Visual Comparison of Graphical Models

Arne Schipper, Hauke Fuhrmann, Reinhard von Hanxleden
Christian-Albrechts-Universität zu Kiel
Department of Computer Science
Real-Time and Embedded Systems Group
{ars,haf,rvh}@informatik.uni-kiel.de

Abstract

Collaborative development, incremental design and revision management require the ability to compare different versions of software artifacts. There are well-established approaches for comparing textual artifacts such as program files. However, the situation is different in graphical modeling. So far there exists little support to compare models visually—graphically in the model diagram itself. This paper presents several possible approaches and explores one of these in further detail.

We apply paradigms of visualizing text files side-by-side to graphical diagrams and enhance the views by additional features such as automatic layout, navigation and folding. These means allow to compare even complex models without missing differences or getting lost in text based structure compares. As a proof of concept, the proposal is implemented in KIELER, a prototypical modeling environment based on Eclipse.

1. Introduction

Graphical modeling aims at creating higher abstractions of a system by displaying graphical representations (model *diagrams*) instead of plain text. Diagrams offer multiple dimensions—usually two—instead of only the one-dimensional texts. This is used to depict the contents in an appealing and comprehensible way. The diagrams often are some kind of graphs, where the graph itself holds its semantic. Inherently graphs can be represented by many different embeddings, and a given embedding can be drawn in different layouts. Additionally some information that may have changed is not at all reflected in the diagrams but only within some internal properties of graphical elements. Therefore it is often difficult to manually compare two graphical representations to see what items are different and even what parts are the same when only diagram layout has changed. Therefore computer assistance in finding changes is a crucial feature in collaborative and iterative development.

It is of great importance to the success of system modeling that tools offer intuitive and easy to use interfaces to create and change the model. A major concern is the depiction of changes in graphical models in a graphical way, visualizing the changes in the same manner in which they were produced. This prevents the user (of the modeling tool) from switching of different abstraction levels, when trying to map the textual description of the differences to the diagram. As observed by Mehra et al., graphical comparison is an advantage, as this is the natural way to compare visual objects [7]. The still prevailing approach of converting the differences to some structured text is just a workaround due to the lack of better methods. Transforming the textually displayed changes back to the graphical world requires, according to Green, a "hard mental operation" [4], which is unnecessary and should be avoided. Depicting changes in the diagram itself helps the developer working with it to understand the resulting modification in its meaning immediately. As Ohst et al. point out, it is not appropriate for two-dimensional documents like diagrams to display possible changes in the traditional way, in two linear columns with corresponding elements facing each other [9]. In most cases there is a reason why objects in a diagram are positioned like they are, and should not be realigned, at least not in just one dimension.

That there is a real need for a visual comparison is also supported by commercial applications recently introduced that try to provide at least some limited form of graphical comparisons. SCADE Model Diff (Esterel Technologies) and ecDIFF (Expert Control), intended for Simulink (Mathworks) models, were introduced into the market in 2007 and 2008.

We here propose different alternatives to display differences of graphical models. We combine well established means, like structural comparisons and colored side-by-side confrontation, with advanced model presentation and interaction techniques, like automatic layout, navigation and folding. To validate our approach for a wide and extensible variety of graphical modeling languages, we implemented a prototypical comparison facility in a modeling environment based on Eclipse[1]. Our illustration focuses on Statecharts, one behavioral modeling language of the UML. However, our techniques are language independent and should be applicable to other languages as well, including those of UML/AADL. The contributions of this paper are thus 1) an investigation of general mechanism to visually display model differences, 2) the augmentation of these mechanisms with customized views, and 3) an outline of how this is implemented in a general-purpose modeling environment.

This paper is structured as follows. The next section discusses related work. In Section 3 we examine different types of visual comparison. The implementation is presented in Section 4, results are summarized in Section 5. The paper closes with a conclusion and outlook. As space is limited, we cannot present our proposal here in much detail. A more in-depth presentation, including background on automatic diagram layout, the implementation, and case studies, can be found in Schipper's thesis [13].

2. Related Work

Beginning with the diff tool [6], the comparison of content initially took place at the textual level. The first steps in comparing non-flat file data were taken in database applications, but those worked only on relational data. Chawathe et al. elaborated the ability to compare hierarchically structured information, satisfying a rising demand resulting from the immense growth of the amount of structured data in general [2]. This method is also applied to documents written in the eXtensible Markup Language (XML) by Ohst et al. [9]. First implementations emanated from the XML content of model files and used the XML elements as a base for structural comparison. The representation was in a tree structure. Applications working with this paradigm are The Compare Utility (Spark Systems) or the XML Differencing tool (Stylus Studio).

