Indepmod Class Notation Implementation Document

Lukáš Vyhlídka
ČVUT, FEL

Prague, February 25, 2011

1 Introduction

This document describes the structure of Indepmod Class Notation plugin for Netbeans. The plugin is created in Netbeans platform so if you are not familiar with it, please take a look at [**?**].

2 List of Modules

There are three modules in the Indepmod Class Notation plugin:

- API - provides an public API (Interfaces) that can be found (e.g. by other Netbeans plugins) in the Lookup of the workspace (part of the Editor module).
- Tool Chooser - is used for setting of demanded Tool of actual selected workspace. The module has a panel with single tools - e.g. Class or Relation between classes. This module will be called only ToolChooser for simplicity.
- Editor - the main module which is used as a workspace. Workspace has a JGraph inside which does all the work.
- JGraph - this module is used as a library wrap for JGraph (it allows other modules to use JGraph framework).
- jGoodiesBinding - this module is used as a library wrap for jGoodiesBinding (this will be probably deleted because I think it is not used anymore).

In the Figure **??** you can see the print screen of running application. The whole window system (window, menus, etc.) is provided by Netbeans platform. On the left side you can see the project tree, which is the standard component of Netbeans platform (frequently used e.g. in Netbeans IDE). Next to the project tree there is a user interface of Editor module. As you can see on your own, the Editor module provides a workspace where user can create the class model. On the right side there is a UI of Tool Chooser module.

Figure **??** illustrates the relation between Editor and ToolChooser modules. Editor provides instances of ToolChooserModel and IClassModelModel. These interfaces can be used by other modules. One (and maybe the once) of these modules is ToolChooser which uses ToolChooserModel to set the desired tool of active workspace (Editor module). These interfaces will be described in greater detail in following section.

3 API Module

This module consists of Interfaces which implementations can be found in the lookup of the workspace (workspace is the part of the editor module). There are two packages in this module:

- cz.cvut.indepmod.classmodel.api.model
- cz.cvut.indepmod.classmodel.api.toolchooser

