INSTITUT FÜR INFORMATIK

Taming Graphical Modeling

Hauke Fuhrmann, Reinhard von Hanxleden

Bericht Nr. 1003
May 2010

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Taming Graphical Modeling

Hauke Fuhrmann, Reinhard von Hanxleden

Bericht Nr. 1003
May 2010

e-mail: haf@informatik.uni-kiel.de,
rvh@informatik.uni-kiel.de

Technical Report

Visual models help to understand complex systems. However, with the user interaction paradigms established today, activities such as creating, maintaining or browsing visual models can be very tedious. Valuable engineering time is wasted with archaic activities such as manual placement and routing of nodes and edges. This report presents an approach to enhance productivity by focusing on the *pragmatics* of model-based design.

Our contribution is twofold: First, the concept of *meta layout* enables the synthesis of different diagrammatic views on graphical models. This modularly employs sophisticated layout algorithms, closing the gap between MDE and graph drawing theory. Second, a *view management* logic harnesses this auto layout to present customized views on models.

These concepts have been implemented in the open source Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER). Two applications—editing and simulation—illustrate how view management helps to increase developer productivity and tame model complexity.

Contents

List of Figures

1 Introduction

The main task in software engineering is to command the computer to do the right thing. The programming mechanics of computers has undergone quite an evolution: From manually stamping programs on punch cards over non-reversible type writers to the main method still used today—text editor and keyboard. While different IDEs might offer various support levels for large software artifacts, the basic mechanics of writing or changing a line of code is rather standard and efficient. Hence editing text has been established for many decades.

The introduction of graphical models has added the second dimension to one-dimensional text. However, this new freedom comes at a heavy price: We are back to the early times of mechanical typewriters with rather archaic user interactions. Graphical layout has to be manually defined by placing and routing of nodes and edges. Deleting graphical objects, like using white-out on a typewriter, creates new white-space that might not be large enough to insert new expressions, i. e. new graphical constructs. Manually creating more space in a complex diagram is like using scissors and glue. In fact, in large, industrial projects it is is not uncommon that highly-paid engineers use scissors and glue to create large hand-crafted posters from print-outs to help navigate through complex models.

Graphical views on models are manually defined and hence static like a type-written piece of paper. Creating multiple different views, e. g. for different levels of abstraction, onto the same model requires much manual editing work. Often one ends up working with one single abstraction level or changing syntax from graphical to e. g. structural to get more detailed or more abstract representations. Although abstraction might play an important role for MDE [19], so far, graphical aspects of models certainly do not. Instead of unfolding their potential as a vivid means of communication they remain no more than syntactic sugar. When trying to communicate with the computer through graphical models, the computer will not answer in the same language. For example, model transformations typically loose the graphical information and result in a model without a graphical view, which is like typing in text and getting a punch card as answer. If one believes that a diagram communicates the meaning of a model better than another representation, and if one wants this to be widely accepted by domain users that are not necessarily computer scientists, then one has to teach computers to truly master this language.

This paper presents an approach to bridge the gap between MDE and graph drawing theory to enable the automatic processing of graphical models and fundamentally enhance the user interaction mechanisms. After the related work in Chap. 2, Chap. 3 gives the required terminology and defines the focus of our approach—*pragmatics*. Chap. 4 introduces the central contributions: First, Sec. 4.1 explicates how *meta layout* enables

the synthesis of different diagrammatic views on graphical models. Meta layout offers interfaces to plug in sophisticated layout algorithms and to utilize them according to higher-level optimization criteria. Second, Sec. 4.2 presents how *view management* logic employs this auto layout to dynamically and interactively present custom views on models. Chap. 5 illustrates these concepts with the open source Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER). Sec. 5.2 discusses two fields of application—model editing and simulation. Chap. 6 presents an experimental evaluation, the paper concludes in Chap. 7.

2 Related Work

With the work presented in this paper we try to fundamentally enhance the user interaction paradigms of graphical modeling—the pragmatics defined in Chap. 3. This is an interdisciplinary task and hence there is a large body of related work emerging from related communities.

The MDE community employs means of user experience enhancements orthogonal to ours [30]. There are multiple recent approaches on creating model-to-model transformations *from examples* instead of complex transformation languages [5]. It would be interesting to combine such approaches with the structure-based editing framework presented in Sec. 5.2.2 to give the user very natural ways to define custom editing operations him- or herself. Also transformation languages based on triple graph grammars [4] could augment structure-based editing by graphical views on the transformations themselves.

The field of *Human Centred Software Engineering* [13] also addresses usability and productivity. However, these approaches mainly focus on the question on how to make the best user experience with a given product. In contrast, we try to enhance the development process itself with novel tool support.

Another related community focuses on software visualization [8], which mainly presents what we call *effects* on graphical views (cf. Sec. 4.2). We also employ the notion of *focus & context* by Card et al. [6], see Sec. 5.2. Musiel and Jacobs [23] apply this technique to UML class diagrams, using notions of *level of detail* and a rudimentary specialized automatic layout algorithm. In our approach to view management we try to generalize such ideas by orchestration of software visualization concepts (effects) with the context (triggers) in which they should be applied to dynamically synthesize graphical views on models.

Automatic layout problems for arbitrary diagrams are often NP-complete, and diagram quality is difficult to measure [27]. However, the graph drawing theory community emerged with sophisticated algorithms that solve single layout problems efficiently with appealing results [3, 18]. There exist open layout library projects with multiple sophisticated algorithms such as the Open Graph Drawing Framework (OGDF) [7], Graphviz [12] and Zest, which is part of the Eclipse Graphical Editing Framework (GEF). There are also commercial tools such as yFiles (yWorks GmbH) and ILOG JViews [29].

The KIEL project [26] evaluated the usage of automatic layout and structure-based editing in the context of *Statecharts*. It provided a platform for exploring layout alternatives and has been used for cognitive experiments evaluating established and novel modeling paradigms. However, it was rather limited in its scope and applicability, hence it has been succeeded by the KIELER project, which is the context of the work presented here.

3 Pragmatics

In linguistics the study of how the meaning of languages is constructed and understood is referred to as *semiotics*. It divides into the disciplines of syntax, semantics and pragmatics [21]. These categories can be applied both to natural as well as artificial languages, e. g. for programming or modeling. In the context of artificial languages, *syntax* is determined by formal rules defining expressions of the language [14] and *semantics* determines the meaning of syntactic constructs [17]. "Linguistic *pragmatics* can, very roughly and rather broadly, be described as *the science of language use*" [16]. This also holds for MDE with its artificial languages, as discussed in the following. However, first we clarify some more terminology specific to MDE mainly taken from the modeling linguists Atkinson and Kühne [2].