Considering actual models, an interesting approach is used by CoObRa [15]. The model elements themselves are considered as objects in a Version Control System (VCS). Every operation carried out by the user to the model elements in the Integrated Development Environment (IDE) is mapped to an operation on the object in the VCS. Only these change operations are saved, so this mechanism saves storage space and the difference computation between the versions is derived for free. A client-server concept is used to enable multiple developers to work on one project.

A generic approach in comparing and merging uses SiDiff [14, 17]. Input models are transformed to an internal data structure. The structure-based diff is then performed with this data, leading to a generic description of the differences. Depending on the type and semantics of the input models, the output must be interpreted in an appropriate manner to obtain the differences in the domain of the original model.

Both of the above concepts were implemented as plugins in the round trip engineering tool FUJABA[2], still with the lack of an appropriate graphical representation. A similar method is used by EMF Compare of Brun and Toulmé [1, 16], employed in the Eclipse Modeling Framework (EMF)[3]. EMF Compare is limited to display the changes as structured data in a tree-like view, and again, no graphical facilities are given. However, as further explained in Section 4.2, we can build on EMF Compare to *compute* differences, and enhance it with means to graphically *visualize* the differences.

Another work focusing on Eclipse is a plug-in suite of Mehra et al. [7]. The input model is mapped to Java objects on which the comparison is performed in a generic way similar to EMF Compare and SiDiff. They also provide support to display the differences in a graphical way. However, there are some drawbacks, as the two versions are drawn into one diagram with a re-computation of the layout and ugly overlappings may occur.

The aforementioned SCADE Model Diff is designed to analyze differences between two SCADE models or two versions of a model. The differences are represented in terms of *added*, *deleted*, *changed*, or *moved* elements. Only the semantics are taken into account, no layout information. The results are presented in several ways, in a *diff tab* showing all the differences in a list, in a *diff window*, displaying two tree structures side by side, or in a *location window*, exhibiting two graphical models—SCADE models—with highlighted differences. Furthermore it is possible to generate a textual report of the changes. Unfortunately the tool is limited to the SCADE own dataflow and behavioral languages, and it does not provide advanced features like automatic zooming/panning/folding.

Girschick presents an algorithm to detect and display changes in UML class diagrams [3]. This algorithm is specific to UML class diagrams. Reports of changes are shown in an HTML page, including a graphical view of the merged diagram and a textual description of the changes. The view of the diagram is a merged version of both, which can lead to unaesthetic overlappings when much has been changed between the two versions.

Another example is the plug-in for Pounamu, a meta-Computer-Aided Software Engineering (CASE) tool developed by Zhu et al., using a more generic approach that is not limited to one kind of model [19]. The method to display changes is more interactive and the user can, when checking out a newer version, see every change in the diagram

[1]http://www.eclipse.org

[2]http://www.fujaba.de

[3]http://www.eclipse.org/modeling/emf/

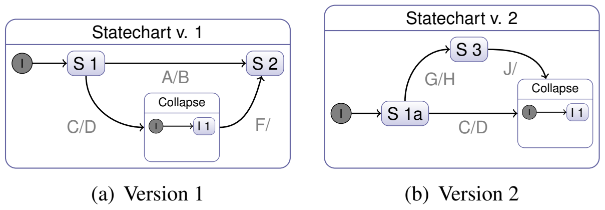


Figure 1. The two original versions of the example diagram.

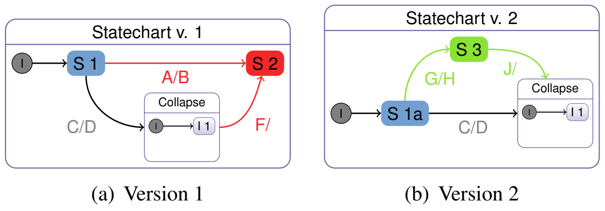


Figure 2. Plain visual diff. Color legend: green/additions, red/deletions, blue/changes.

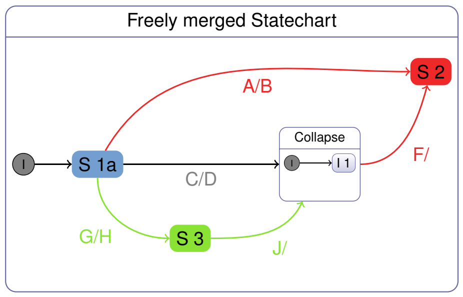


Figure 3. Freely merged visual diff.