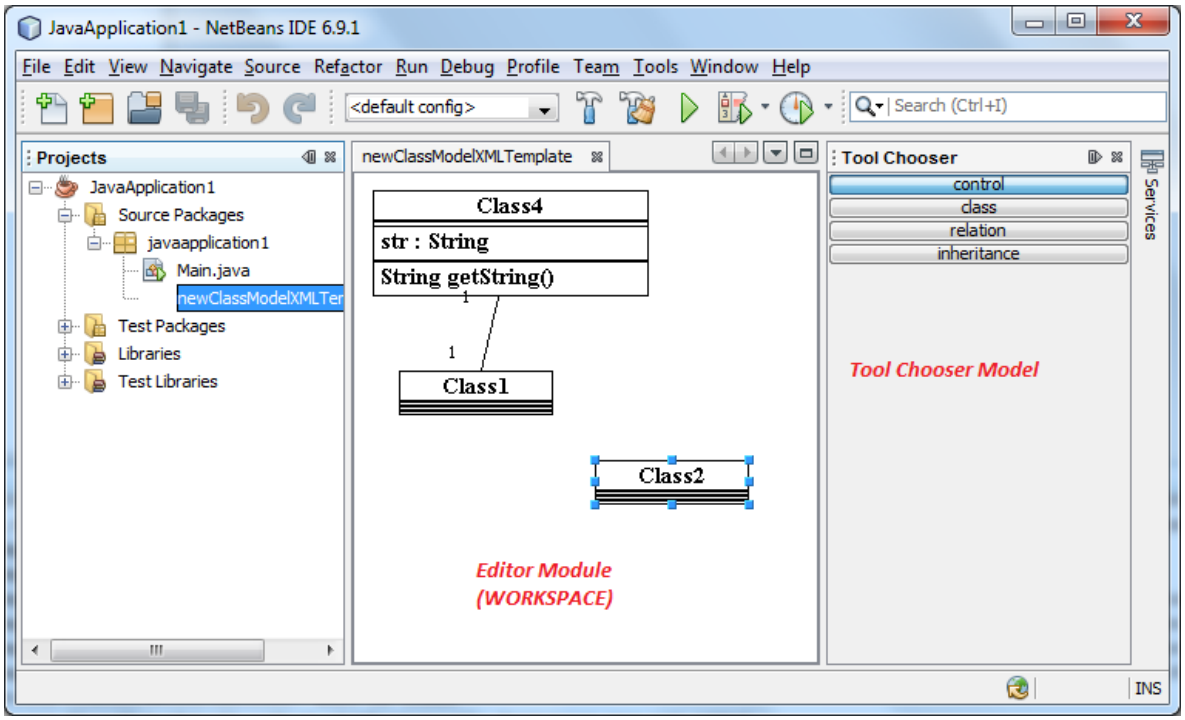

Figure 1: Screenshot of the program

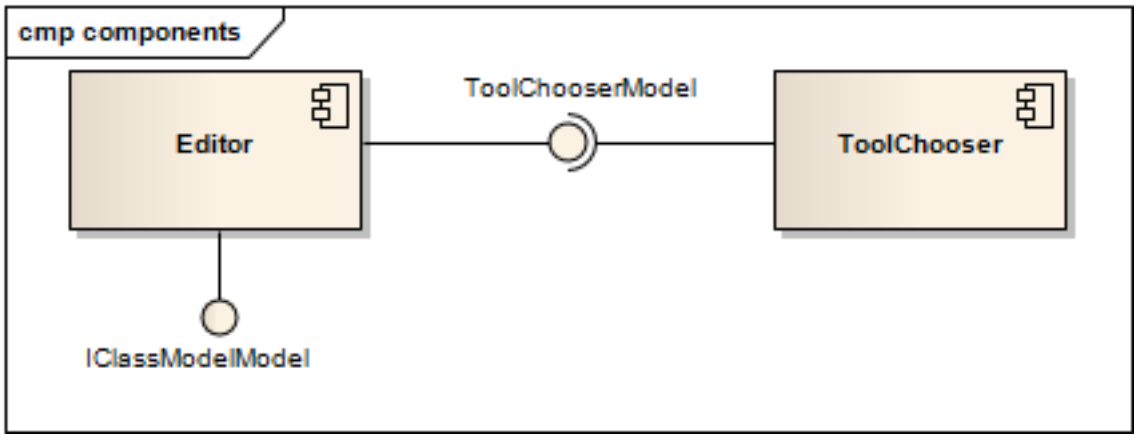

Figure 2: Component diagram of Editor and ToolChooser Modules

3.1 ClassModel API (cz.cvut.indepmod.classmodel.api.model)

In this package there are interfaces which can be used to read the Class Model Notation Model. There is an implementation of the IClassModelModel interface in the workspace lookup. This object provides the information about the type of model (business model / class model) and list of Classes from a model. Classes provides the list of annotations, attributes, methods and relations to other classes. And so on with the attributes, etc.

Relation between (two) classes is represented by instance of IRelation interface. IClass

returns the list of IRelations that belongs to it. IRelation holds information about its type (simple relation, composition, agregation, generalization, implementation) and about what class is at the beginning and at the end of the relation, including cardinalities.

Cardinalities are represented by instance of ICardinality interface. The ICardinality interface has two methods (getFrom() and getTo()) which return from and to value (both are of integer type). Sign of infinity (e.g. 1..* or *) is treated as -1. In the Table **??** you can see examples of return values of some cardinalities.

Cardinality	getFrom() result	getTo() result
0 (or 0..0)	0	0
1 (or 1..1)	1	1
4 (or 4..4)	4	4
* (or 0..*)	0	-1
1..*	1	-1
0..1	0	1
2..5	2	5

Table 1: Examples of cardinalities

TODO: Image with structure (tree)

3.2 ToolChooser API (cz.cvut.indepmod.classmodel.api.toolchooser)

This package is used by both ToolChooser and Editor modules. Workspace (part of Editor Module) has an instance of ToolChooserModel in it's lookup. ToolChooser uses this instance (from the lookup of active workspace) to set the desired tool (e.g. new class). Structure is really simple and you can see it in the Figure **??**.

