Efficient Exploration of Complex Data Flow Models[*]

Patrick Frey,[1] Reinhard von Hanxleden,[2] Christoph Krüger,[2]
Ulf Rüegg,[2] Christian Schneider,[2] and Miro Spönemann[2]

[1] ETAS GmbH, Stuttgart, Germany
`patrick.frey@etas.com`

[2] Dept. of Computer Science, Christian-Albrechts-Universität zu Kiel, Germany
`{rvh,ckru,uru,chsch,msp}@informatik.uni-kiel.de`

Abstract: The modeling tools that are commonly used for embedded software development are rather limited when it comes to communicating certain model properties between different groups of engineers. For example, calibration engineers need to understand dependencies between signals and calibration parameters, while function developers create models with a divide-and-conquer strategy, where details of signal dependencies are hidden by abstract interfaces.

We state requirements for modeling tools to improve the exploring of complex data flow models and to facilitate the understanding of engineers from different domains. We propose an approach that combines *transient views* and *automatic layout* and present two implementations based on different technologies, GMF and KLighD. While both technologies fulfill all requirements, KLighD turned out to be superior in terms of both performance and programming effort. The implementations are based on an open-source framework and are employed in a commercial product that targets the calibration process for automotive software development.

1 Introduction

In many application domains, such as the automotive industry, model-driven software development (MDSD) has become the established approach for the design and specification of system features, as well as their implementation in form of software executed by embedded computer systems. MDSD offers advantages such as separation of specification and implementation, reuse of function specifications across different development phases from simulation over prototyping to target integration, and automatic generation of safe code for different target microcontroller platforms. Commercial tools such as ASCET from ETAS GmbH, Simulink from The MathWorks, Inc., and the research framework Ptolemy from UC Berkeley, offer similar means to model functions graphically based on block diagrams for data flow oriented functions, or statecharts for control flow oriented functions. In such tools, complex functions can be divided into manageable pieces such that the problem of graphically specifying the functions is mastered. This results in nested graphical models

[*]This work was also funded in part by the Program for the Future Economy of Schleswig-Holstein and the European Regional Development Fund (ERDF)

consisting of several hierarchies of elements, each represented by a diagram. A complex embedded system, an engine control system of an automotive vehicle for example, can contain several hundreds or even thousands of individual diagrams.

While the graphical modeling approaches of MDSD are well suited to divide and conquer complex functions into manageable parts, they do not address the need of engineers to get a seamless understanding of the overall functionality at the system level. This, however, is especially important after a function has been designed by one engineer and needs to be understood by other engineers.

Control applications such as anti-lock braking systems or engine control systems often need to be fine-tuned to match a desired behavior or to optimally control a physical process. For this purpose, calibration engineers need to get an in-depth understanding of how the functions in the electronic control system work. Since many functions are developed by means of MDSD approaches, the graphical models are an important source of information to get such an understanding. Often, the engineers do not have direct access to the models and the tools themselves, but are only provided with a textual documentation with a fixed page size, suited for printouts, where screenshots of the model hierarchies are depicted. It is not untypical for calibration engineers, who are highly-paid application experts, to have to work with documents that exceed 5000 pages, where the cross-navigation index alone may consume about a third of the pages. Needless to say, retrieving specific information and assembling a complete picture of the application from such serialized, static documents is thus a very tedious and time-consuming exercise.

Contributions This article presents an approach for exploring and browsing fragmented complex data flow models that may come from several sources. The work presented here has been driven by concrete demands for the calibration of electronic control units, but we expect the results to be applicable to other areas, facing similar challenges, as well. We state requirements for tooling support and propose a number of methods to fulfill these requirements, specifically 1) a *transient views approach*, where the information that is relevant for model exploration is extracted from the source models and transformed on-the-fly into a generic light-weight format for presentation, 2) systematic use of *automatic layout* for drawing the diagrams, and 3) an exemplary view modification increasing the benefit of our model browser and illustrating some opportunities of the transient views approach.

We present two exemplary implementations of these concepts, and compare and evaluate them in terms of tool responsiveness and implementation effort. The implementations are part of the EHANDBOOK solution (ETAS), which provides interactive documentation facilities with an integrated model viewer, and of the KIELER open source project.[1]

Outline The rest of this paper is organized as follows. We discuss related work in the remainder of this section and collect requirements for proper tooling support in Sec. 2. The basic concepts are presented in Sec. 3, the corresponding implementations in Sec. 4. Comparisons and evaluations are discussed in Sec. 5. Finally, we summarize in Sec. 6.