The main artifacts in MDE are *models* with two main concepts: A model *represents* some software artifact or real world domain and *conforms* to a *metamodel*, defining its *abstract syntax*. Additionally, the *concrete syntax* is the concrete rendering of the abstract concepts. Concrete syntax can be textual or displayed in a structured way, for example a tree view extracted from an XML representation of the abstract syntax. To be comprehensible, also a graphical syntax is very often used, the Unified Modeling

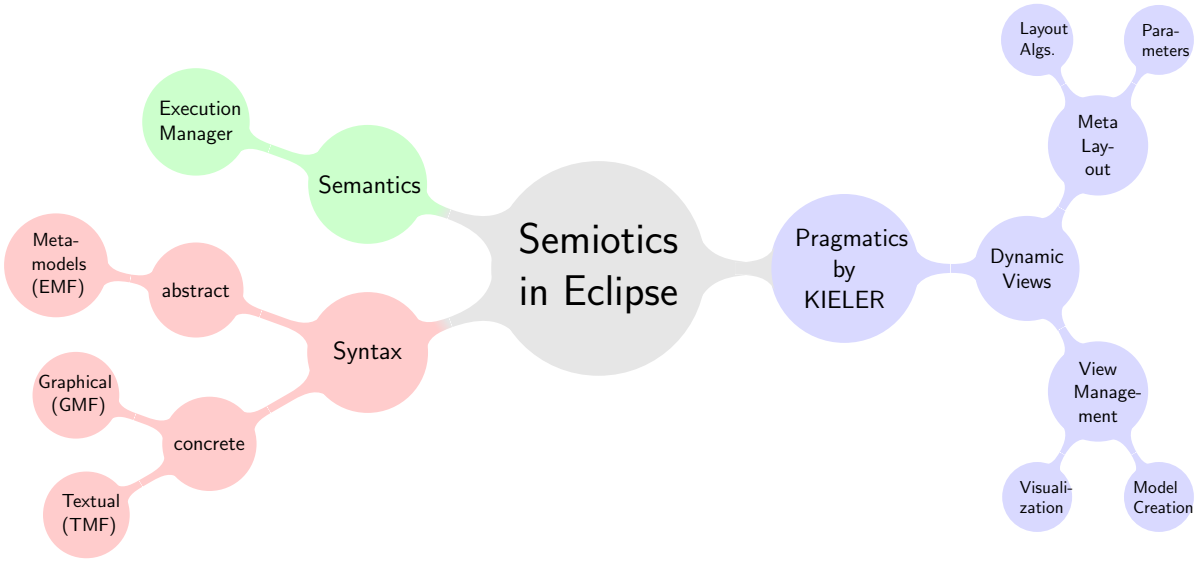


Figure 3.1: KIELER focuses on *pragmatics* and enhances the use of syntax and semantics of models which are defined by modeling platforms such as Eclipse

Language (UML) is one example.

A *graphical model* is a model that can have a graphical representation. A *view* onto the model is a concrete drawing of the model, sometimes also *diagram* or *notation model*. The abstract structure of the model leaving all graphical information behind is the *semantical* or *domain model*, or just *model* in short. Hence the *model* conforms to the abstract syntax, while the *view* conforms to the concrete syntax. A view can represent any subset of the model, which in some frameworks is used to break up complex models into multiple manageable views. Hence there is no fixed one-to-one relationship between model and view.

State-of-the-practice approaches still lack generic answers on how to specify *semantics* [22], but handle *syntax* of models very well, both abstract and concrete. They provide code generators to easily provide model implementations, syntax parsers and textual and graphical editors with common features like the Eclipse Graphical Modeling Framework (GMF)[1].

The third field of linguistics, *pragmatics*, traditionally refers to how elements of a language should be used, e. g., for what purposes a certain statement should be used, or under what circumstances a level of hierarchy should be introduced in a model. We slightly extend this traditional interpretation of pragmatics to all practical aspects of handling a model in its design process [11]. This includes practical design activities themselves such as editing and browsing of graphical models in order to construct, analyze and effectively communicate a model's meaning.

[1] `http://www.eclipse.org/modeling/gmf/`

4 Taming Complex Models

The main problem with pragmatics in state-of-the-practice modeling IDEs is the widely accepted way of user interaction with diagrams:
What-You-See-Is-What-You-Get (WYSIWYG) Drag-and-Drop (DND) editing. DND here encompasses all manual layouting activities that a modeler has to perform, such as positioning or setting sizes of graphical objects (nodes) or setting bend points of connections (edges). We do not distinguish whether such actions are real drag-and-drop operations with the mouse or are performed by keyboard.

When working with graphical models, it is useful to have an immediate graphical feedback on editing operations, hence WYSIWYG is not the problem. However, DND adds a lot of extra mechanical effort on editing diagrams. To quote a professional developer [25]: "I quite often spend an hour or two just moving boxes and wires around, with no change in functionality, to make it that much more comprehensible when I come back to it."

With such standard editing paradigm one often ends up with exactly one static view for a subset of a model where the developer once has decided the abstraction level—e. g. level of detail or subset of displayed nodes. To get a different view requires to start the editing process all-over.

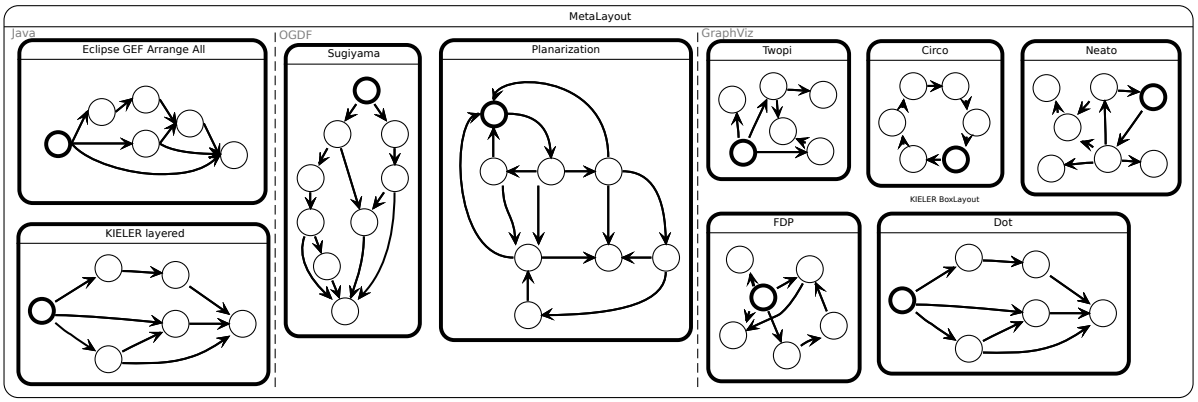


Figure 4.1: Meta layout in KIELER: Employ different layout algorithms in one diagram.

4.1 Meta Layout

The idea of *meta-layout* is to synthesize views automatically, thus freeing the user to focus on the model itself. As discussed further in Sec. 4.2, this not only saves time formerly spent on manual drawing activities, but yields completely new possibilities for user interaction. The meta-layout framework consists of two main parts: (1) A bridge between layout algorithm libraries and diagram editors and (2) parametrization possibilities to get the desired layout result of available algorithms, see also Fig. 4.1.