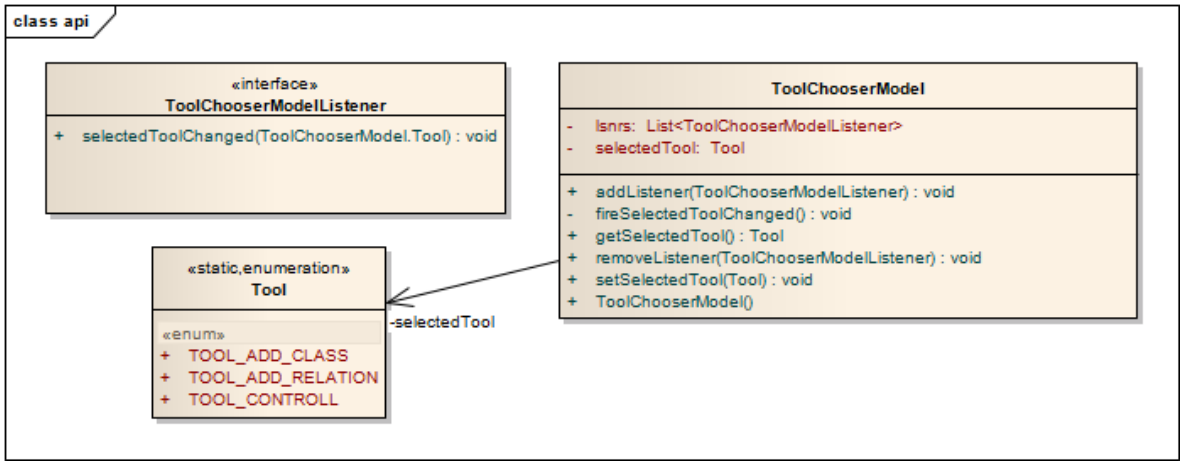

Figure 3: Class Diagram of ToolChooser API package

4 ToolChooser Module

This module is used for setting of demanded tool of active workspace. The structure can be seen in the Figure **??**.