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

Related Work

UC Berkeley's Ptolemy project is an example of a modeling tool that allows heterogeneous compositions of model parts [EJL$^+$03], which is what we also want to do here. Each part can define locally how its content shall be executed using a *model of computation*. The composition of parts is done according to the *actor-oriented design* paradigm [LNW03], where *actors* communicate via ports. Ptolemy uses a simple and extensible meta model [BLL$^+$08] defining the models' abstract syntax that is implemented in Java. The Ptolemy framework focuses on semantic aspects of heterogeneous models. Thus, each actor comes with all information necessary for model simulation, and the models are treated as monolithic artifacts. Here, in contrast, we concentrate on the exploration and browsing of large-scale models by abstracting them into light-weight structures, which can be inspected more efficiently.

Considerable effort has been spent on simplifying the development of modeling tools for customized or domain-specific modeling environments. Corresponding development environments include meta modeling facilities for creating the basic data structures as well as support for determining the representations of those structures in diagrams. Examples of such tools are Marama [GHL$^+$13], DiaMeta [Min06], GME [LMB$^+$01], VMTS [MLC06], GMF Tooling,[2] and MetaEdit+.[3] Those frameworks, however, focus on the creation of models rather than browsing existing models most comfortably. In our scenario existing complex models from different languages shall be explored by users. This requires a high quality tool in terms of responsiveness as well as accurate rendering. Editing assistance such as undo and redo operations, however, is not required.

The work of Storey et al. [SWFM97] employs automatic diagram synthesis for program comprehension and architecture recovery of given code rather than representing specification data in a reader-friendly form. In a follow-up work Bull et al. [BSLF06] developed the *Zest*[4] framework enabling visualizations of flat graph structures in Eclipse. Its aim is to provide a graph widget that seamlessly integrates into the existing widget zoo. This framework, however, supports neither ports nor nested graph representations.

Regarding the visualization of hierarchical models, an approach that follows the *fisheye view* concept [SB92] was introduced by Schaffer et al. [SZG$^+$96]: the content of hierarchical nodes is displayed directly inside their bounding box. The *fisheye zoom* technique allows dynamic collapsing or expanding of composite nodes in order to hide or reveal their content. This leads to the concept of *focus & context*, where the details of the currently viewed component are directly embedded in the context the component is used in. Earlier focus & context implementations employed algorithms for modifying the previous layout in order to eliminate node overlaps [RMG07, SFM99], which is especially suited for changing the layout as little as possible, thus helping the user to preserve his or her *mental map* of the model. However, it is yet unclear how such layout modifications can be done under consideration of port constraints. Here, we combine a focus & context visualization with graph layout methods enhanced by orthogonal routing and port constraint support.

[2] `http://www.eclipse.org/modeling/gmp/?project=gmf-tooling`
[3] `http://www.metacase.com/mep/`
[4] `http://www.eclipse.org/gef/zest/`

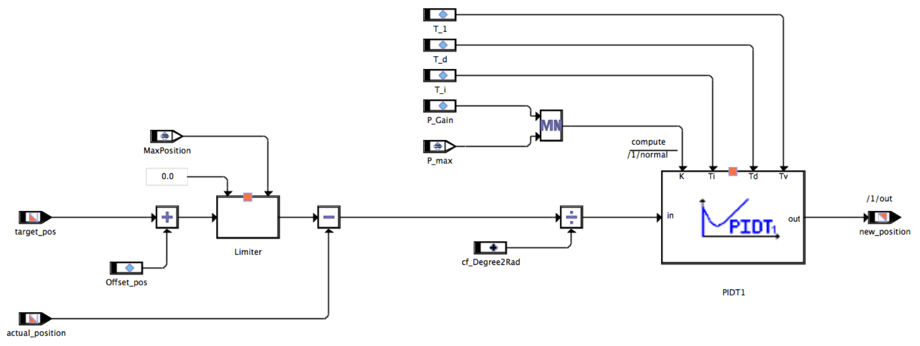


Figure 1: An ASCET model with original, manually drawn layout.

2 Exploring Complex Models – Requirements

In the following, we discuss requirements imposed on modeling tools that we found necessary to improve the experience of navigating complex models. We focus on the process of presenting and browsing existing models, which may be fragmented, i. e. spread over multiple files, and neglect any functionality to create new models or alter existing ones.