The layout bridge connects a range of layout algorithms with established graphical model diagram editors. Fig. 4.2 shows example layout/editor combinations. Fig. 4.2a shows the Eclipse Modeling Framework (EMF) Ecore tools class diagram editor with a Mixed-Upward-Planarization algorithm of the OGDF [15], which takes into account the different types of edges—inheritance vs. relations. Fig. 4.2b is a UML activity diagram of the upcoming GMF-Papyrus UML editor suite using the dot algorithm of the Graphviz library [12], which is well suited for compound graphs without inter-level edges. Fig. 4.2c shows a use case diagram of Papyrus, employing a force directed algorithm of the Graphviz library.

As illustrated in Fig. 4.3, the meta layout framework contains a basic graph data structure, the *KGraph* shown in Fig. 4.2a, for exchanging data between a concrete diagram editor and a layout algorithm. To achieve genericity, this does not assume any specific format of either of the two worlds. Glue code that translates between used data structures in both domains allows to use any diagram editor with any layout algorithm. The KGraph is used as an intermediate format to (1) formulate the layout problem and to (2) store the layout result, i. e. the concrete coordinates and sizes. The KGraph follows the ideas of GraphML[1] but is simplified to the needs in this context.

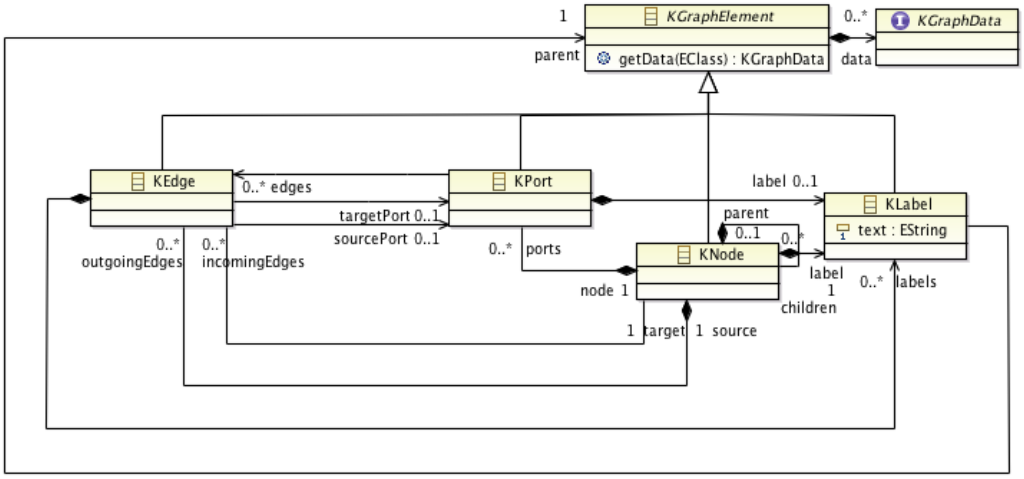
The layout process is executed as follows:

1. Use diagram glue code to read the model structure from its current view (e. g., but not necessarily, from a diagram editor). Create a KGraph from this structure.
2. Pass the KGraph to the layout algorithm. Use algorithm library specific glue code to transform the graph into the internal data structure of the library.
3. Call the automatic layout algorithm which creates a layout result containing coordinates and sizes in its internal data structures.
4. Attach the layout result from the algorithm back to the KGraph.
5. Apply the layout result from the KGraph to the diagram.

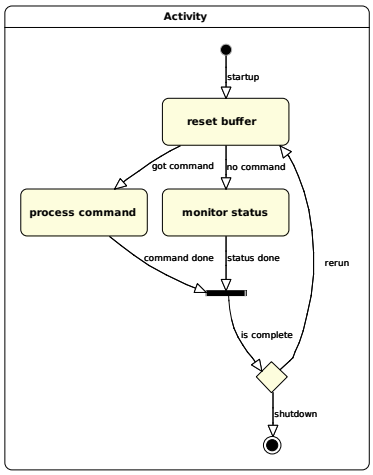
Meta layout not only bridges between diagrams and layouters, it also tries to do this in a smart customizable way.

There are certain requirements in diagram syntaxes as well as certain limitations of layout algorithms that it tries to mitigate:

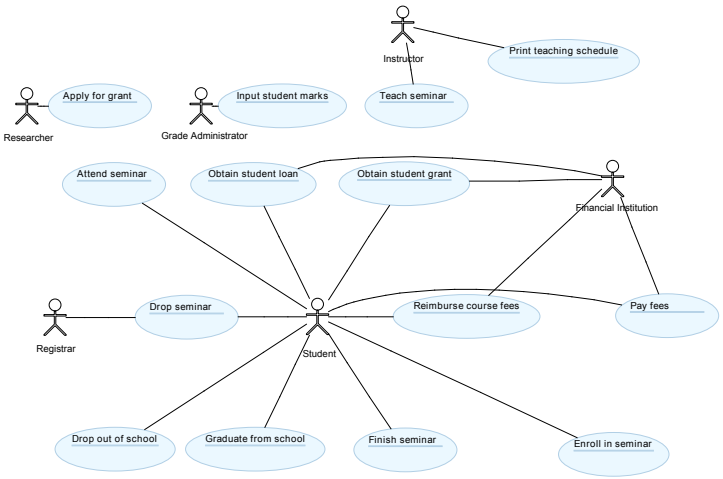
[1] `http://graphml.graphdrawing.org/`



(a) The KGraph as an EMF Ecore class diagram with mixed upward planarization [15].



(b) Papyrus UML Activity diagram / Graphviz dot layout [12].



(c) UML Use Case diagram of Papyrus / Graphviz neato layout [9].

Figure 4.2: Automatic layout examples for different editors with different layout algorithms.

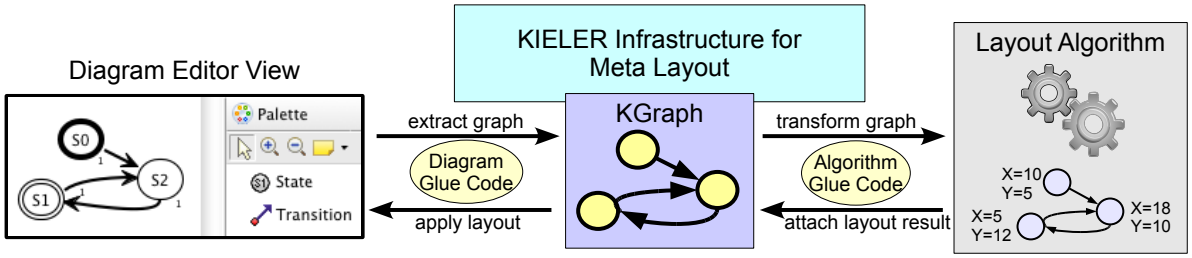


Figure 4.3: Overview of the KIML.

Listing 4.1: Recursive Layout

```
algorithm recursiveLayout(G: compound graph, v: vertex in G)
    for each child v_c of v: recursiveLayout(G, v_c)
    if v is not a leaf then
        retrieve the layout algorithm A associated with v
        set up A with the layout options associated with v and its children
        execute A on the children of v with the given configuration
```

Problem Diagrams may contain nodes, edges, multiple labels at all objects and optionally ports [31]. Algorithms might be limited to perform only layout for some of the elements, e. g. not support port constraints or labels.