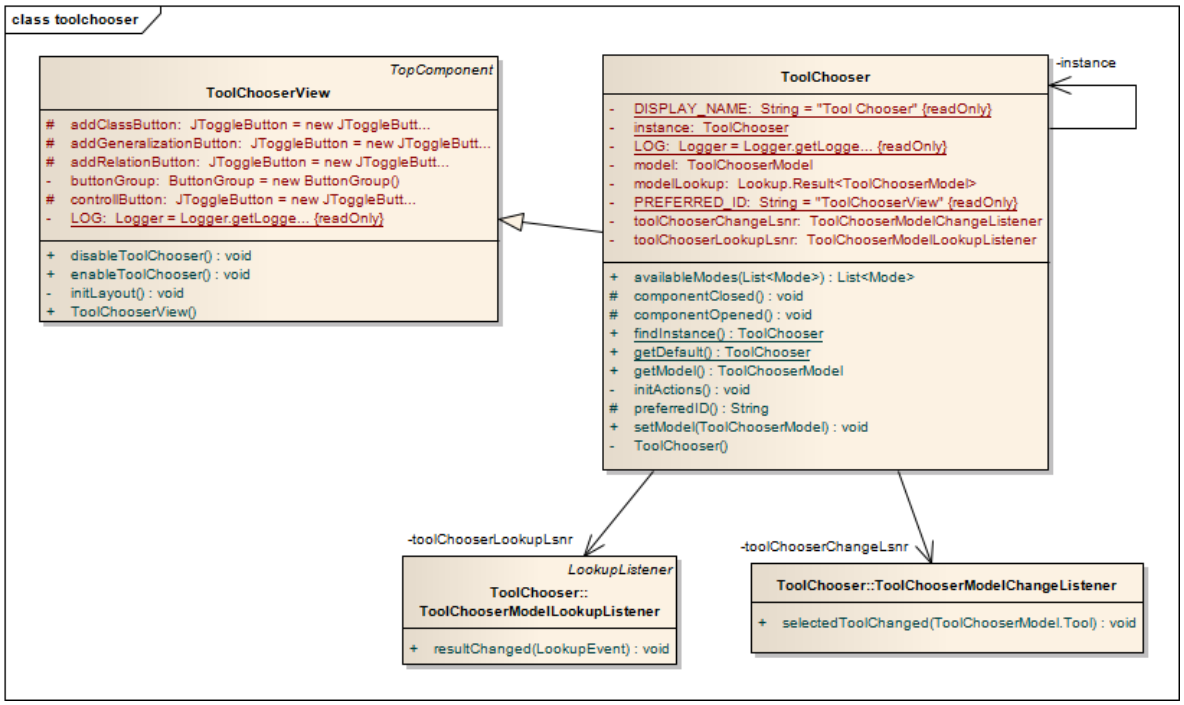

Figure 4: Class Diagram of ToolChooser Module

ToolChooser is a singleton which is inherited from the TopComponent class. TopComponent is provided by Netbeans platform. Its ancestors can be used as a user interface (component that is visible to the user) in the application. TopComponent is derived from standard JComponent class (javax.swing) and adds many methods which can be used for controlling a window (more info in [**?**] in chapter called Window System).

ToolChooser searches for instance of the ToolChooserModel in the lookup of active workspace (Editor module). When the instance is found, ToolChooser can set its tool (class, relation, etc.). This means that it can be used to set the desired tool of that workspace.

In the layer.xml file there is set the action for opening the ToolChooser (Windows → Tool Chooser).

5 Editor Module

This is the fundamental module whose purpose is to create a workspace where user can create a class model. For class modeling is used framework JGraph (http://www.jgraph.com/). More info about JGraph framework can be found in [**?**].

5.1 Module Structure

Main class (part of the cz.cvut.indepmod.classmodel.workspace package), which comprises the workspace, is the ClassModelWorkspace. Thus, if you want to study the code, you should start right here. ClassModelWorkspace is extended from the TopComponent. This class is responsible for whole initialisation. It creates:

- ClassModelGraph - this class is extended from JGraph and represents the class model graph to the user. Instance of this class is situated inside the ClassModelWorkspace component (JGraph is also derived from JComponent)
- ClassModelModel - Implementation of IClassModelModel which returns the list of classes that are in the class model and the type of the diagram (class or business model).
- ClassModelMarqueeHandler - this class is responsible for handling of user inputs that are made inside the ClassModelGraph.

ClassModelWorkspace is simply JComponent (TopComponent) which has the JGraph inside. JGraph presents the class model to the user. There are cells (representing classes) which are related together by edges (representing relations).

In next section I will try to describe the implementation of the Editor Module. I Will start from the initialization and after that I will try to explain every part that is created during the initialization.

5.2 Editor Module Initialisation

As I have already said before, the ClassModelWorkspace class do the initialization of the whole module. There are two cases of initialization. The first case, when user creates new class diagram, and the second, when user opens an existing file. Both cases are very similar. They differ only in the way of ClassModelDiagramDataModel initialization. For this purpose there are two constructor variants.

The first, non parametric, is used when the new class diagram (with no associated file) is created. This constructor creates new instance of ClassModelDiagramDataModel by ClassModelDiagramModelFactory.

The second constructor is used when the user opens a file with a class model. It accepts an instance of ClassModelXMLDataObject (one parameter). This object represents the file in which is the class model saved (more in the chapter **??**). The constructor gets the input stream of that file and asks the ClassModelXMLCoder to decode its content. ClassModelXMLCoder decodes it and returns ClassModelDiagramDataModel instance filled according to that file's content. ClassModelXMLCoder is part of the persistence layer and will be discussed in the chapter **??**.

ClassModelDiagramDataModel class represents the data which has to be persisted when user want to save his class diagram. At present, instance of this class holds the GraphLayoutCache instance (class of JGraph framework which holds the information about cells in the graph), list of static data types (default list of data types for certain language which is chosen during new class diagram file creation) and the type of the diagram (class or business model, it is also chosen during new file creation).

After the ClassModelDiagramDataModel is gained (created or loaded) the initialization is the same. What the ClassModelWorkspace creates have been already written up.