The *actor* models of our driving application, such as shown in Fig. 1, consist of other actors that are connected by edges via ports (denoted by little arrows). To assess the size of such diagrams, Klauske [Kla12] analyzed 12 Simulink models from automotive applications and measured an average size of 3333 nodes and 4274 edges per model. However, each hierarchy level (the direct content of a composite actor) is usually rather moderate in size. In Klauske's measurements each level contains 22 nodes and 29 edges on average.

A very basic requirement is to draw the elements of which diagrams are composed in the same way as in their original modeling tools. The symbols used to draw these elements often convey important semantic properties, e. g. about the type of a node. The mathematical operators for addition, subtraction, division, and minimum are identified easily in Fig. 1 due to their intuitive graphical representation. Without this representation, the rectangular node figures would all seem like black boxes.

Model Harmonization Large, possibly fragmented models shall be presented in a seamless fashion. For this purpose, several requirements can be stated.

H1 The impression of fragmentation shall be eliminated; hierarchy and fragment boundaries have to be spanned without breaking the natural flow of navigation.

H2 Likewise, no additional tool windows are to be opened when showing further details of the model.

H3 Existing relationships between the fragments, e. g. wires that cross a hierarchy level, shall be connected and be visible within the view.

H4 Multiple different modeling languages shall be combinable within one diagram, e. g. data flow notions as well as statechart notions.

Automatic Layout The automatic generation of graph-based views requires the created elements and shapes to be positioned in the available view area. We discern between the *micro* layout, affecting the composition of figures used to draw each single element, i. e. a node, edge, port, or label, and the *macro* layout, affecting the placement of these elements on the canvas [SSvH12]. The requirements on these two levels of diagram layout are very different: for micro layout we need a flexible mechanism for relative placement and size determination, while for macro layout we rely on *aesthetic criteria* for graph drawing, which have been well studied [BRSG07].

The most important macro layout criteria imposed in the context of actor diagrams as considered here are the following.

L1 Edges shall point from left to right, except feedback edges, which may point to the opposite direction.

L2 Edges are connected to specific ports on their source and target nodes. Usually these ports cannot be moved arbitrarily, but are subject to different kinds of positioning constraints (see below).

L3 Each output port may be connected to multiple input ports, effectively forming a directed hyperedge.

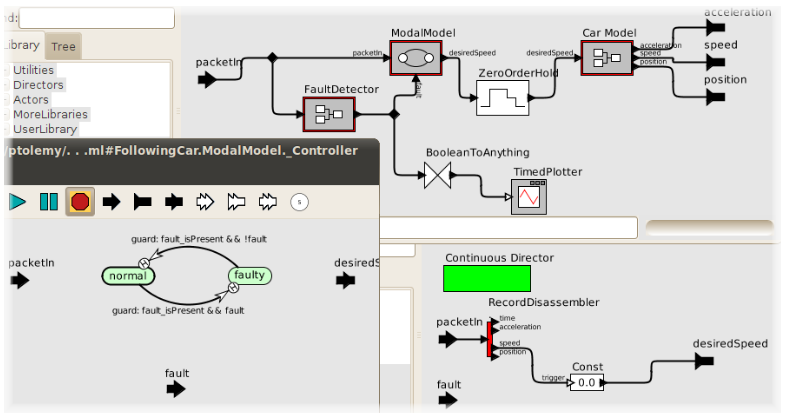
L4 Edges shall be routed orthogonally, i. e. only using horizontal or vertical line segments, with as few crossings and bends as possible.

L5 The drawing shall be compact, i. e. it shall have a small area and good aspect ratio (near that of a computer screen).

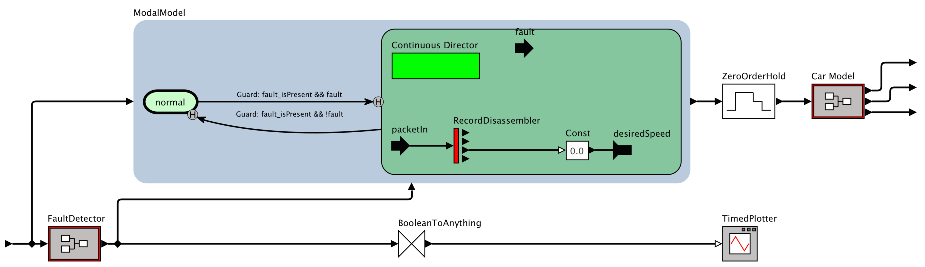
L6 If applicable, the layout shall be as close as possible to that seen in the original modeling tool in which the diagram was created, which we call the *original* layout.