Solution Specify requirements and limitations explicitly for diagrams and algorithms. So for a concrete algorithm interface there is some meta information added about what features the algorithm supports i. e. for what *kinds* of diagrams it is suited best. Vice versa, diagram editors can specify in meta information what kind of diagrams they provide. Parameters provided by algorithms can be made available, for example layout directions or seed values. Fig. 4.5a shows a KIELER screenshot and shows the layout options for one selected diagram region.

Problem Diagrams can be compound, i. e. nodes may contain other nodes. This is typically seen for example in UML State Machines or Packages. Many algorithms do not work on compound graphs.

Solution Apply layout recursively for compound graphs following Listing 4.1, starting with leaf nodes. This works for all layouters unless there are any hierarchy crossing edges. Such cases require special treatment [32].

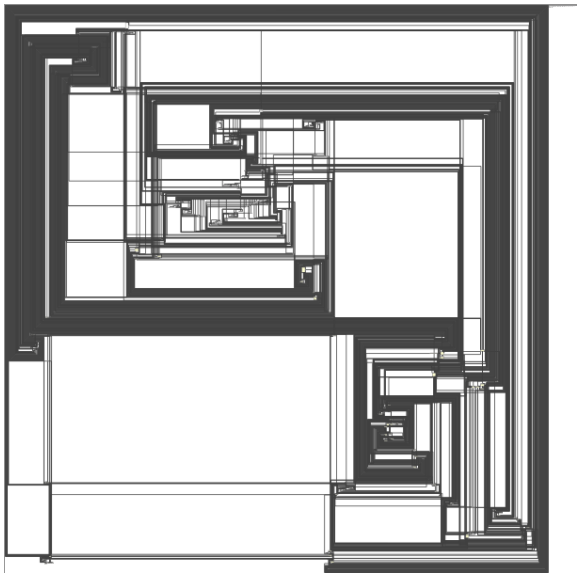
Problem One single layout algorithm might result bad layouts for complex models.

Solution Meta layout allows to use multiple different layout algorithms for different parts of one and the same view as shown in Fig. 4.1. This is well suited for compound models.

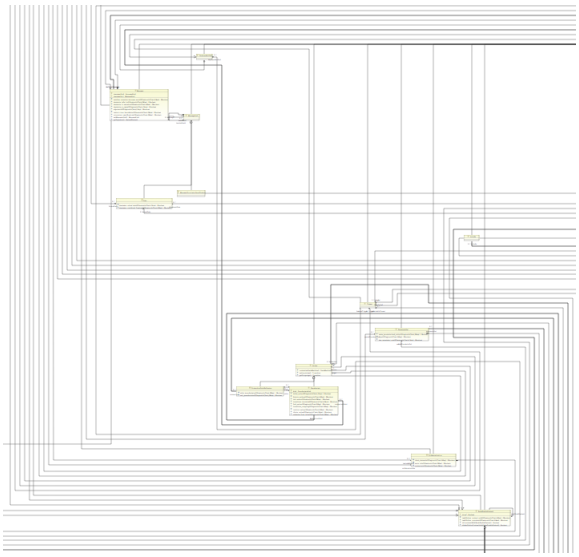
In summary, meta layout bundles a set of layout algorithms and matches them with concrete diagram syntaxes. It lets the user mix parameters and layouters to find the optimal layout result for custom model views.

However, there are limits of automatic layout of views when models become too complex. Consider for example the current UML2 Metamodel, which consists of 263 types with no compound structuring and thousands of relations and inheritances between the classes. This metamodel is available as an EMF Ecore model in the Eclipse Model Development Tools (MDT)[2], as semantical model augmented with some very small manually

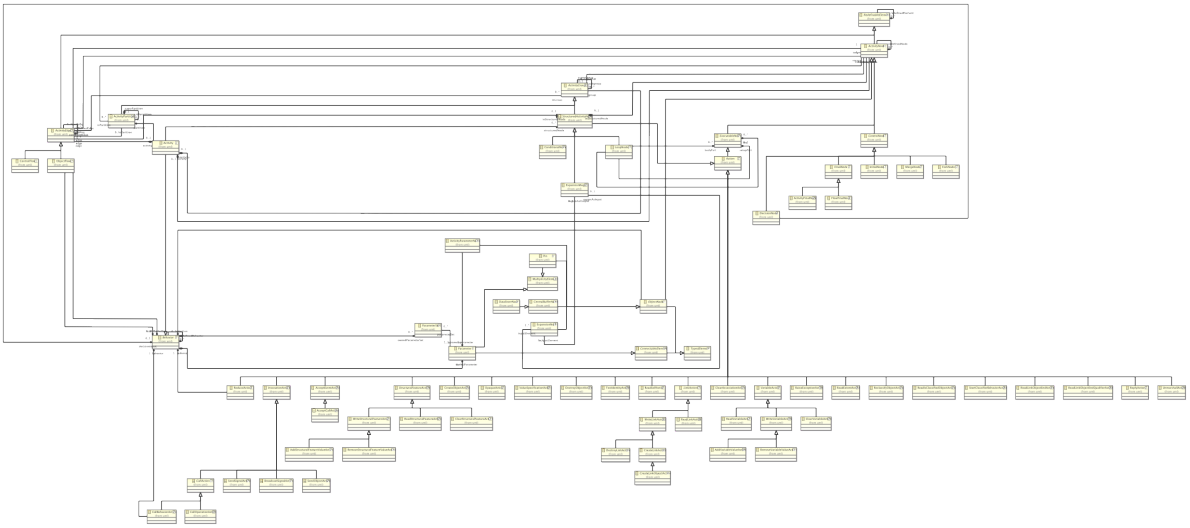
[2] `http://www.eclipse.org/modeling/mdt`



(a) Complete model.



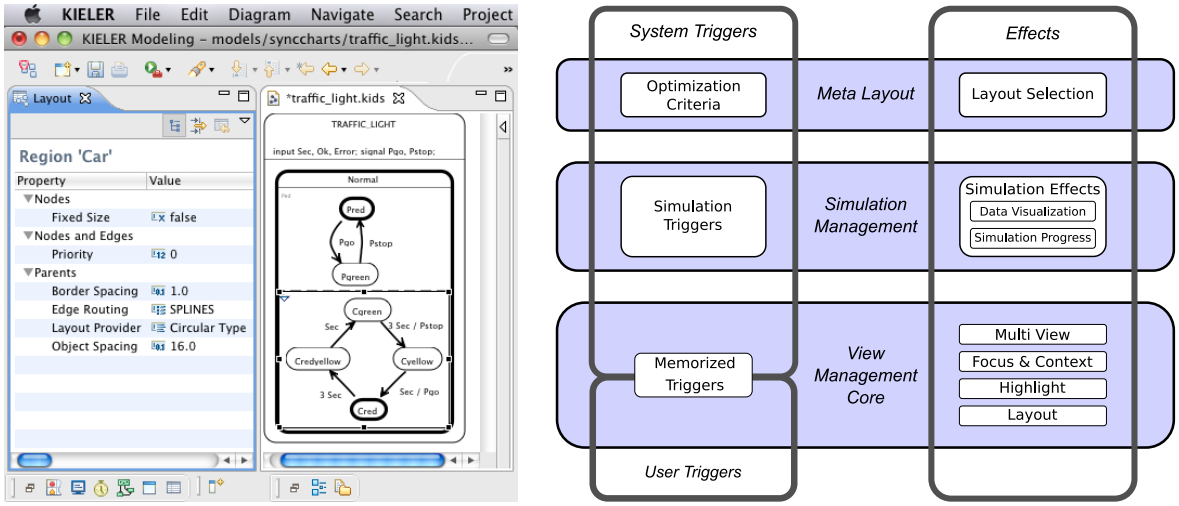
(b) Cutout with 20x zoom.