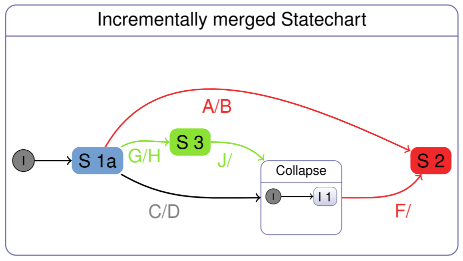


Figure 4. Incrementally merged visual diff.

immediately and accept or reject it. The graphical representation works with two layers on top of each other, one for each diagram.

Overall, there still appears to exist little work on how these differences are presented best to the user, especially when they are supposed to appear in the diagram itself. There are several works considering graphical languages, perception and representation, correlation between syntax and semantics, secondary notation, etc. [11, 12]. However, a sound solution for a graphical comparison still appears to be lacking.

3. Visual Comparison

A sound layout is vital for the correct understanding of graphical models [10, 5]. The automatic layout of graphical diagrams is not trivial, but not discussed further here. We assume that the modeling environment does provide an automatic layout facility, as is for example the case with KIELER (see Section 4.1).

The mental map of the user, that is the position of elements, their connections and sizes, should be preserved as much as possible to support the user's understanding of different diagram versions [8, 12]. This applies also to the visual comparison, when different diagram versions are presented to the user and they may be altered to visualize possible changes.

The following classification of possible visualization mechanisms combines previous proposals (presented in Section 2) with further alternatives. The two versions referred to can be selected by the user. This will generally be, but not necessarily, the actual version where he or she is working on and any older one. To illustrate the alternatives, we will compare the Statechart examples shown in Figure 1. The classification is as follows:

Plain: The two original layouts are just shown side by side, with colors or similar markers indicating differences. This is illustrated in Figure 2.

Animation: A small animation or video is created, which shows the transition from one version to the other by morphing the Statechart, thus maintaining the mental map of the user.

Pop-up: Having enabled the compare mode, the user can navigate through the one version of the Statecharts, which is annotated with modifications, and pop-ups will show in detail the changes that occurred in the neighborhood relative to the other version.

Free merge: A merge of the two versions is calculated. This merged model will be laid out from scratch, with colors showing alterations from one release to the next one. This can be seen in Figure 3, the coloring is like in *plain*.

Incremental merge: This is similar to the free merge and shown in Figure 4. The calculation of the merge remains the same. The layout is not computed from scratch, instead one of the original layouts serves as a reference for the merged layout, maintaining the mental map of the developer.

A side by side comparison, in its static case as described here, is the simplest way of comparing entities. The first thing coming into mind as an analogy for this type of comparison is the ordinary textual diff, enhanced by a graphical representation showing the versions in two columns with corresponding text blocks at the same vertical level.

There are several advantages in this mechanism. No new layout has to be computed. Just the two existing layouts are next to each other. In this manner, different colors could help the developer to discover the changes. This is particular true for *states* that just have changed attributes, a characteristic which cannot be detected in a graphical model at first glance.

Trying out the proposals manually showed that some of them are not adequate, like *pop-up*, and some, like *animation*, seemed to be too elaborate to be implemented within standard tools.

4. Implementation

As a proof of concept and to be actually able to test the visual comparison, we implemented an *enhanced visual diff*, i. e. the *plain visual diff* with additional features added such as automatic layout, folding, zooming, and panning. This naturally takes advantage of hierarchies in a model; we detect when an internal element of a model has changed, but see internal differences only when we are interested in them. The implementation was integrated into the existing experimental framework Kiel Integrated Environment for Layout for the Eclipse Rich Client Platform (KIELER).

4.1. KIELER

KIELER[4] is an experimental modeling framework to investigate the graphical model-based design of complex systems. KIELER is based on the Eclipse rich client platform and is modularly structured. Due to this, many plug-ins of the Eclipse world can be used. The KIELER Infrastructure for Meta Layout (KIML)[5] provides the automatic layout of different diagram types.

