Difference Visualization for Models (DVM)

Visualizing model changes directly within diagrams

Andreas Scharf, Albert Zündorf
Kassel University, Software Engineering,
Department of Computer Science and Electrical Engineering,
Wilhelmshöher Allee 73,
34121 Kassel, Germany
[andreas.scharf | zuendorf]@cs.uni-kassel.de
http://www.se.eecs.uni-kassel.de/

ABSTRACT

Today's software development and maintenance is time consuming, cost-intensive and particularly an iterative process. Since models and diagrams are the main artifact in the development process of numerous research institutes and software companies, it is necessary to show and merge differences as the model evolves. While there are plenty of difference tools available for textual artifacts (like source code) this does not hold for diagrams.

This paper presents an approach to show and merge deltas of different model versions directly within the corresponding diagram editor. This is done by integrating the *Difference Visualization for Models (DVM)* framework into existing editors with as little effort as possible.

1. INTRODUCTION

Modern software consists of several million lines of code which are changed frequently by software development teams. Producing software does not mean to simply write the code once and never touch it again. In fact it is an iterative process: Bugs have to be fixed and the team has to take care for changing requirements which increases the code size. Every change (or *delta*) can be interpreted as a new version of the source code. This in turn leads to different versions of the same document which are mostly managed with version control systems like CVS or subversion.

During the development phase of a software project there often is the need of showing deltas between different versions of the same document. In the majority of cases these documents are interpreted as flat and unstructured text - content and logical configuration are not considered. The user then gets information about added, deleted and modified lines and is able to merge both versions of the document. The tool support for visualizing such changes for unstructured text documents like source code is excellent. However for more and more developers the source code is not the primary artifact anymore.

To design structure and behavior of large software projects the Unified Modeling Language (UML) [3] is used by developers. Particularly class diagrams are utilized to describe (parts of) systems. Like source code, diagrams are subject to frequent modifications which also raises the requirement of visualizing and merging these changes. Unlike source code, diagrams are structured files. Hence the traditional algorithms and tools to compute, visualize and merge deltas can not be applied in a meaningful way.

Existing approaches for visualizing model based changes are either difficult to use or hard to implement. The EMF Compare [5] for example provides a generic tree editor which is difficult to use because the UI is completely different from your usually used editor. The approach presented in [2] on the other hand integrates such a mechanism into Fujaba. This allows you to visualize differences directly in the class-diagram editor by the cost of duplicating the meta model to annotate the model elements with delta information.

We present an approach to visualize model based deltas directly within their corresponding diagram editors. Because you are already familiar with these editors you don't have to invest time to learn a new tool. Instead overlays techniques are used to enhance the existing GUI to visualize the deltas and give you the possibility to merge the different versions. The *Difference Visualization for Models (DVM)* framework may be integrated into existing Eclipse diagram editors build with the Graphical Editing Framework (GEF). This is done by providing new classes on the one hand and by exposing some Eclipse extension-points on the other hand. EMF Compare is used to calculate the delta between two model versions but the DVM framework is open for your model compare framework. To evaluate the DVM framework it has been integrated into the modeling IDE UML Lab [6].

2. MOTIVATION

Software artifacts like source code or models are changed frequently. If the software contains bugs and you know the point in time where everything worked well you might consider to have a look at the changes that were applied to the code between these two times. Another scenario is using a version control system: Before you update or commit your changes you often review the changes between your working copy and the remote files.

Thus, the correct computation and particularly the visualization of deltas is a central aspect in a team for an error-free and efficient work process. For you as the developer it is crucial to understand which changes occurred, if they are meaningful or not and if they correlate with your work in any way. Besides the visualization it is important to also provide merge functions to combine your local and remote artifacts.

The described features are well supported concerning unstructured text files like source code. If you use the synchronizing view in eclipse for example to review changes in shared software artifacts like source code, algorithms like the

Longest Common Subsequence (LCS) [1] are used to compute the delta between different versions of files. You then get a line based comparison of the changes mostly displayed in a simple text editor. While this approach is sufficient for unstructured artifacts like source code this does not hold for the model based development approach because of two reasons:

- Computing the delta of models by using algorithms like the LCS is not possible. Even if you consider a XMI representation of your model there are several disadvantages using a text based approach. As an example moving a model element yields removed and added lines in the XMI document and you loose the semantic of the move operation.
- Because models are modified by using special diagram editors displaying the differences in a simple text editor is not an option. Furthermore it is eligible to really use your existing and well known editor rather than using a dedicated tool because this saves time to learn a new tool.