(c) A filtered view to the same model showing only Activity model parts.

Figure 4.4: Class diagram of the UML 2.1 metamodel in Eclipse. Standard navigation techniques come to their limits. Views become unusable. Filtering in view management can synthesize a feasible view.

layouted views, but due to the model's complexity, there is no complete view available [24]. With meta layout, it is possible to synthesize such a view; Fig. 4.4a shows the layout generated by KIML using the Mixed-Upward-Planarization algorithm [15], which is optimized for class diagrams and respects the different types of edges. However, the result looks more like a VLSI integrated-circuit die and is hardly usable. Especially the numerous relations make the diagram unreadable. The algorithm focuses on the avoidance of edge crossings which produces big and long "highways" of edges. Stan-



(a) KIELER specifying layout options.

(b) Aspects of view management [11].

Figure 4.5: Meta layout and view management.

dard navigation techniques like manual zooming and panning come to their limits; see also Fig. 4.4b. This limitation of plain layout application prompts the need for *view management*, discussed next.

4.2 View Management

When models and their corresponding views become too complex, it is time for abstraction. *View management* is inter alia a means to automate the choice of right abstraction levels. For a given model, view management chooses the subset of the model that should be presented in a view. It decides the *level of detail* [23] for all graphical elements and adds other graphical effects to views. This automatic synthesis of views is only possible due to the automatic layout service offered by meta layout. Hence meta layout can be regarded to provide model views as a service which view management uses. The idea of view management is to focus automatically to the parts of the model that are "currently interesting."

Obviously the context in which the user employs the model is important for this task. For example, to learn only about a smaller subset of the UML, e. g. Activity models, one may create a customized view on the UML metamodel that only contains elements immediately relevant to Activity models. Fig. 4.4c shows such a view that is again automatically synthesized with meta layout. However, this limited set of only 79 classes with much less edges presents a view that actually can be used very well to browse Activity models.

To make view management context sensitive requires a generic architecture that allows to define conditions under which certain views shall be synthesized. View management

listens to *triggers* or *events* under which certain graphical *effects* should be executed on the view. The orchestration of a set of triggers and effects forms a *view management scheme* (VMS), see also Fig. 4.5b. Triggers are categorized in user triggers—e. g. manual selection of elements—and system triggers—e. g. an event during a simulation run. Effects range from highlighting elements, configuring levels of details, filtering graphical objects to visualizing simulation data. An important effect uses the meta layout to rearrange the view that might have been changed by other effects like filters. See concrete applications in Chap. 5.

Triggers and effects are usually lower level implementations on the IDE platform where view management is performed. Hence most of them are intended to be provided by the platform developers, only a few by diagram editor providers and not by users. We provide a useful starting set where some are presented in the case studies below.

Combinations are higher level VMS abstractions that connect triggers with effects. Simply put they specify *under which conditions* (triggers) *which kind of view* (effects) shall be synthesized. The next chapter presents an implementation of view management and discusses two applications.

5 Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER)

The approaches presented in this paper are implemented and evaluated in the project KIELER, the Kiel Integrated Environment for Layout Eclipse Rich Client.[1] In the spirit of genericity, KIELER builds on the plug-in concept provided by Eclipse and especially its modeling projects.[2] As illustrated in Fig. 3.1, KIELER provides enhancements for pragmatics, to be combined with syntax and semantics defined by other projects.

5.1 Kieler Infrastructure for Meta Layout

The Kieler Infrastructure for Meta Layout (KIML) uses the Eclipse Modeling Framework (EMF) to specify abstract syntax. For concrete syntax KIML supports graphical editors generated with the Graphical Editing Framework (GEF), a framework to implement graphical DSL editors. The Graphical Modeling Framework (GMF) is a generative approach to GEF editors that has a standard persistence handling of models and their views (the *notation model* in GMF terminology). KIML provides a generic implementation of the diagram glue code (Fig. 4.3) for GEF/GMF that performs the following tasks:

- It extracts the graph structure from graphical GEF objects (so-called Edit Parts) into the KGraph.
- It provides a command to apply the layout results back to the diagram following the GEF request-command pattern. Therefore KIELER layout can be used with every GEF/GMF compatible editor. This is non-trivial, because GEF/GMF are not designed for automatic manipulation of views but only for single-element manual user interaction. For example changing bend points of an edge in one single step together with the position of a corresponding edge label requires some low level manipulation.
- It exploits the style mechanism of the notation model of GMF to make user defined layout options persistent in a generic way for all GMF editors without introducing new files.

Hence, for most GMF editors KIELER automatic layout can be used out-of-the-box. Optionally the Eclipse extension point *layoutInfo* is used to specify default values for

[1] `http://www.informatik.uni-kiel.de/rtsys/kieler`
[2] `http://www.eclipse.org/modeling/`

layout options, e. g. diagram types to setup default layout types. This has been done for example for the MDT/Papyrus UML suite [9]. Such meta information in form of an XML extension specification (plugin.xml) is the only thing a tool smith needs to provide to give the user a smooth user experience with KIML and his or her diagram editor. For other concrete syntax frameworks based on GEF, like the Generic Eclipse Modeling System (GEMS), Marama or Autofocus, the glue code would have to be extended accordingly.

For layout algorithm integration KIELER provides the extension point *layoutProvider*. There one specifies the layout options that the corresponding algorithm accepts and priorities for diagram types that it supports. The algorithm itself has to be implemented following a simple abstract class.

The main layout method simply has to follow the following signature:

public abstract void doLayout(KNode layoutNode, IKielerProgressMonitor monitor)

The given layoutNode forms the layout problem in shape of a KGraph. It is already augmented with the layout information of the current view, i. e. the layout options and the current coordinates and sizes. The task of the method is to exchange these values by new ones that yield the layout result of the algorithm. If not specified otherwise, the given graph is only a flat subset of one hierarchy level of the possibly compound view. It is then processed recursively following Listing 4.1. Usually the processing will take a significant amount of time. Hence with the monitor one can give some feedback to the user interface.

