available category as done by `VisualizeAction`.

The parametrized actions are created by `VisualizeActionGroupDynamic` that differs from `VisualizeActionGroup` in several places. First right after its creation you should select (right below it's invisible when disable) update instead of element list. The content of the `update(event)` method is simple, however note:

- the `add` statement used to add the parametrized action.
- the `disable(boolean)` method.
- there is also a method `setPopup(boolean)` to dynamically visualize the added actions inline with all the other menus or as a separate menu (popup).

Remember to open the Inspector when you specify the `cat` parameter because you when to enter the `toString()` method.

Here the PlantMPS plugin differs from the original mbeddr plugin that always enable both parametrized and non parametrized actions. Actually the current implementation of PlantMPS could have been done only with dynamic actions enabling the popup in case of more than one action and disabling the popup in case of just one action. But this would have given a smaller tutorial. The concept that result to be handled by the dynamic group is `DummyVisualizableMoreCategory` and its instance `DummyVis2`.

Create actions programmatically

Actions can also be created, and used programmatically, see for instance `ChangeCategoryAction` that implements the combobox that let the user switch from one category to another category of the same node. In the `AbstractChangeCategoryAction` class note how the `enabled` property is created with a specific concept (`Property` from baseLanguage).

History

The class `VisualizationHistory` is almost pure java but:

- to implement the history i necessary to save a reference to the node to visualize at that point of the history and which category to show. However the node is not assigned directly to a field of an history item, as you can see from the use of the `SNodePointer` class. This is to prevent memory leaks as pointed out in HowTo -- Adding additional Tools (aka Views).

Add PlantUML and SVG support

At this point we are going to use the `IVisualizable` interface from the o.m.plantmps language, its's not described in details because it really simple and it has nothing related with plugins. Just remember that to create the field `Project project` one needs to import j.m.project@java_stub. I've pointed out this because dependencies are often the dark side of MPS.

Note that the PlantMPS plugin differs from mbeddr plugin in the way it responds to user click over the SVG image. As stated above mbeddr executes an HTTP request toward its HTTP server and it execute the node selection on the editor. Whereas PlantMPS directly execute the code that select the node, see `MbeddrUserAgent.openLink()`.

Adding jars

After you have placed your jars into a directory, in our case the `lib` dir. under the plugin solution dir, from the Module Properties window (alt+enter) you have to:

- Common tab: click Add Model Root and choose javaclasses.
- then on the right select the jars for your dir and click the Models button.
- into the Java tab add the jars into the Libraries section.

If everything went well you should see a new item into the navigation tree of your language/solution named stubs. Remember that in case you add other jars the Add Model Root -> javaclasses is required only when you add the firsts jars.

Implenting language, generation plan and java_stub diagrams

to be continued