Have Multiple Views with one Single Diagram! A Layer Based Approach of UML Diagrams

Cedric Dumoulin, Sébastien Gerard

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Have Multiple Views with one Single Diagram!

A Layer Based Approach of UML Diagrams

Cédric Dumoulin, Sébastien Gérard

N° 7432

Octobre 2010

Thème 035

ISSN 0249-6399

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche LILLE - NORD-EUROPE

Have Multiple Views with one Single Diagram! A Layer Based Approach of UML Diagrams

Cédric Dumoulin[1], Sébastien Gérard [2]

Thème : Embedded and Real Time Systems
Projet DaRT

Rapport de recherche n° 7432 – Octobre 2010 - 9 pages

Abstract: A diagram is the artifact used to show a graphical view of one or more elements of a model. Sometime designer whish to have slightly different views of the same model element: to show variation points, to explore alternatives, to highlight some details, to add stereotypes or comments … In actual tools, each view is shown in its own diagram, realized by copying a based diagram and adding the modifications. Each time you modify the base diagram, you have to modify all copies. We propose to introduce in diagram the notion of layers, similar to layers found in graphical tools (like Gimp). A layer is a kind of slide showing some graphics. Layers are stacked one over the others. The resulting diagram is the sum of its visible layers. This proposal enable multiple views in one single diagram. The proposed solution allows doing much more things discussed in this article.

Keywords: UML, Tools, Views, Layers, Diagram, Animation

[1] LIFL and INRIA-Lille Nord Europe, University of Lille, France. cedric.dumoulin@inria.fr
[2] CEA LIST, Laboratory of Model Driven Engineering for Embedded Systems (LISE), Boîte courrier 65, Gif sur Yvette Cedex, F-91191 France. Sebastien.Gerard@cea.fr

Centre de recherche INRIA Lille - Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley Bât.A, Park Plaza -59650 Villeneuve d'Ascq
Téléphone : +33 3 59 57 78 00— Télécopie : +33 3 59 57 78 50

Have Multiple Views with one Single Diagram! A Layer Based Approach of UML Diagrams

Résumé: ….

Mots clés: ….

1 Introduction

Models are made of model elements which can in turn contain other model elements. UML models, for example, are made of elements like package classes, properties, behavior … A package can in turn contain other packages and classes, also a class can contain properties and behaviors.
All these model elements are visible to a human user through their representation in a concrete syntax, more often a graphical one (like UML). This graphical concrete syntax is represented in what we call a diagram which can be edited in a diagram editor. Thus diagrams and diagram editors are the artifacts helping us (us human) to see and to modify what is in a model.
A diagram shows some aspects of one or more elements of the model and hides other elements irrelevant for the intent of this diagram. In the example (Figure 1) two diagrams called *Public View* and *Detailed View* show two views of a same class "`Regulator`": the first one focuses on the *public properties* and the second shows *all the properties*.

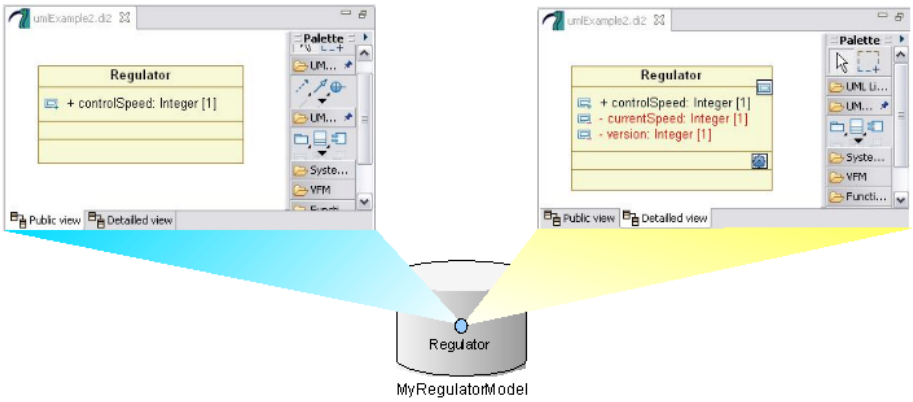

Figure 1 Public View and Detailed View Diagrams

In this multi-views context we would like to be able to represent different variations around a base diagram: a view adding stereotypes and tagged values; a view showing comments and notes; a view zooming on selected details; a view highlighting some concepts or some aspects with the help of different colors; a view revealing the hierarchy of the classes (ancestors and children); …
To obtain these different views, one should create a new diagram for each of them, copy the base diagram and then add the new relevant elements. The number of possible views, and thus of diagrams can increase drastically. These can become even worth if we want to combine views: for example to obtain a detailed view with stereotypes …
Another drawback of this approach based on classic diagrams is a problem of maintenance of the views: Imagine that we want to add another class to the base diagram, or we want to modify an association; all the views should be updated to reflect the changes.
If we look in the domain of Computer Graphic Editors [1,2], there is a concept of layers allowing to draw a graphic by stacking layers one over the other, each layer drawing a part of the graphic. A layer can be edited without affecting the others. Layers can also be used to "*suppress unwanted information when viewing or printing a document*", which is similar to what we want to do in the aforementioned use cases.