For better understanding you can see the sequence diagram in the Figure **??** which shows the initialisation of new ClassModelWorkspace. For simplicity there is only what ClassModelWorkspace creates. The processes inside these classes are not shown (Sequence diagram would have been really big) but it will be discussed later.

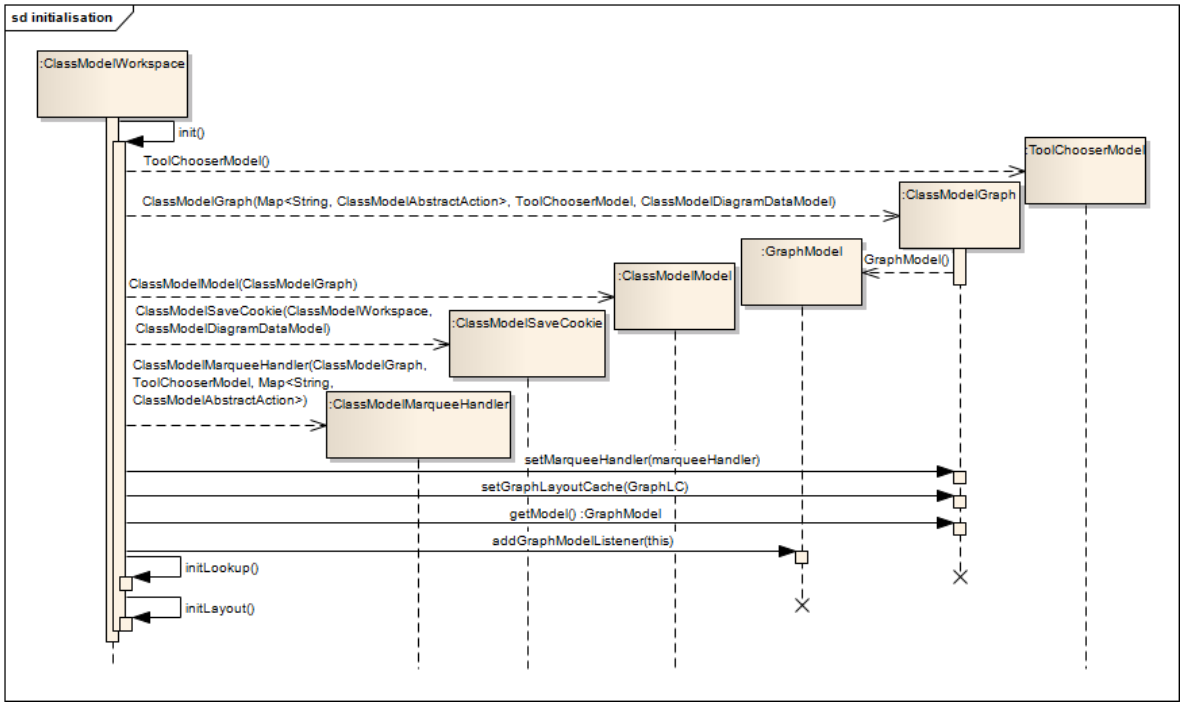

Figure 5: Simplified sequence diagram of ClassModelWorkspace initialization

5.3 Netbeans File Association

Netbeans platform allows its plug-ins to recognize new types of files. So when you want to associate new file type with your plugin, you can do it easily. If you are not familiar with this, you can find it in the [**?**] in the chapter called Data System.

The file type association settings can be found in layer.xml file. This file is used for settings of Netbeans module. At present there is settings of file type association and settings of wizard that can be used to create new class diagram file. In this wizard user sets the name of the file, type of the diagram and the language (Java, C++, etc.). Files that are associated with this plugin are xml files with .cls suffix. Content of the file will be discussed later.

5.4 ClassModelGraph implementation

ClassModelGraph is extended from JGraph and adds some functionality for Class manipulation. Added methods are:

- insertCell(Point p) - creates new cell on desired position according to the selected tool (in ToolChooserModel). Usage of this function is shown in the section **??** in Figure **??**.

- getAllClasses() - returns the list of classes that are in the diagram. Usage of this function will be shown in the section **??**
- getAllTypes() - returns the list of all data types in the diagram. In addition to the classes (the class is a data type) there are also static data types (like String or int in Java).