Ports are placed on the border of their respective node, but their exact positioning is subject to different constraints that depend on the specific application (Criterion L2). We consider different constraint levels that determine how much the automatic layout process is allowed to modify port positions [KSSvH12]: with FREE constraints, ports can be freely placed, while FIXEDSIDE assigns a specific node side to each port. With FIXEDPOS constraints, port positions must not be modified by the layout process at all.

Criterion L6 is particularly relevant when users are already familiar with an existing diagram from the original tool. Retaining the original layout would help users to recognize the model at first glance, without requiring them to adjust their mental map of the model. Several metrics have been proposed to measure the closeness of two layouts [BT00]. However, an aspect that is not covered by these abstract metrics is to respect domain-specific constraints, e. g. placing inputs of the model to the left and outputs to the right.



(a) Ptolemy's original editor *Vergil*. The content of each hierarchical node is displayed in a new tool window, thus the user can easily lose the context he is working in.



(b) KIELER's Ptolemy viewer. Hierarchy is embedded directly into the nodes, and multiple visual representations are possible within the same diagram.

Figure 2: Snippet from Ptolemy's CarTracking model. Three hierarchy levels are visible, of which the outermost level (Following Car actor) contains data flow. One of its actors (ModalModel) contains a statechart, of which a state (faulty) is refined by a data flow model.

3 Towards Transient Views of Actor Models

Transient Views We apply the *transient views approach* to synthesize the graphical representations of semantic models automatically [SSvH13]. This approach is about the *on-demand* creation of diagrams without storing any intermediate data persistently. Thereby, no specific relationship between objects in the application model and elements in the diagram is prescribed. This way implicit model information can be made explicit, and fragmented information can be aggregated in order to present them to users most conveniently (Criterion H1). Concrete diagrams are created by composing *view models* that are then handed over to a rendering tool. They are automatically arranged, and heavy-weight editing facilities are omitted in favor of responsiveness of the tool. The approach is optimized for user interactivity like changing the depicted amount of detail, e. g. by expanding or collapsing nodes.

The fact that the view models denoting the diagram are completely separated from the source models paves the way for composing diagrams from different hierarchy levels of a model or even different modeling languages in the same view, fulfilling Criterion H4. Thus, the so created diagrams are not restricted to actor-based models, but can also visualize state machines, process models, or component composition specifications. This way model visualizations meeting all the harmonization requirements stated in Sec. 2 can be realized. As illustrated by Fig. 2, the combined visualization of multiple hierarchy levels can help the user to set the focus without losing the corresponding context of the overall model. Furthermore, view models need not to be created in one run, but may be built up incrementally. For example, nested diagram elements can be attached lazily when their container element is expanded. View models may also be updated continuously, e. g. for displaying feedback data while performing simulations or in-system-tests.

In spite of the separation of application models and view models, transient view mappings allow to associate diagram elements to the model elements they are derived from. By means of such associations, queries can be performed on model elements that are chosen via their representatives in the diagram, and the results can be visualized in the diagram for easiest understanding by the user.

Automatic Layout Automatic macro layout can be realized using graph layout methods [DETT99]. Some of the macro layout criteria listed in Sec. 2 have been thoroughly studied in graph drawing research. The main method for obtaining a left-to-right layout as stated in Criterion L1 is the *layer-based* (a. k. a. *hierarchical*) approach, which was proposed by Sugiyama et al. [STT81]. Regarding Criterion L3, Sander proposed an extension of the layer-based approach for routing orthogonal hyperedges [San04]. More recently, further extensions have been published to support port constraints for Criterion L2 [KSSvH12, SFvHM10]. Minimizing the number of edge crossings and bends (Criterion L4) are both NP-hard problems, but numerous heuristics have been developed [DETT99]. In contrast, the compactness of layouts stated in Criterion L5 has not been addressed much yet in the context of layer-based drawing. Most computed layouts are acceptable w. r. t. compactness, but further research in that area could certainly improve them.

A simple solution to meet Criterion L6, closeness to the original layout, is to extract the layout information from the original view model, attach it to the new view model created in our browsing application, and apply that layout directly to all diagram elements. With this procedure it is possible to obtain identically looking diagrams in both the original tool and the new browsing tool. However, there are two major limiting factors: the approach requires a good hand-made layout that satisfies the first five layout criteria, which is very time-consuming, and it cannot be applied when the sizes of some elements change or new connections are drawn, since that could cause unwanted overlappings. The latter happens in particular when focus & context browsing methods are employed as outlined in Sec. 2.