Hence our proposal to adapt the concept of layers to diagrams. In our proposal a layer is a kind of slide containing some figures (classes, associations …). Layers are stacked one over the others. The resulting diagram is the sum of its layers: figures of all stacked layers are visible in the diagram, as if we look the layers in transparency. Furthermore, it is possible to select layers that will participate to the resulting diagram.
The paper is organized as follows: after this introduction, section 2 gives an overview of the basic principles of a classic diagram. Section 3 details our proposal for diagrams build with layers. Section 4 presents how our multi-views problem exposed in the introduction can easily be solved with our proposal. Finally section 5 draws some general conclusions and future works.

2 Classic Diagram Overview

A diagram is a view on some elements of a domain model. It is made of figures representing the elements. For the user staring at the diagram, the figures represent the concrete syntax of the domain model. A diagram is just a view on the model: several diagrams can show the same domain element in different situations. Modification of the domain element, like its name, in one diagram will be reflected in other diagrams, while modification of the figure, like its position, size or color, will only impact the diagram.
In fact, a diagram shows two kinds of data: graphical data and domain data. Graphical data are the figure *position*, *size*, and *appearance* (color, font …). The diagram owns the graphical data from which the domain data are accessible (Figure 2).

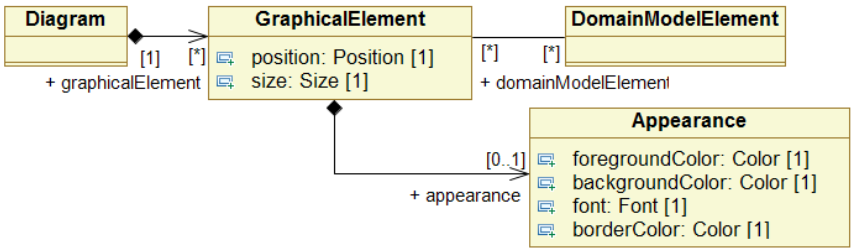

Figure 2 Classical Diagram

For some domain elements, the spatial position in the diagram is meaningful: there is a containment relation. For example, a class represented in a package means that the package owns the class. Also, a method or a property represented in a class means

3 Layers based Diagram

In our proposal a diagram is now made of layers stacked one over the others. A layer is a kind of slide showing some figures. The resulting diagram is the sum of all the visible layers: all figures of all visible layers are shown by the diagram (Figure 3). In fact, a layer acts as a classical diagram: it contains graphical data (representing figures) referencing the domain model elements (Figure 4).

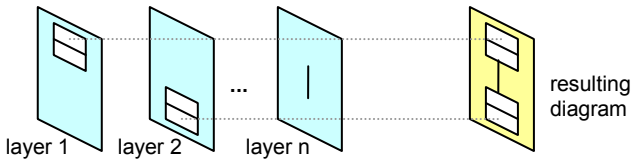

Figure 3 Diagram made of layers

A layer is associated to some graphical properties (background color, foreground color, font, opacity …) used as default for all the figures owned by it. Each figure can still override the default values. Changing a default property impacts all figures drawn by the layer. The layer has an extra property not found in figure: Opacity. It is used to control the opacity of all figures rendered by the layer. *Appearance* properties and *opacity* allow changing colors of a set of figures or to fade them to put them in background …

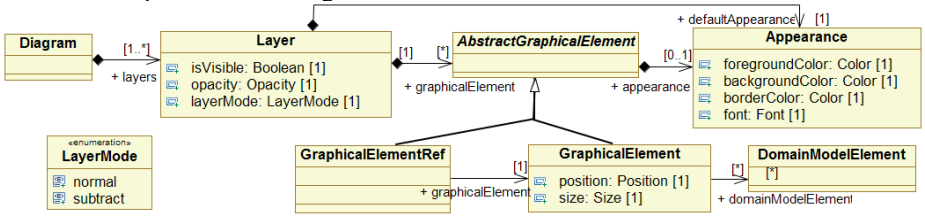

Figure 4 Diagram with layers

Each layer has also a property *'isVisible'* indicating if it is visible or not in the resulting diagram. If this property is set to true, all graphical data owned by the layer will be visible in the resulting diagram. If the property is set to false, the graphical data owned by the layer will not be visible. Thanks to this property it is possible to show different versions or views of a same base diagram: One layer contains the figures composing the base diagram, while other layers contain the variations. Playing with the *isVisible* property allows hiding and showing the different views.

The diagram can be animated by adding a controller switching the layers in a programmed sequence.

A diagram with only one layer behave exactly like a classic diagram: creation, modification and deletion of figures are done in the same way.