For event handling in the graph (ClassModelGraph) there is the ClassModelMarqueeHandler. Instance of this class is created in the ClassModelWorkspace and is added into the ClassModelGraph instance. Purpose of this class is to handle all user inputs in the ClassModelGraph (in the canvas of the graph) and control the popup menu (JPopupMenu and its content). When user does an action (e.g. click with mouse), the ClassModelMarqueeHandler will find out if it is a control click (e.g. cell selection), new cell addition, edge (line between cells) addition and so on. This class is also responsible for rendering of the temporary line when user creates new edge.

5.5 ClassModel API implementation

Implementation classes of ClassModel API (its interfaces are in the API Module) are situated in the cz.cvut.indepmod.classmodel.cell.model.classmodel package. Main problem I had to deal with was to design where to store the data of the ClassModel.

Basically, ClassModel class is used as a User Object for JGraph cells. User Object (Class Model instance) does not have normally the pointer to its cell (only cell has the pointer to it's User Object). ClassModel holds information like name of the class and list of its attributes, methods and annotations. But where to store the information about relations with other cells (classes)? This information is stored in the JGraph (in its model). So the first idea was to copy this information into the ClassModel instance when an relation is created. But this is not very nice because of data duplication. The second purpose was to add an pointer to cell into the ClassModel instance. But how? The answer is quite simple. I created the ClassModelClassCell which extends DefaultGraphCell of JGraph. The extension of this class is that it adds an pointer to itself into the User Object if this User Object is instance of ClassModel.

So problem is solved. Some information like the name are stored inside the User object and some like the relations are gained from the JGraph cell. In the Figure **??** you can see how is created a class when user selects new class tool in the ToolChooser and clicks somewhere in the ClassModelGraph (JGraph) workspace.

5.6 JGraph class cell rendering

In this section I will explain how I render the class (the rectangle with the class name and lists of annotations, attributes and methods) into the JGraph workspace. But first a little theory remind. JGraph uses MVC pattern for cell rendering. Cells (the model of cell) in JGraph are represented by DefaultGraphCell (or its subclasses). DefaultGraphCell implementation stores the user object and an attribute map inside which is used when you want to set the appearance of that cell. All graph cell has an associated view (VertexView implementation). This VertexView implementation associates the renderer, editor and cell handle together for an cell (DefaultGraphCell implementation). The cell and view is associated together by