Even though we did not focus on computing the delta of models we want to point out some approaches to better understand the requirements that a proper framework should meet. The technique presented in [4] is based on recording every single change that occurs in a model and simply save the change stream. This way it is possible to compute the delta in an elegant way. Using heuristics to match elements of different model versions and then compute the delta is another way to solve this problem even if finding good heuristics is not a trivial task. EMF Compare uses several weighted heuristics to compute the delta and serves as the diff-algorithm in our solution.

Mostly you are in the situation that you already have a diagram editor and only want to integrate the described functions with as little effort as possible. This requires a potential framework to be as generic and flexible as possible regarding the computation, visualization and merge functions. To our knowledge there exists no suitable solution which meets all the mentioned requirements. The approach presented in [2] is the closest match because it integrates a delta visualization mechanism into the classdiagram editor of Fujaba but does not provide any merge functionality and is really limited to this editor.

3. THE DVM FRAMEWORK

As already mentioned it is a good idea to visualize model based deltas directly within their corresponding diagram editors. In the eclipse environment the preferred technology to create diagram editors is GEF. The DVM framework allows you to add the difference visualization and merge functionality into existing GEF based editors with as little work as possible.

To be able to recognize the deltas within the diagram at first sight and distinguish them from other diagram elements we decided to use an overlay technique. This has some advantages:

- Overlays can be distinguished from your diagram elements easily.
- The UI for visualizing deltas does not depend on the UI used by the diagram editor which means that the overlays can be used generically. Changing the editors UI does not affect the delta visualization.
- To integrate the delta UI elements with your existing editor you don't need to change your figures.
- The design of the delta UI elements is very similar to the presentation used by conventional text based editors. Thus you intuitively know what is meant by a given delta UI element.

In the following sections we point out some details of the DVM framework beginning with the UI representation of differences, followed by the merge mechanism and then discussing some implementation details.

3.1 Overlay technique

Figure 1 shows an example used to visualize a delta concerning a changed attribute of a class within a classdiagram.

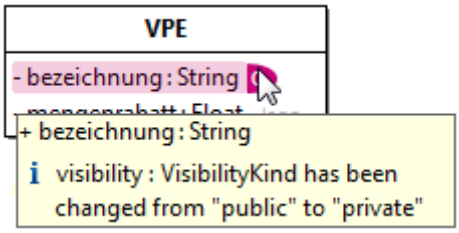

Figure 1: Tooltips provide detailed information

Beside the visualization for any type of node (e.g. classes, attributes, operation etc. in classdiagrams) it also important to support diagram edges. Figure 2 shows how the DVM framework visualizes a deleted association. In addition to the edge itself, all elements that belong to this edge are are overlayed with a delta annotation.

As you can see from figure 2 it is not clear at the first sight what exactly has been changed. To provide more information about the change and support merging of deltas the overlays are extended with special markers. Hovering over these markers populates a tooltip which displays detailed information about the delta. Figure 1 shows an example of a changed visibility of the attribute `bezeichnung:String`.

3.2 Context dependent delta visualization

The kind of the delta of course has great influence on the overlay generated by the DVM framework. Table 1 gives an overview of the possible delta types along with the letters used for the special markers and the default colors.

Type	Abbr.	Description
ADD	A	Element added
DELETE	D	Element removed
MODIFY	C	Element modified
MOVE	M	Element moved

Table 1: Default colors of delta types

The DVM framework takes care for choosing the correct overlay in case of added or removed elements. This operation is context dependent. An example: Let's assume that you have a class `Person` in your local model which does not exist

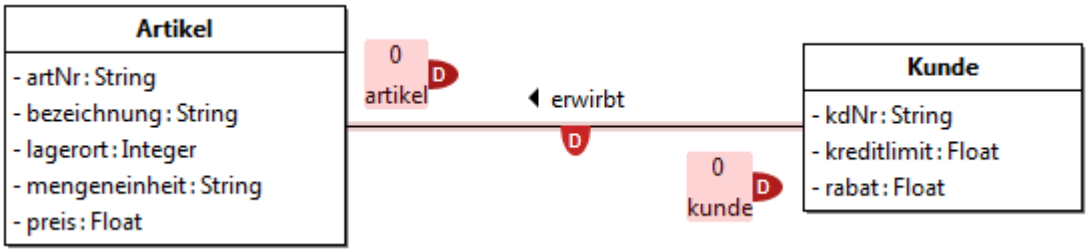

Figure 2: Overlay to visualize a delta for an association

in the remote version. Further your local version of the model is newer than the one on the server then this means that the class `Person` has been added and thus the DVM framework generates an overlay of type `ADD`. If the remote version is newer than your local one then the framework generates an overlay of type `DELETE`. In the former case this can be called *forward-diff* and in the latter *backward-diff*.