When other layers are added, one of them should be selected as the active layer. The diagram shows all the layers, but addition of new figures is done only in the active layer. In other world, the newly added graphical data will be held and controlled by the active layer.

A layer allows to override the appearance properties (like the ones found in the appearance property tabs of tools like Papyrus [3,4]: colors, font, properties to show …) of any figure owned by another layer. For this, the layer can hold a new set of appearance properties for a figure (by the way of the class `GraphicalElementRef` in Figure 1). When an appearance property is set with different values in more than one layer for a figure, then the value found in the upper layer is taken into account. With this mechanism, it is possible to change the appearance property of an existing figure from another layer. For example, a layer can be used to set to true the "`show stereotype`" properties of figure drawn in another layer. Thus, it will be possible to hide or show the stereotypes by hiding or showing the layer.

The position and size of an element are not part of the appearance properties, and thus can not be overridden with this mechanism. These two properties are only set in the layer owning the figure. This restriction is made to avoid layout problem and some potential problems linked to elements contained in other elements (like a class owned by a package in one layer and moved outside the package in another layer).

To ease diagram manipulation, the position and size of an element can be changed from any active layer, even if the active layer is not the owner of the figure. The new position and size

will be set in the graphical data of the layer owning the figure, and not in the active layer. Also, while a figure is associated to one and only one layer, it is possible to change its owning layer at any time.
The layer mechanism allows to draw a class in one layer, and its members (like its properties) in another layer. But what will happen if the layer containing the class is not visible while the layer containing the members is visible? In our proposal, when there is a containment relation and the container is not visible, then all contained elements are not visible, even if they are owned by a visible layer. So, in the previous example, the members will not be visible, which is the expected behavior.
A similar behavior exists for connections. A connection is a kind of figure connecting two elements. In our proposal, if one end of the connection is not visible, then the connection and its satellites (labels, roles, cardinalities …) are not visible. This behavior avoids dangling connections.
A layer can work in two different modes: normal or subtract. The normal mode behaves as describe until now: all figures owned by a layer are added to the resulting diagram. The subtract mode allows to subtract existing elements from a diagram: all graphical elements referenced by a layer will be subtracted from the resulting diagram.

4 Multiple Views in One Single Diagram

With layers, it is easy to realize multiple views with one single diagram: A base view is realized in a layer while other layers are used to add (or hide) the elements requested by other views. It is now possible to switch from a view to another by enabling the corresponding layer. It is also possible to combine several views simultaneously by enabling all the corresponding layers. The maintenance of all the views is easy as there is only one diagram to maintain: additions of new figures or modifications are immediately available to all layers, and so to all views.
The example of Figure 1 can be rewritten in one single diagram as follow: a first layer draws the class and the public members (*Public View*); another layers draw the details (*Details View*); A third layer could color the details in red; it is also possible to add a layer showing the stereotypes, another showing the comments … If we want to add a new class and an association to the diagram, it is done on the first layer. It will be immediately available to all views. One can see the different views or combine them by selecting the corresponding layer.
Layers based diagrams can also be used to animate diagrams. For example, it is possible to animate States Diagram or Activities Diagram: all States or Activities are drawn in one layer, and for each state or activity there is an additional layer used to highlight it, to add comments, to show objects flowing between activities ... The diagram can be animated by programming the order into which the layers will be visible.

5 Conclusion and Future Works

Our proposal to introduce layers into diagrams allows to numerous view on a model while reducing the number of diagrams (which can be seen as a paradox) and without increasing the maintenance cost. The proposal is purely graphical, it changes only the way a diagram can be rendered, and has no impact on the domain model itself.
The proposal also offer other possibilities not much exploited in today tools, like the ability to animate diagrams.
The proposal is under development in the near to come version of Eclipse Papyrus, an open source UML Tool [4].

Some extensions are also planned. The first extension will bring a link between layers and UML models. For this, we will provide a UML profile allowing to control layer properties from within a stereotyped UML model. As example, this will allow animation of a diagram with the help of an activity diagram.
Another extension will allow to associate automatically model elements to layers. Actually elements are created into the layers from a palette, or by dragging them from other diagrams or from outline. The automatic way will allows to associate a user defined expression, like "all classes of diagram X" or "all methods starting with Xyz". The results of the expression will be associated with the layer. The system will use an automatic layout to place the newly added elements. As example, this can be used to have diagram reflecting automatically the content of a package, or a class showing automatically all its public methods. The extension will not impose any language, but will allow usage of any existing language supporting queries like OCL.
This extension will also allow to automatically build views based on a stereotype or a tagged value, for example building views based on a version number, a developer name …

References

1. http://en.wikipedia.org/wiki/2D_computer_graphics#Layers (2009)
2. GIMP, the GNU Image Manipulation Program, http://www.gimp.org/ (2009)
3. Papyrus UML Tool, http://www.papyrusuml.org (2009)
4. Papyrus (Eclipse), http://www.eclipse.org/modeling/mdt/?project=papyrus (2009)