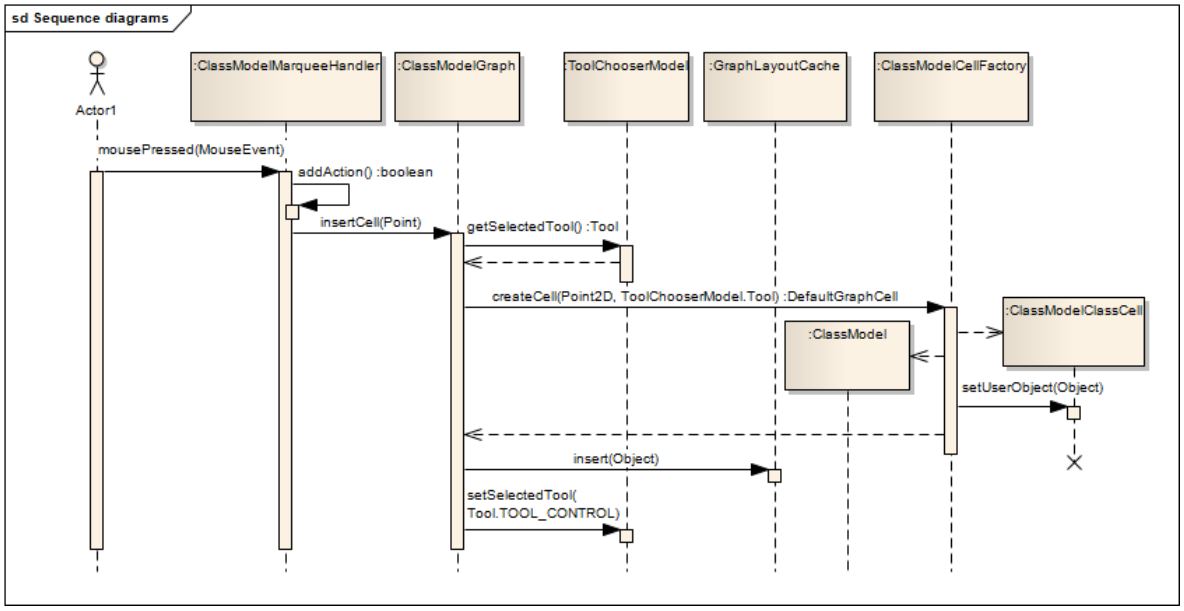

Figure 6: Sequence Diagram of class creation

CellViewFactory which returns a cell view for a particular cell. More on this theme can be found in [**?**].

JGraph basically supports rendering of basic shapes like rectangles, ovals and so on. When you want to render your own shape, you have to create your own cell view (VertexView), renderer (CellViewRenderer) and CellViewFactory implementation.

So this is exactly what I did. I created ClassModelCellViewFactory which is derived from JGraph's DefaultCellViewFactory and overrides it's method createVertexView(Object o). If the object in the parameter is instance of ClassModelClassCell class, the method returns new instance of ClassModelVertexView (derived from JGraph's VertexView). Otherwise the method returns the default view (calls its parent's method). ClassModelVertexView returns statically instantiated ClassModelVertexRenderer which is derived from VertexRenderer and overrides its method getRendererComponent(...). This method returns new instance of ClassComponent if the User Object of the cellView (argument) is of ClassModel type. ClassComponent is extended from JComponent and renders the class according to the ClassModel user object.

These classes can be found in the package cz.cvut.indepmod.classmodel.workspace.cell. Instance of ClassModelCellViewFactory is inserted into the GraphLayoutCache during the workspace initialisation. You can see the class diagram in the Figure **??**.

5.7 Persistence

Persistence layer implementation is situated in the cz.cvut.indepmod.classmodel.persistence.xml package. There is a ClassModelXMLCoder class that is created as a singleton. Instance of this class is responsible for encoding and decoding the ClassModelDiagramDataModel instance into or from a stream (InputStream or OutputStream). Its method for encoding, encode(ClassModelDiagramDataModel model, OutputStream stream), is called by SaveCookie