We propose to use both the original and automatically computed layouts according to the following scheme. We choose one of these alternatives on each hierarchy level of the composite diagram. If none of the nodes on a given hierarchy level are expanded and no new connections to the surrounding level have been added, the original layout is applied, otherwise the automatic layout is applied. This can optionally be enhanced by methods for

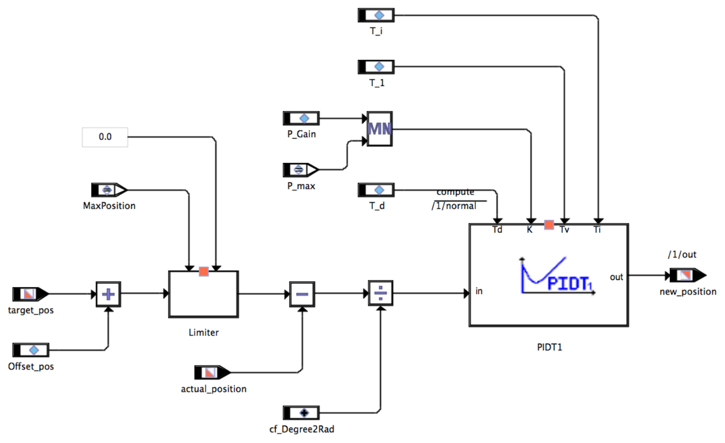


Figure 3: Automatic layout of the ASCET model shown in Fig. 1 (here with FIXEDSIDE port constraints on the Limiter and PIDT1 nodes). The automatic layout is quite similar to the manually drawn one, supporting our assumption that state-of-the-art algorithms are able to provide layouts of adequate quality.

dynamic graph layout [Bra01] using the original layout as prototype, which constrain the computed layout to be as close as possible to that prototype. In our experience, however, today's state-of-the-art layout algorithms already produce layouts of such quality that in most cases the effort of including dynamic layout methods would not pay off. Fig. 3 shows an automatic layout of the diagram in Fig. 1, which is drawn with original layout.

Bridging Hierarchy Boundaries – An Exemplary View Customization

In the graphical notations of actor-based models, (see Fig. 3), each actor is connected with other actors of the same hierarchy level through ports and links. The ports are depicted by little symbols placed onto the boundaries of the figure representing the actor. Regarding the actor itself, those *external* port views are part of the actor's *context*. In contrast, specifications of the interior of non-atomic actors usually represent the actor's ports as floating nodes, which are connected with other elements that are part of the specification (see Fig. 2a). Those *internal* port views are part of the actor's *focus*.

Following the concept of focus & context, our application shall be able to visualize the content of a composite actor surrounded by its context (cf. Criterion H1). However, this concept implies that both the floating internal ports and the actor's external ports are present in the view, which can lead to confusion. According to Criterion H3, internal and external ports shall be connected as shown in the left of Fig. 4. This way the data flow is made explicit and can be followed much easier.

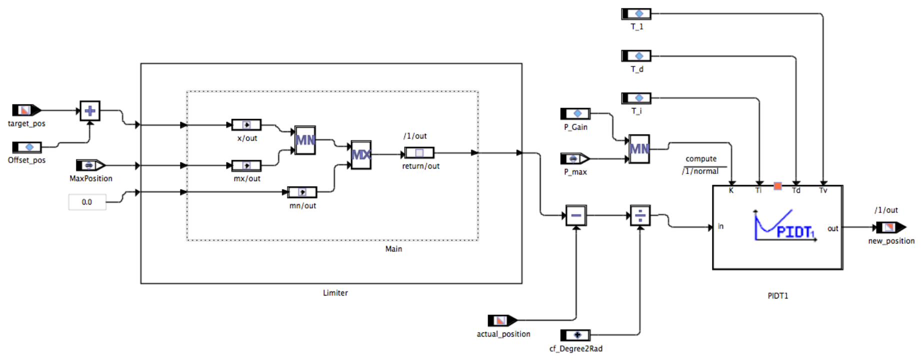


Figure 4: Expansion of the Limiter block with direct links between hierarchy levels.

In most actor-based modeling languages ports are subject to FIXEDPOS constraints (see Sec. 2). However, when focus & context browsing is employed, it is advisable to relax these constraints. Adding edges to connect the content of a focused node with its context and keeping strict port constraints could lead to confusing edge routings: for instance, connections to input ports anchored to the top side would need to be routed all the way to the left side of the contained diagram. If the constraints are relaxed to FREE, in contrast, the layout algorithm can arrange all input ports to the left and output ports to the right, which complies better with the overall flow of connections and thus allows shorter edges and less bend points. The expanded node in Fig. 4, which originally had two input ports on the top side (see Fig. 1), has been drawn with such relaxed constraints. As a consequence, we need a flexible interface in order to dynamically adapt parameters of the layout algorithm such as the port constraints depending on the context. We use the layout configuration interface provided by KIELER for this purpose [SSM$^+$13].