The graphical editors intended to use the visual comparison were created for the use in KIELER with techniques such as the Eclipse Modeling Framework (EMF), the Graphical Editing Framework (GEF)[6] and the Graphical Modeling Framework (GMF)[7]. For the initial implementation a Statechart editor was used, but the visual comparison works also

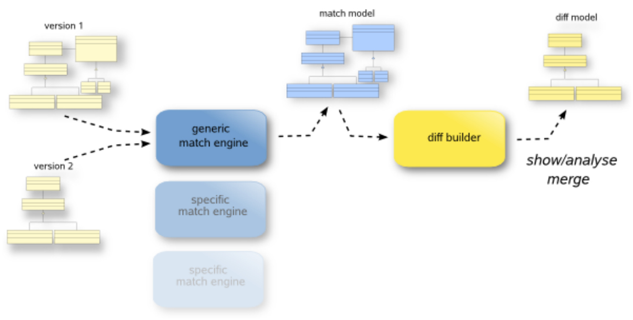


Figure 5. The structure of EMF Compare[8]

with a developed dataflow editor, as well as with the standard Eclipse UML tools, which are based on EMF/GMF, and any other EMF/GMF editor.

4.2. EMF Compare

As the main point of visual comparison is about the representation, existing supporting applications were used where applicable. Of particular benefit here was the aforementioned EMF Compare plug-in. This is a plug-in which extends the normal compare function of Eclipse by the support for EMF models. As depicted in Figure 5, first a *match engine* tries to find matches between the elements of the different versions with various metrics and computes a match model that is essentially a union of the compared models. Second, a *diff builder* extracts the differences into a *diff model*, which consists of *additions*, *deletions* and *changes*. The matching and differencing algorithm was inspired by work of Xing et al. [18]. Using EMF Compare, we can build on an established means to compute the differences, and can focus on just visualizing them.

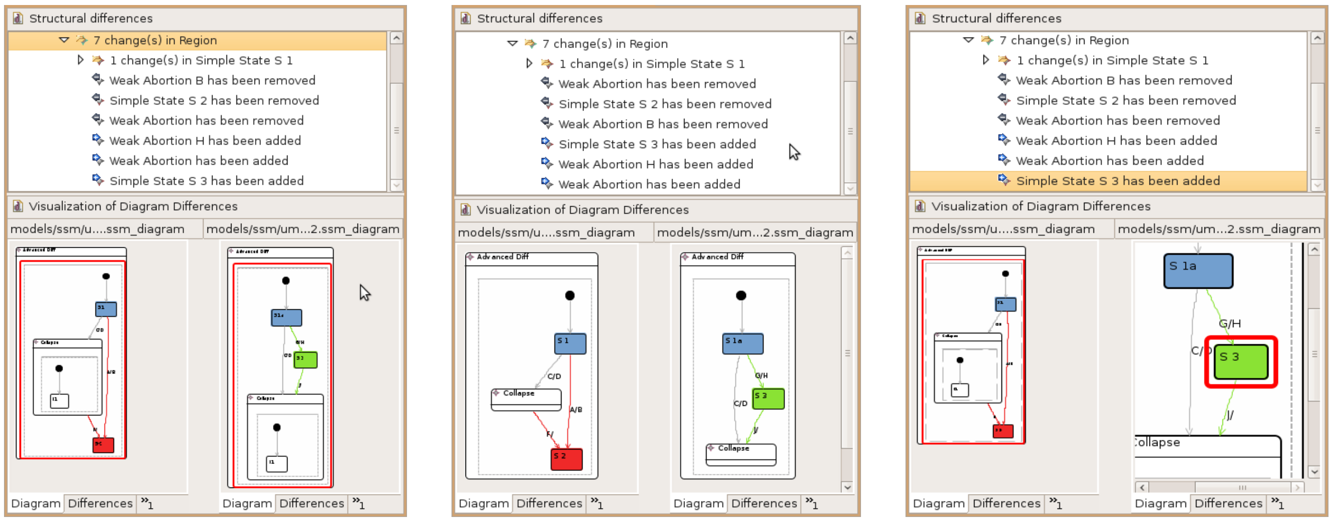
4.3. Navigation and Visualization

Figure 6 shows how the example diagrams from Figure 1 are compared by our visual comparison tool. We use the approach established by EMF Compare of side-by-side windows with the two versions of the diagram, with an additional third window on top that gives a structured textual description of the changes and guides the user through them.

It turned out to be very useful to provide the comparison tool with means to easily navigate through the changes. This was achieved by an adaptable click and zoom mechanism. Whenever clicking on an element in any of the windows—that includes also the top window with the textual description of the changes—, the other two windows scroll and zoom to the corresponding position in the diagram.