3.3 Merging deltas

We already talked about the special marker which extends every overlay. Besides the tooltip which gives detailed information about the delta, the marker is clickable allowing you to trigger merge actions. Figure 3 shows a marker menu exemplary for a delta of type `MODIFY`. This enables you to either accept oder discard this change.

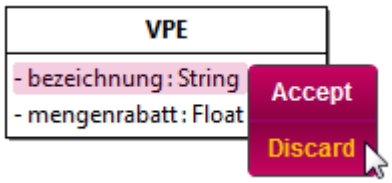

Figure 3: The merge menu

The displayed text in this menu also depends on the delta kind. If you have a delta of kind `ADD` for example then you have the choices *Keep* and *Remove* instead of those shown in figure 3.

3.4 Diff Algorithms

As already stated we did not focus on implementing a new diff algorithm to compute the delta between different versions of a model. Instead we provide a `DiffAlgorithm` and a `Merger` interface. The former provides access to the underlying diff algorithm while the latter is able to execute merge operations. The DVM engine uses the `DiffAlgorithm` interface to compute the delta between two versions of a model and transforms the result of this operation into an internal `DiffViewerModel`. This model is then used for further operations like generating the overlays or triggering merge actions. Using an additional model rather than simply use the EMF Compare model has the advantage of decoupling the DVM framework from special diff and merge implementations.

The default `DiffAlgorithm` is the `EMFCompareDiffAlgorihthm` which is able to compare ecore based models. The `EMFCompareMerger` is used to merge the deltas

3.5 How to use DVM

We now want to give a brief overview of how the DVM framework can be integrated into existing GEF diagram editors. To display the generic overlays you have to annotate your model with the delta information provided by the DVM `DiffViewerModel`. This is done by implementing the `DiffViewerAnnotator` interface and executing the following steps:

1. Merge the delta into your local model if the delta kind is `DELETE`. This is necessary to display local deleted elements.
2. Find the element related to the delta in your local model and register it in the DVM framework
3. Undo any changes made in the first step if the delta kind is `DELETE`. Note that care must be taken to not remove any visual representation of your local element because otherwise the deleted element can not be displayed.

Steps 1 and 3 need a bit more explanation: If you want to display a locally deleted element (which means that you do not have the element in your domain model) you first have to get this element into your domain model and create the corresponding visual representation. This also means that the DVM framework needs to modify your local domain model even if you just want to display the differences. We decided to undo any of these changes made in the domain model but keep the visual representation in Step 3. Another way could be to not undo the changes, keep track of the modified elements and undo the changes for discarded deltas and do nothing for accepted ones. This really is the most difficult step while integrating the DVM framework into your editor. We did not find a generic way to do this because the mechanism to undo changes is specific to your application.

After implementing the `DiffViewerAnnotator` you have to expose this class to the DVM framework using a special eclipse extension point provided by the DVM framework. In the last step two new GEF EditPolicies, `DiffVisualizationNodeEditPolicy` and `DiffVisualizationEdgeEditPolicy`, are used to visualize the generic overlays and provide support for merge actions. EditPolicies in GEF are the controllers in the MVC design pattern which means that they can get information for both model and the corresponding visual representation. Depending on whether your EditPart is a node or an edge you register the former or the latter EditPolicy on your EditPart. If now a new delta is registered in step 2, both EditPolicies are able to find the correct GEF figure, determine it's size and position and finally create the

generic overlay. To visualize all deltas in a new diagram you have to implement the DVM `DiagramCreator` interface. Existing diagrams can be found by implementing the `DiagramFinder` interface.

As you can see, integrating the DVM framework is relatively easy. To evaluate the framework it has been successfully integrated into the classdiagram editor of the modeling IDE UML Lab. Figure 4 contains a screenshot of the classdiagram editor of UML Lab extended by DVM delta overlays.

4. RELATED WORK

To our knowledge the number of comparable work is very low and there is no framework like the DVM. However we want to present two approaches to compare models and visualize deltas.

We already mentioned the EMF Compare framework. This framework consists of a couple of eclipse plugins and thus is intended to be used inside the eclipse environment. EMF Compare provides algorithms to calculate the delta between two versions of an ecore model. Therefore as long as your model is an ecore model you can use the generic diff algorithm of EMF Compare. Besides the diff functionality EMF Compare provides a generic tree editor to visualize and merge the differences. This generic tree editor both has advantages and disadvantages: On the one hand you can use your ecore model without modification. On the other hand it means that you have to learn new tool instead of using your own diagram editor.