4 Two Approaches for Realization

In this section, we present two different realizations of the transient-views-based concepts introduced in Sec. 3. One uses the established GMF Tooling for rapid prototyping of graphical editors, while the other uses a viewer framework based on KIELER with the focus on high performance and minimizing the time-to-diagram. Both realizations use the KIELER layout algorithms for automatically computing macro layouts as described in Sec. 3. The foundation is laid by an implementation of the layer-based graph layout algorithm with extensions for port constraints and orthogonal edge routing [KSSvH12]. The diagrams shown in Fig. 2b, 3, and 4 all have been arranged with that algorithm.

We employed the two realizations for visualizing Ptolemy models in an open source application, as well as ASCET and Simulink models in an industrial application. The latter is implemented and validated in the EHANDBOOK (ETAS), an Eclipse-based interactive documentation system for ECU software. This system aims to support the efficient explo-

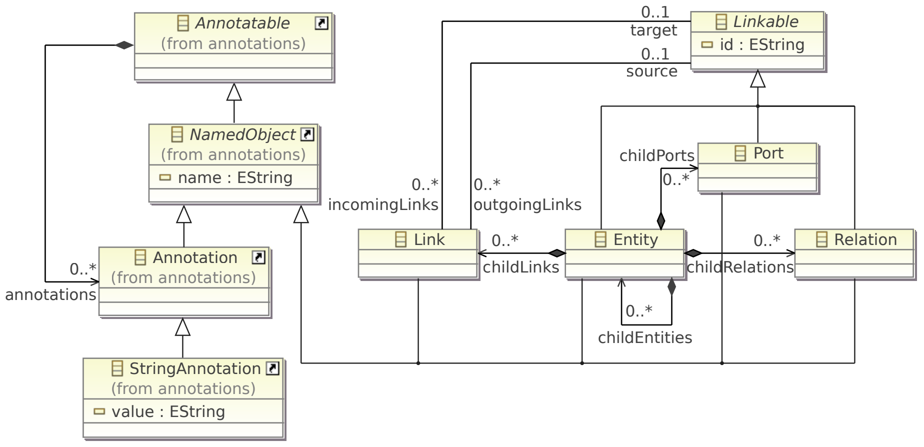


Figure 5: KIELER Actor Oriented Modeling (KAOM) meta model describing structural information and key-value annotations.

ration of complex models and to facilitate the system-wide function understanding needed by calibration engineers.

4.1 Graphical Modeling Framework (GMF)

GMF Tooling uses a model-driven approach to generate graphical editors from abstract specifications. These specifications are built around an application-specific meta model based on the Eclipse Modeling Framework (EMF), which is used to represent concrete model instances. In our application, however, model instances are extracted from different third party tools that are not based on Eclipse. We bridge this technological gap using a generic meta model, called KIELER Actor Oriented Modeling (KAOM) and shown in Fig. 5, that contains only the necessary data for displaying the models. Models from different sources, e. g. Ptolemy, ASCET, or Simulink, are all first transformed into this common EMF-based format. The code generated by GMF Tooling then takes care of creating corresponding diagrams (the *view* and the *controller* in terms of the MVC paradigm). This process involves creating a dedicated concrete view model that is an instance of the GMF *Notation* model for storing macro layout information, a set of *edit parts* for controlling user interaction, and a set of *figures* for drawing the diagram elements.

The KAOM meta model is inspired by the MoML format used by Ptolemy [BLL$^+$08, Chapter 1]. The central class is Entity, which represents nodes of the diagram, e. g. primitive actors such as addition operators or composite actors containing other entities. Actors contain Port instances to describe their interface, and ports can be connected via Link instances. Relation is used to properly represent Ptolemy models, but is currently not used for other languages. Each of these classes can contain Annotation instances, which are basically key-value pairs for attaching arbitrary data to model elements. We use annotations to store the source language and the specific type of an element in order to select the according figure from a predefined library, which is important for rendering the diagram element in the same way as done in its source tool. Furthermore, we add annotations holding the concrete position of each element in the original layout.