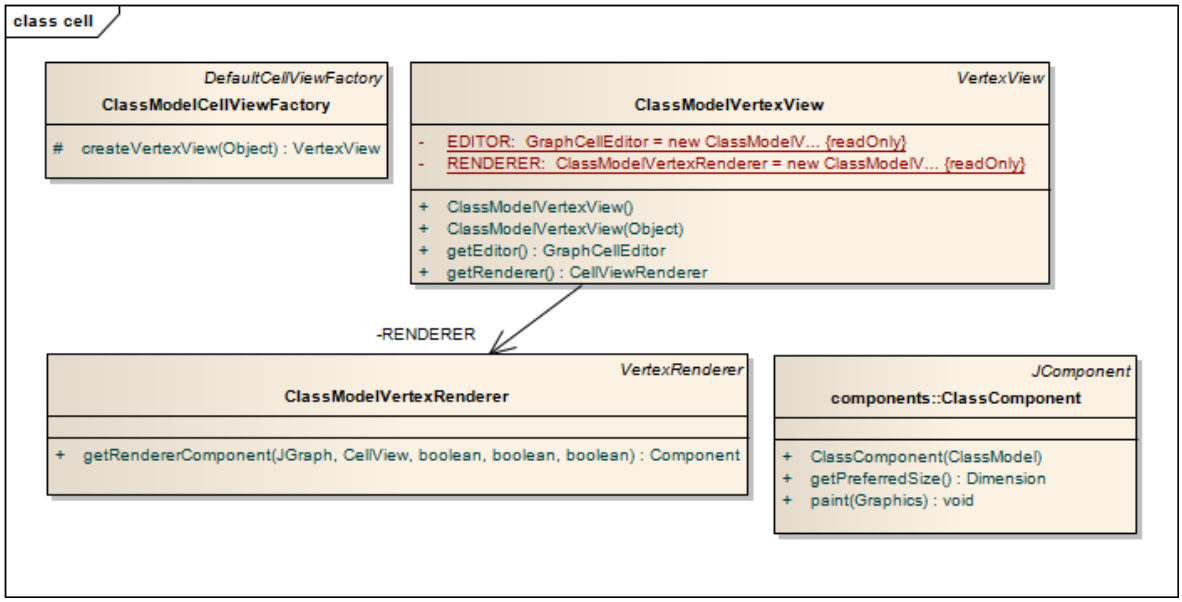

Figure 7: Structure of classes which render the class cell

(discussed in the section **??**).

ClassModelXMLCoder uses java.beans.XMLEncoder and java.beans.XMLDecoder. This creates the xml file that represents the steps to create the exactly same object that was encoded. Because XMLEncoder can encode in default only some types of objects, there are some persistence delegates that allow to encode particular object I use. These persistence delegates can be found in the cz.cvut.indepmod.classmodel.persistence.xml.delegate package.

If you want to know more about how XMLEncoder or XMLDecoder works, please take a look at [**?**].

5.8 Lookup

Lookup is (in Netbeans platform) a technique to join more plugins together with loose coupling. On the module level, lookups enable to register a service by its provider and find the service by its consumer. On the level of single components (TopComponent), lookups enable to exchange data between this component (provider) and consumers. More reading about this can be found in [**?**] in the chapter called Lookup.

Editor Module uses the Lookup on the component level. ClassModelWorkspace creates its Lookup during the initialization. It adds there an instance of ToolChooserModel and an instance of ClassModelModel.

ToolChooserModel instance is used by ToolChooser Module to set the desired tool. Thanks to this, there can be one ToolChooser component which manages all ClassModel-Workspaces. Of cource, ToolChooserModel instance in the lookup can be used by other plugins also.

ClassModelModel instance implements the IClassModelModel interface and can be also used by other plugins. It provides the Class Model API to the outside world. Thanks to this there can be for example another plugin that will generate the code from the class model

of this module.

When user does any change into the model (add new class, new relation between classes, change the attribute of the class or something similar), ClassModelWorkspace inserts an SaveCookie instance into its Lookup. This tells the Netbeans platform that this component is changed and can be saved (save action in File menu is enabled). When user calls this action, the SaveCookie tells the ClassModelXMLCoder to save the GraphLayoutCache into the associated file. When it is done, this SaveCookie implementation is removed from the Lookup, so the Netbeans platform knows that the component does not have to be saved.

5.9 Package Structure

You can see the package structure in this directory tree:

- cz.cvut.indepmod.classmodel
  - actions ..... Actions for buttons etc.
  - file ..... SaveCookie and file type association support
    - wizard ..... Wizard for creation of new class model file
  - frames
    - dialogs
  - modelfactory
    - diagrammodel
  - persistence ..... layer for data saving
    - xml ..... implementation of persistence for saving into a XML file
      - delegate ..... XML Delegates for mapping into xml
  - resources
  - util
  - workspace ..... Main package containing the TopComponent with JGraph
    - cell ..... Cells, renderers, VertexViews, etc. for JGraph
      - components ..... Component for JGraph
      - model
        - classmodel ... Implementation of ClassModel API from API Module

Reference

[1] *JGraph 5 manual, found at* ***http://www.jgraph.com/downloads/jgraph/jgraphmanual.pdf***

[2] PETRI, Jrgen. *NetBeans Platform 6.9 Developer's Guide. Birmingham : Packt Publishing Ltd., 2010. 273 s. ISBN 978-1-849511-76-6*

[3] MILNE, Philip. Sun Developer Network [online]. c2010 [cit. 2011-01-29]. Using XMLEncoder. Available from WWW: ¡http://java.sun.com/products/jfc/tsc/articles/persistence4/¿.

[4] http://platform.netbeans.org/tutorials/60/nbm-filetemplates.html

[5] http://blogs.sun.com/geertjan/entry/creating_a_better_java_class