From this interface one can directly start to write a layouter in Java or add some glue code to bridge it to existing layout algorithm libraries. In order to give KIML some initial horse-powers, it provides some libraries or bridge code for them:

- Graphviz [12] is connected using a user installation of the tool, calling its process and piping Strings of dot graph syntax, which gets synthesized and parsed using Eclipse Textual Modeling Framework (TMF) Xtext. Graphviz provides a layer-based, multiple force-directed and some specialized algorithms, like the circo radial layouter.

- OGDF [7] is a C++ project with many recent and sophisticated algorithms. It gets connected using the Java native interface with the SWIG[3] wrapper generator.

- GMF provides an "arrange all" functionality, that provides a simple layer based [33] approach. This is also bridged in pure java to add the recursive functionality to this algorithm and to compare it to other libraries.

- Also provided is the Kieler Layout of Dataflow Diagrams (KLoDD), a customized hierarchical layout algorithm that supports hyperedges and especially port constraints as a pure java implementation [31].

[3] `http://www.swig.org/`

5.2 Applications for View Management

As an example, the following illustrates how view management in KIELER augments the editing and simulation of *SyncCharts* [1].

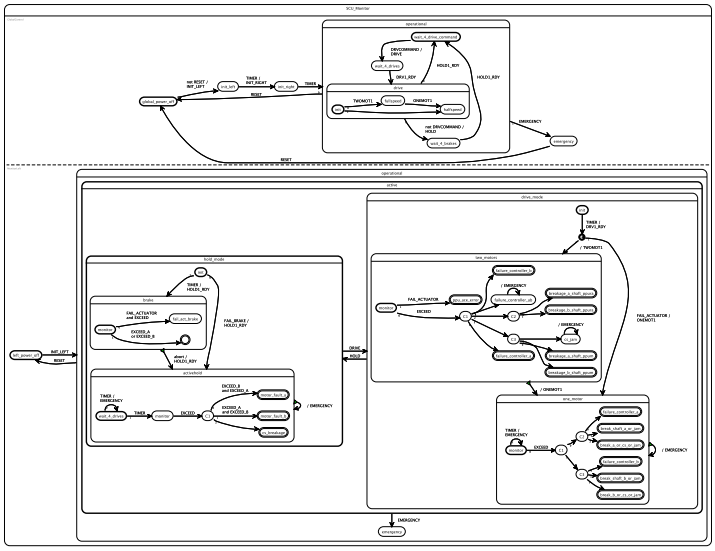
5.2.1 Simulation with Focus & Context

One means to learn about the behavior of a SyncChart is to execute it stepwise while the simulation browser highlights active states. This paradigm is used by most state machine based tools like Matlab/Simulink/Stateflow of The Mathworks, Rhapsody of Telelogic/IBM, SCADE of Esterel Technologies or Ptolemy II of UC Berkeley. The usual means for navigation are panning, zooming and opening different parts of the model in different windows/canvases. However, for complex models it becomes difficult and effort prone to manually navigate through a model. Debugging the application with one view gets difficult because it is hard to follow the advancements of state transitions when either (1) looking at the whole chart as an overview loosing details or (2) zooming into specific parts of the diagram loosing the context of the cutout. Figs. 5.1a/b demonstrate this with an avionics application [10].

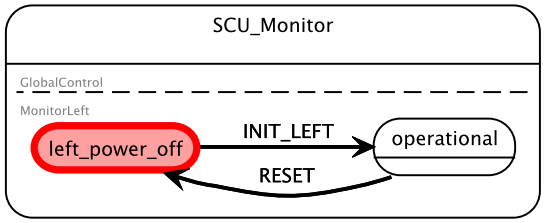
To alleviate this problem, the view management service can synthesize a new view on the model dynamically. The idea is to use *focus and context* methods to present only the "interesting" parts of the model [26]. For SyncCharts, a natural definition of "interesting" considers the currently *active* states, as illustrated in Figs. 5.1c/d. In KIELER, a specific trigger for the simulation notifies the view management about changes in state activity. A simple effect then highlights active states. An additional effect changes the level of detail at which the model objects get displayed in the view. In KIELER this is implemented by using GEF's methods to collapse or expand compartments, which comprise the contents of states and parallel regions. Afterwards view management uses KIML to rearrange all elements and zooms-to-fit to make best use of the given space. This unfolds the potentials of focus & context, as it presents all required details in the *focus* while still showing the direct neighbor inactive states collapsed with reduced detail level as the *context*. An animated morphing between the different views is provided to match the mental map of the user [28]. For an impression of this, the reader is referred to example videos on-line (or the KIELER tool itself).

Focus and context is applicable for many use cases in an MDE design process. However, only by combining it with automatic layout it unfolds its potentials. It can be used to reduce manual navigation efforts in many contexts by defining custom view management triggers that indicate the focus objects in the view.

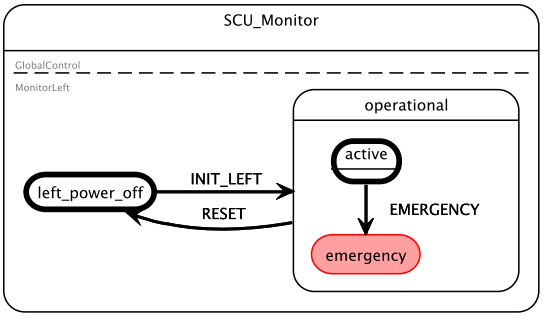
Currently these view management schemes have to be implemented in the low level programming language. However, we are working on ways to define them more abstract to allow possibly even users to define custom focus and context specifications for their own languages and usages.



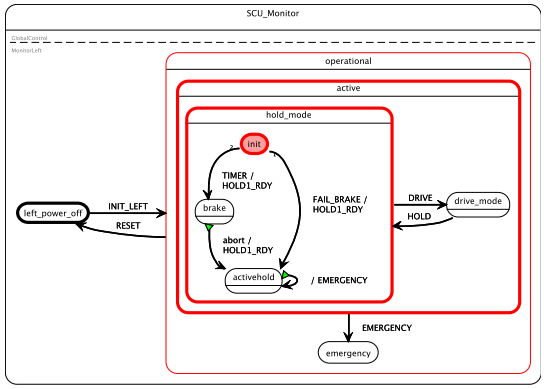
(a) The whole SyncCharts model.



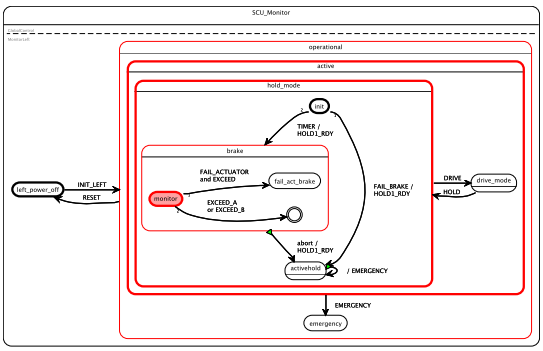
(b) Focus & Context (1): Starting simulation collapses all inactive states and manually collapsed regions.



(c) Focus & Context (2): Advancing simulation will always expand only active states with their full hierarchy.



(d) Focus & Context (3): Active states are the *focus* and shown with full detail while inactive states are the context and their *contents* get collapsed.