GMF supports collapsing and expanding composite nodes, which fits directly with our focus & context approach. In theory it would be possible to load a whole model at once, let GMF create the graphical viewer, and initially collapse all composite actors; users could then selectively expand the actors in their focus. However, many models from industrial applications are too large for this naive approach to work: loading the models would take a long time, or might even fail due to memory limitations. Fortunately, as mentioned in Sec. 2, even for such large-scale applications it is quite typical for each hierarchy level to have a limited number of actors and connections such that they can be printed easily on one page. Following this observation and the approach of Scheidgen et al. [SZFK12], we split the input models such that each hierarchy level is persisted as a fragment. When a diagram is opened, only its top-level fragment is loaded. Upon expansion of a composite actor, its content is loaded lazily from the corresponding fragment, and when it is collapsed, its content is unloaded again. This method limits memory consumption to the subset of model elements that are actually shown in the generated view and greatly reduces the time to open an initial view compared to the standard behavior of GMF, but of course it also raises the time to expand composite actors.

4.2 KIELER Lightweight Diagrams (KLighD)

KLighD enables the visualization of models and other graph-like data in form of node-link-diagrams according to the *transient views approach* [SSvH13]. Its aim is to provide this opportunity without the burden of making oneself familiar with the peculiarities of drawing frameworks and techniques of arranging diagrams. In contrast to GMF Tooling, which derives diagrams from application models in a one-to-one manner, KLighD relies on custom diagram synthesis mappings to formally describe diagrams based on given application data. The view models produced by such mappings adhere to the *KGraph/KRendering* format, which is well-suited for applying automatic layout and modifying diagrams interactively [SSvH12]. The fact that it is specified in EMF's meta modeling language Ecore enables the full integration with Eclipse-based MDSD concepts and tools for implementing diagram synthesis mappings.

The drawings of the desired diagrams, which correspond to the *views* in the MVC pattern, are rendered by the mature 2D graphics framework Piccolo2D,[5] which has been migrated to SWT for use in Eclipse. The life cycle of those diagrams is controlled by an MVC-like *controller* that is part of KLighD. This controller is in charge of updating the views according to changes in the view models, as well as implementing the first class citizen operations hiding and showing, expanding and collapsing, focusing elements, etc. Similarly

[5]`http://www.piccolo2d.org/`

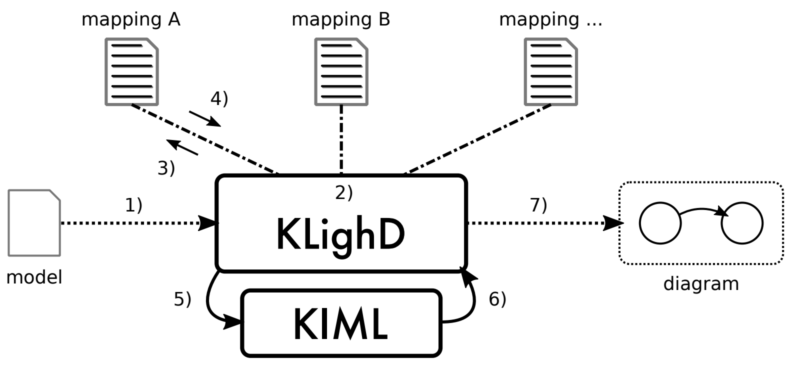


Figure 6: Diagram synthesis process of KLighD [SSvH13]: 1) Request for diagram of application model, 2) mapping selection, 3) mapping application, 4) receipt of corresponding view model, 5) handover to KIML, 6) receipt of view model with layout data, 7) handover to a Piccolo2D diagram canvas and diagram rendering.

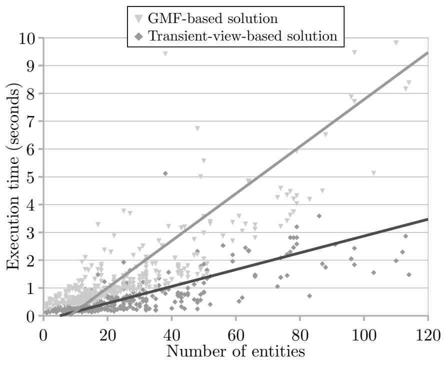
to the GMF-based solution, the arrangement of the diagram elements is contributed by the KIELER Infrastructure for Meta Layout (KIML). The KGraph part of the view model is the input for the KIML component, which evaluates layout directives such as port constraints (see Sec. 2), selects and executes layout algorithms, and augments the view model elements with concrete position information. The procedure of creating graphical views of given models is outlined in Fig. 6.