The solutions presented in [2] integrates the visualization of deltas into the classdiagram editor of Fujaba. This is done by importing a special XMI file which contains the model elements along with the delta information. To annotate the model elements a duplicate of the meta model which is extended by the delta model elements is necessary. Furthermore special components are required to finally display the changes within the classdiagram editor. The workflow to compute and visualize the delta between two versions of a model is complex in this approach because you have to provide the mentioned XMI file by yourself with third party tools. Additionally the delta visualization support is limited to the classdiagram editor and it is not possible to merge any differences.

5. SUMMARY AND FUTURE WORK

In this paper we have presented an approach to visualize differences between models directly within the corresponding diagram editors. The DVM framework can be integrated in GEF based diagram editors in the eclipse environment and supports both visualizing and merging of deltas. We evaluated the DVM framework by integrating it into the classdiagram editor of the modeling IDE UML Lab and saw that the approach works as intended.

The framework lacks some features. For example a three-ways-diff is not possible by the time writing this paper. The DVM framework therefore cannot visualize any conflicts as they involve three versions of a model: The local, the remote and the base. As this is necessary while working in a team which uses some kind of version control system we plan to implement this feature as soon as possible.

Furthermore we have some ideas where the DVM framework could be used as well. One example is the visualization of the delta triggered by a refactoring operation. If the refactoring is complex it may not be obvious what exactly has changed in the model. Thus visualizing these changes is a useful feature.

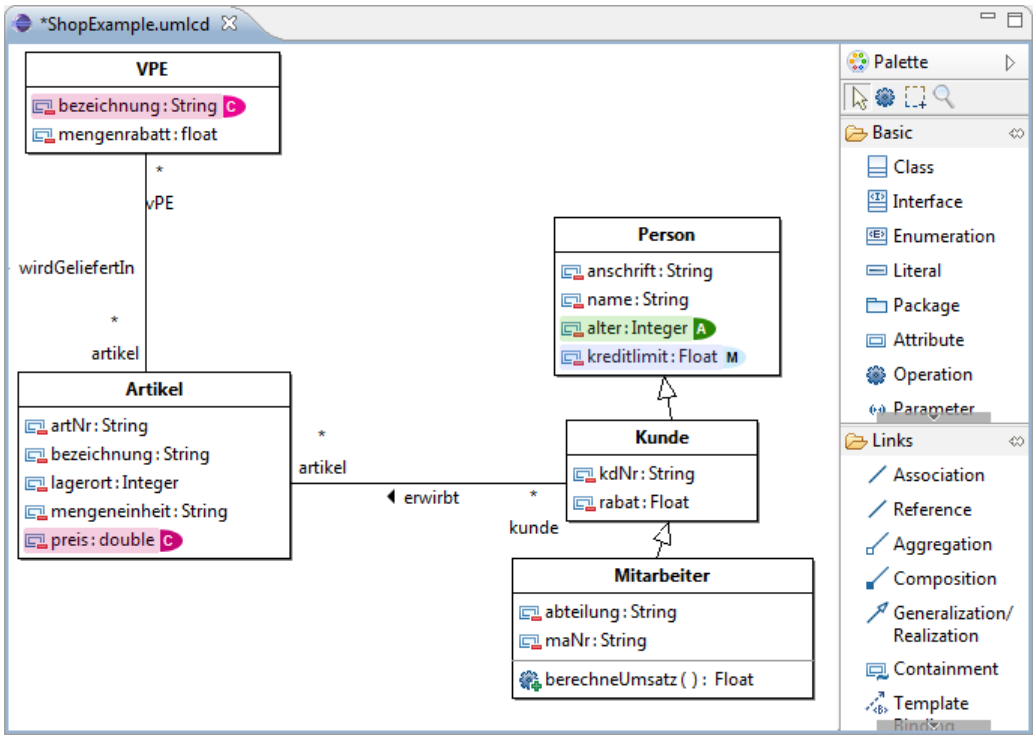

Figure 4: DVM framework integrated in UML Lab

6. REFERENCES

[1] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proc. Seventh International Symposium on String Processing and Information Retrieval SPIRE 2000*, pages 39–48, Sept. 27–29, 2000.

[2] S. Lück. Ein differenzanzeige- plugin für klassendiagramme in fujaba. Master's thesis, Universität Siegen, 2004.

[3] Object Management Group. Unified modeling language.

[4] C. Schneider. CASE Tool Unterstützung für die Delta-basierte Replikation und Versionierung komplexer Objektstrukturen. Master's thesis, Technische Universität Braunschweig, 2003.

[5] N. Skrypuch. EMF Compare. http://www.eclipse.org/modeling/emf/?project=compare.

[6] Yatta Solutions GmbH. UML Lab. http://www.uml-lab.com/, 2009.