(e) Focus & Context (4): Even in deep hierarchy usually the full complexity of the model is hidden.

Figure 5.1: Focus & Context in a SyncChart

5.2.2 Structure-Based Editing

Another task in an MDE design process is to create or modify models. One approach to harness view management is to go back to textual editing. A textual editing framework like Xtext[4] can be enriched with graphical views synthesized on-the-fly.

An alternative approach that stays in the graphical domain and keeps the direct visual feedback like WYSIWYG is *structure-based editing*. It employs model-to-model (M2M) transformations on the semantic model—its structure. It is an interactive approach where the user can work on the model view. The workflow for editing a model reduces to the following steps:

1. Focus a graphical model object for modification.
2. Apply an editing transformation operation.

View management with KIML applies the transformation, creates new graphical elements, and rearranges the resulting view.

The general implementation scope is shown in Fig. 5.2.2. Again, to be generic, it allows any M2M transformation framework to be used with KIELER Structure-Based Editing (KSBasE). The corresponding transformation engine has to be wrapped in a TransformationFactory. To integrate with the user interface, KIELER connects to the Eclipse TMF Xtend transformation system and all graphical GMF editors.

In the KSBasE implementation, the set of pre-defined editing transformations is offered in the user interface in the main menu, toolbar, context menu, GMF's popup balloon menus and keyboard shortcuts. Adding Xtend transformations and configuration of menu contributions for GMF editors is done through an Eclipse extension point.

[4] `http://www.eclipse.org/Xtext/`

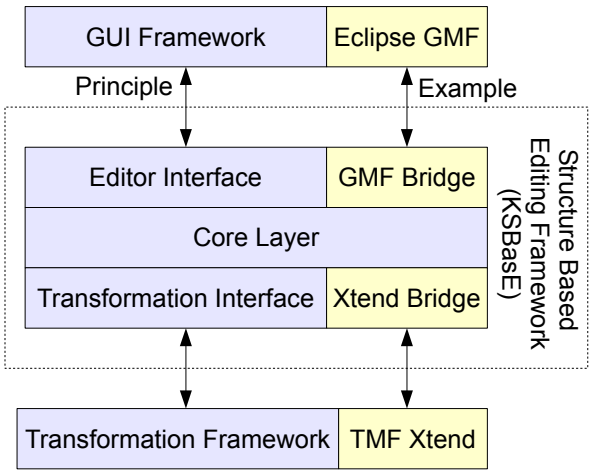
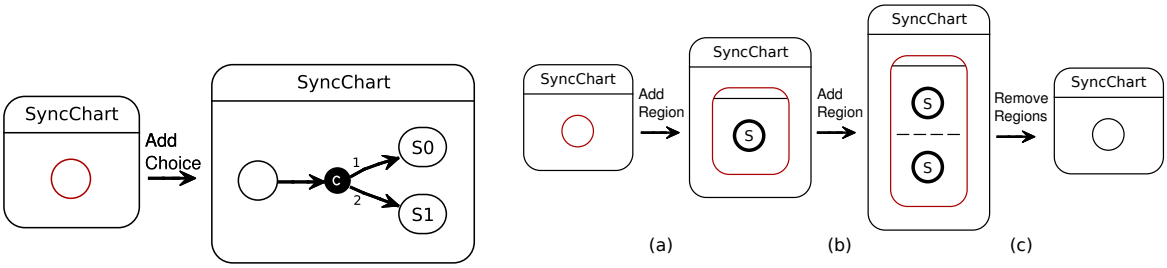


Figure 5.2: Scope of KSBasE



(a) Adding a choice construct.

(b) Adding a region to simple and a complex state, followed by removal of all regions.

Figure 5.3: Example transformations for SyncCharts

KIELER provides a complete set of example transformations to edit SyncCharts. This is complete in the sense of fully covering the syntax of SyncCharts as well as giving some helpful "transformational sugar" to speedup the editing process. Some example transformations are depicted in Fig. 5.3.

Key tasks of KSBasE are the following:

- Extract the current possibly heterogeneous selection of elements in the view and pass their semantic objects as parameters in the right order to the chosen transformation function.
- Offer only those transformations in the user interface that are applicable for the current selection.
- Interact with view management to synthesize a new view after a transformation.

6 Evaluation

To assess the benefits of view management for model editing, we have conducted a study using KIELER. The hypothesis to be evaluated was that structure-based editing reduces the development times for creation and modification of graphical models significantly compared to usual WYSIWYG Drag-and-Drop (DND) editing. The 30 subjects divided into three different categories: The *class* group was familiar with the syntax of SyncCharts but not with modeling editors. The *practical* group took part in a practical course and had some experience already with Eclipse GMF editors. The last group comprised developers of the *KIELER team*, combining experiences with SyncCharts and the Eclipse SyncCharts editor.

The task was to create three different SyncCharts, using a different input method in random order for each: (1) standard Drag-and-Drop editing, (2) DND editing with manually triggered automatic layout and (3) structure-based editing as presented above. The models were provided in a comprehensible but formal textual notation. The experiment and its outcome are described in detail elsewhere [20], but Fig. 6 summarizes the results.

Editing with automatic layout decreased the necessary modeling times in average by nearly 33%. Full KSBasE reduced the times by another 15% compared to DND. From auto-layout to KSBasE the difference was mainly influenced by the earlier experience, e. g. how well keyboard shortcuts could be employed.

The SyncCharts in the tasks were of rather simple structure, and only creation was required, no modifications. Hence only rather plain transformations in KSBasE were

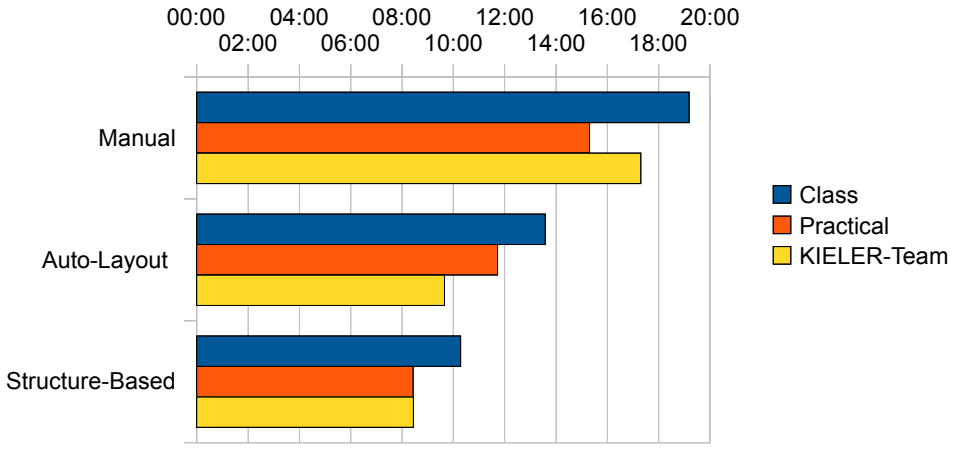


Figure 6.1: Evaluation of different editing methods.

necessary to complete the tasks. More complex transformations might result in even greater speedups.