[4]http://www.informatik.uni-kiel.de/rtsys/kieler/
[5]http://rtsys.informatik.uni-kiel.de/trac/kieler/wiki/Projects/KIML
[6]http://www.eclipse.org/gef/
[7]http://www.eclipse.org/gmf/

[8]http://wiki.eclipse.org/index.php/EMF_Compare


Structural differences
Visualization of Diagram Differences
models/ssm/u....ssm_diagram
models/ssm/um...2.ssm_diagram
Advanced Diff
Collapse
S 1
S 1a
S 2
S 3
C/D
A/B
G/H
Diagram
Differences

7 change(s) in Region
1 change(s) in Simple State S 1
Weak Abortion B has been removed
Simple State S 2 has been removed
Weak Abortion has been removed
Weak Abortion H has been added
Weak Abortion has been added
Simple State S 3 has been added

(a) Collapsing disabled.

7 change(s) in Region
1 change(s) in Simple State S 1
Weak Abortion has been removed
Simple State S 2 has been removed
Weak Abortion B has been removed
Simple State S 3 has been added
Weak Abortion H has been added
Weak Abortion has been added

(b) State Collapse has been collapsed, as there are no changes inside.

7 change(s) in Region
1 change(s) in Simple State S 1
Weak Abortion B has been removed
Simple State S 2 has been removed
Weak Abortion has been removed
Weak Abortion H has been added
Weak Abortion has been added
Simple State S 3 has been added

(c) Auto-navigation zooms and scrolls to selected changes.

Figure 6. Enhanced visual comparison of two Statecharts.

Another sensible option is to collapse the regions of the diagram which are not of interest. In the particular case of Statecharts, it is possible to collapse *composite states* in which no change occurred to gain more space during the diagram view, and to draw the users attention to the actual changes. This is shown in Figure 6(b), in contrast to Figure 6(a). The automatic layout facilities of KIML are employed to use the space best.

The implementation is tightly integrated with the established Eclipse work flow. The visual comparison is launched when clicking on two diagram files, just as it is done with two textual files. It is also possible to compare a diagram file against its local history. The color scheme used to indicate additions, changes and deletions is the same as used by EMF Compare, see also Figure 2. This is consistent within Eclipse and can be customized by the user.

Automatic zooming and panning navigates the user to selected changes as depicted in Figure 6(c). Changes can be either selected in the structural view or in the graphical views. The zoom level might be different in the two graphical representations to show the affected objects and its context.

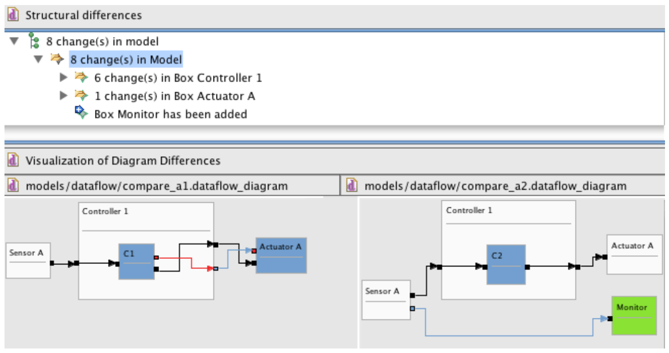
5. Results

We collected feedback within our work group and from an industrial development unit (Philips Medical Systems) that uses Statecharts. The overall evaluation was quite positive. The general implementation, the facility to perform a visual comparison as such, was well accepted. During the use some interesting benefits could be identified. Whereas simple structural changes of a diagram, such as removing or adding of a state, can be detected in a reasonable amount of time manually, especially for small changes the visual comparison was deemed very useful. Those small changes incorporate mainly the altering of attributes, on the one hand of states, which in most cases are not shown in a diagram directly, on the other hand changes of the triggers and effects of a transition. Those changes manifest themselves just in different letters drawn next to a transition and are hard to recognize graphically.

The main benefit when showing these differences of the diagram was the way they were presented to the user. In former difference representations the changes were shown textually to the user, who then had to identify them in the diagram later on to interpret them and to deduce the functional differences. In our approach, the changes are instead directly mapped to the two diagrams.

Other aspects mentioned positively were the ability to collapse regions which were not of interest and the synchronous scrolling and zooming of the diagrams.