7 Conclusions and Outlook

Visual models help understanding complex systems. However, with the user interaction paradigms established today, activities such as creating or browsing visual models can be very tedious. We presented an approach on enhancing the *pragmatics* of model-based design—the way a user interacts with models. The concept of *meta layout* enables the dynamic synthesis of different diagrammatic views on graphical models. *View management* builds upon automatic layout to configure views on models given a certain context in which the model is examined. Two applications—simulation visualization with focus & context and structure-based editing—illustrated view management in KIELER. An experimental evaluation supports the claim that view management with auto-layout helps to tame complexity in graphical modeling.

Concrete further modeling languages under investigation are the UML and actor-oriented dataflow languages. This involves finding complete sets of transformation functions as well as ways to visually debug those languages.

Ongoing work is integration and development of more layout algorithms to support more specialized graphical editor syntaxes and to increase the aesthetics of layout results in general.

Another current goal is the adoption of a view management language, as augmentation to a modeling language, to allow to formulate view management use cases and to establish view management as a "first-class citizen" in modeling.

Bibliography

[1] Charles André. SyncCharts: A visual representation of reactive behaviors. Technical Report RR 95–52, rev. RR 96–56, I3S, Sophia-Antipolis, France, Rev. April 1996. `http://www.i3s.unice.fr/~andre/CAPublis/SYNCCHARTS/SyncCharts.pdf`.

[2] Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Software*, pages 36–41, 2003.

[3] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[4] Enrico Biermann, Karsten Ehrig, Christian Khler, Gnther Kuhns, Gabriele Taentzer, and Eduard Weiss. Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In *Model Driven Engineering Languages and Systems (MoDELS'06), Lecture Notes in Computer Science*, volume 4199/2006, pages 425–439. Springer Berlin/Heidelberg, 2006.

[5] Petra Brosch, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. An example is worth a thousand words: Composite operation modeling by-example. In *Model Driven Engineering Languages and Systems, (MoDELS'09)*, volume 5795 of *Lecture Notes in Computer Science*, pages 271–285. Springer Berlin / Heidelberg, 2009.

[6] Stuart K. Card, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, January 1999.

[7] Markus Chimani and Carsten Gutwenger. Algorithms for the hypergraph and the minor crossing number problems. In *ISAAC 2007: Proceedings of the 18th International Symposium on Algorithms and Computation*, volume 4835 of *LNCS*, pages 184–195. Springer-Verlag, 2007.

[8] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behavior and Evolution of Software*. Springer, 2007.

[9] Hauke Fuhrmann, Miro Spönemann, Michael Matzen, and Reinhard von Hanxleden. Automatic layout and structure-based editing of UML diagrams. In *Proceedings of the 1st Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2010)*, Dresden, March 2010.

[10] Hauke Fuhrmann and Reinhard von Hanxleden. Enhancing graphical model-based system design—an avionics case study. In *Conjoint workshop of the European Research Consortium for Informatics and Mathematics (ERCIM) and Dependable Embedded Components and Systems (DECOS) at SAFECOMP'09*, Hamburg, Germany, September 2009.

[11] Hauke Fuhrmann and Reinhard von Hanxleden. On the pragmatics of model-based design. In *Foundations of Computer Software. Future Trends and Techniques for Development—15th Monterey Workshop 2008, Budapest, Hungary, September 24–26, 2008, Revised Selected Papers*, volume 6028 of *LNCS*, 2010.

[12] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software—Practice and Experience*, 30(11):1203–1234, 2000.

[13] Jan Gulliksen, Bengt Göransson, Inger Boivie, Jenny Persson, Stefan Blomkvist, and Åsa Cajander. Key principles for user-centred systems design. In *Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle*, volume 8 of *Human-Computer Interaction Series*, pages 17–36. Springer Netherlands, 2005.

[14] Corin A. Gurr. Effective diagrammatic communication: Syntactic, semantic and pragmatic issues. *Journal of Visual Languages and Computing*, 10(4):317–342, 1999.

[15] Carsten Gutwenger, Michael Jünger, Karsten Klein, Joachim Kupke, Sebastian Leipert, and Petra Mutzel. A new approach for visualizing UML class diagrams. In *SoftVis '03: Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 179–188, New York, NY, USA, 2003. ACM.

[16] Hartmut Haberland and Jacob L. Mey. Editorial: Linguistics and pragmatics. *Journal of Pragmatics*, 1:1–12, 1977.

[17] David Harel and Bernhard Rumpe. Meaningful modelling: What's the semantics of "semantics"? *IEEE Computer*, 37(10):64–72, 2004.

[18] Michael Jünger and Petra Mutzel. *Graph Drawing Software*. Springer, October 2003.

[19] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[20] Michael Matzen. A generic framework for structure-based editing of graphical models in Eclipse. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, March 2010. `http://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/mim-dt.pdf`.

[21] Charles William Morris. *Foundations of the theory of signs*, volume 1 of *International encyclopedia of unified science*. The University of Chicago Press, Chicago, 1938.

[22] Christian Motika, Hauke Fuhrmann, and Reinhard von Hanxleden. Semantics and execution of domain specific models. Technical Report 0923, Christian-Albrechts-Universität Kiel, Department of Computer Science, December 2009.

[23] Benjamin Musial and Timothy Jacobs. Application of focus + context to UML. In *APVis '03: Proceedings of the Asia-Pacific symposium on Information visualisation*, pages 75–80, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[24] Object Management Group. Unified Modeling Language: Superstructure, version 2.0, Aug 2005. `http://www.omg.org/docs/formal/05-07-04.pdf`.

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

[26] Steffen Prochnow and Reinhard von Hanxleden. Statechart development beyond WYSIWYG. In *Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS'07)*, Nashville, TN, USA, October 2007.

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

[28] Jean-Michel Boucheix Richard Lowe. Learning from animated diagrams: How are mental models built? In *Diagrammatic Representation and Inference: 5th International Conference, Diagrams 2008*, page 266, Herrsching, Germany, September 2008. Springer-Verlag New York Inc.

[29] Georg Sander and Adrian Vasiliu. The ILOG JViews graph layout module. In *GD 2001: Proceedings of the 9th International Symposium on Graph Drawing*, volume 2265 of *LNCS*, pages 469–475. Springer-Verlag, 2002.

[30] Ahmed Seffah, Jan Gulliksen, and Michel C. Desmarais. An introduction to human-centered software engineering. In *Human-Centered Software Engineering—Integrating Usability in the Software Development Lifecycle*, volume 8 of *Human-Computer Interaction Series*, pages 3–14. Springer Netherlands, 2005.

[31] Miro Spönemann, Hauke Fuhrmann, Reinhard von Hanxleden, and Petra Mutzel. Port constraints in hierarchical layout of data flow diagrams. In *Proceedings of the 17th International Symposium on Graph Drawing (GD'09)*, volume 5849 of *LNCS*, pages 135–146. Springer, 2010.

[32] Kozo Sugiyama and Kazuo Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):876–892, Jul/Aug 1991.

[33] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.