A significant advantage of the generic EMF approach when working with models in the KIELER environment is that the visual comparison can be used with any EMF model, even with any editor, as long as it is generated with GMF. For example Figure 7 shows a comparison of two dataflow diagrams as used by Matlab Simulink, Ptolemy or SCADE. They were created with a GMF editor and automatically laid out with KIELER. Moving from Statecharts to dataflow diagrams did not require any extensions or modifications of our compare facility.



Structural differences

8 change(s) in model
8 change(s) in Model
6 change(s) in Box Controller 1
1 change(s) in Box Actuator A
Box Monitor has been added

Visualization of Diagram Differences

models/dataflow/compare_a1.dataflow_diagram
models/dataflow/compare_a2.dataflow_diagram

Figure 7. Visual comparison of dataflow diagrams.

6. Conclusions and Outlook

To provide facilities to compare diagrams graphically is a logical consequence of graphical modeling. The presented comparison tool is just a first step, the next stage would be to implement in the same manner views that support the users graphically when merging changes into diagrams.

It should also be worthwhile to implement and evaluate some of the other proposals presented here, such as free and incremental merge. With well-designed incremental layout algorithms the incremental merge could be useful, especially if space is scarce and preserving the mental map is crucial.

Acknowledgment

We thank Jan Täubrich and his colleagues from Philips Medical Systems DMC GmbH, Hamburg, for suggesting this topic and very helpful comments.

References

[1] C. Brun. Comparing and Merging Models with Eclipse: an Update on EMF Compare. In *EclipseCon 2008*, Santa Clara, California, Mar. 2008.

[2] S. S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 26–37, New York, NY, USA, 1997. ACM.

[3] M. Girschick. Difference detection and visualization in UML class diagrams. Technical Report TUD-2006-5, Department of Computer Science, TU Darmstadt, Aug. 2006.

[4] T. R. G. Green. Cognitive Dimensions of Notations. In *Companion Proceedings of the CHI '98 Conference on Human Factors in Computing Systems*, pages 443–460, Cambridge, UKDepartment of Computer Science, University of Calgary, 1989. Cambridge University Press.

[5] D. Harel. On the aesthetics of diagrams. *Lecture Notes in Computer Science, Mathematics of Program Construction*, 1422/1998:1–5, 1998.

[6] J. Hunt and M. McIlroy. An algorithm for differential file comparison. Technical Report 41, Bell Laboratories, July 1976.

[7] A. Mehra, J. Grundy, and J. Hosking. A generic approach to supporting diagram differencing and merging for collaborative design. In *ASE '05: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 204–213, New York, NY, USA, 2005. ACM.

[8] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[9] D. Ohst, M. Welle, and U. Kelter. Differences between versions of UML diagrams. *SIGSOFT Softw. Eng. Notes*, 28(5):227–236, 2003.

[10] M. Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44, June 1995.

[11] H. C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages and Computing*, 13(5):501–516, 2002.

[12] H. C. Purchase, E. E. Hoggan, and C. Görg. How important is the "mental map"? — An empirical investigation of a dynamic graph layout algorithm. In M. Kaufmann and D. Wagner, editors, *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 2006.

[13] A. Schipper. Layout and Visual Comparison of Statecharts. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Dec. 2008. http://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/ars-dt.pdf.

[14] M. Schmidt and T. Glötzner. Constructing Difference Tools for Models Using the SiDiff Framework (Informal Research Demonstration). In *ICSE 2008 Companion Proceedings, 30th International Conference on Software Engineering*, Leipzig, May 2008.

[15] C. Schneider, A. Zündorf, and J. Niere. CoObRA - a small step for development tools to collaborative environments. In *Workshop on Directions in Software Engineering Environments; 26th International Conference on Software Engineering*, Scotland, UK, 2004.

[16] A. Toulmé. Model Comparison Panel. In *EclipseCon 2007*, Santa Clara, California, Mar. 2007.

[17] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. Difference Computation of Large Models. In *ESEC-FSE '07: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 295–304, New York, NY, USA, 2007. ACM.

[18] Z. Xing and E. Stroulia. Differencing logical UML models. *Automated Software Engg.*, 14(2):215–259, 2007.

[19] N. Zhu, J. C. Grundy, J. G. Hosking, N. Liu, S. Cao, and A. Mehra. Pounamu: A meta-tool for exploratory domain-specific visual language tool development. *Journal of Systems and Software*, 80(8):1390–1407, 2007.