5 Evaluation

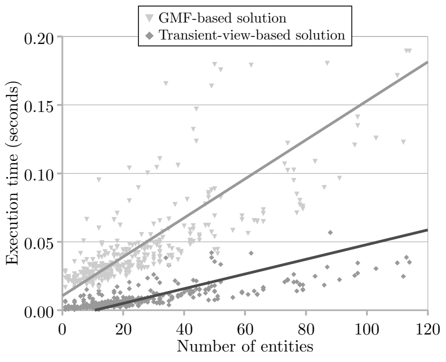
This section presents evaluations comparing the GMF-based and KLighD-based approaches presented in the previous section. We consider two aspects of these approaches: performance and implementation effort.

Performance We measured the execution time first for synthesizing view models and rendering the diagrams, and second for applying automatic layout and updating the diagram rendering. The measurements were performed with about 360 example models provided by the Ptolemy project. These models represent more realistic content than randomly constructed ones do. In addition, this collection covers a reasonable range of diagram elements per model.

Each of those models was examined 5 times with an intermediary sleep time of a few seconds, allowing the tool to perform cleanup operations and the garbage collector to tidy up the memory. Based on the data obtained this way, we computed the mean execution time for opening and closing diagrams of each model, as well as for computing and applying an automatic layout. The result is shown in Fig. 7: we measured an overall average speedup of 2.64 for opening diagrams with KLighD compared to GMF, and a speedup of 7.41 for



(a) Opening diagrams



(b) Automatically arranging diagrams

Figure 7: Experimental measurement results for execution time.

automatically arranging diagrams. The superior fluidity of the KLighD-based viewer is noticeable at first glance while using the tool, especially for operations such as collapsing or expanding composite elements.

We monitored the heap memory that was used by the whole application for both techniques by means of the VisualVM[6] tool. With large examples we observed a reduction of up to 50% for the KLighD-based approach compared to GMF. Since the concrete measured amounts of consumed memory include the offset required by the application platform, the ratio of the adjusted values would be even more in favor of KLighD. The measurements were done on a typical mobile business computer with a quad core CPU, a memory of 8GB, and an up-to-date Java Runtime Environment (JRE) installed.

Comparison of Implementations We experienced several problems of the GMF-based solution regarding its implementation and maintenance. While the time to obtain a first version of a diagram editor for KAOM models is very short, the realization of many further features and details requires a lot of effort. The GMF Tooling generated 96 Java classes with over 12 000 lines of code; understanding that code and how it relates to the corresponding source models is a tedious task, but regrettably it is often necessary. The feature that involved most effort was the accurate reconstruction of the figures for rendering the many different node types of the source languages, especially considering that they are all represented by the class Entity in the KAOM model. The code generated by GMF had to be extended in order to dynamically adapt the visual representation of each entity depending on annotations of the corresponding KAOM model element.

The KLighD-based solution allows much more direct and light-weight modifications of the created diagrams. In particular, the indirection of an intermediate meta model such as KAOM is not required, and adapting the rendering of entity figures can be done in a descriptive manner using elements of the KRendering meta model. This leads to a more

[6]`http://visualvm.java.net/`

intelligible and maintainable code base. For instance, the GMF-based visualization of Ptolemy models was implemented in the Xtend[7] language and compiled to 1372 lines of Java code with 733 lines of hand-written code for the transformation to the KAOM format, plus 1576 lines for the correct rendering of Ptolemy diagram elements, 5337 lines generated by EMF for the KAOM meta model, 2374 lines of generic extensions of the GMF editor code, and the aforementioned generated GMF code, which was customized with 14 hand-edited code generation template files. This amounts to a total of roughly 24 000 lines of code. The KLighD-based visualization with the same functionality is made of Xtend code that compiles to 4829 lines of Java code with 884 lines of hand-written code, which is less than 6 000 lines in total.

6 Summary and Future Work

Today's modeling tools provide reasonable support for application developers, who are typically responsible for just a small portion of the system. However, it is sometimes necessary to get an understanding of overall system functionality and to extract information that is spread over a range of components. We have identified a number of requirements that arise here, and have presented a concept combining *transient views* and *automatic layout* to address them. The concept has been realized with two different Eclipse-based technologies: GMF and KLighD. The presented methods allow the seamless browsing of previously fragmented models as well as the integrated handling of heterogeneous models comprising different source notations.

Comparing the two realizations of the transient views concept, we found that KLighD allows to implement such applications with less effort both for the first prototypes and in the long term compared to GMF. Furthermore, it reaches much better performance both in terms of execution time and memory consumption. Hence, KLighD meets its design objective stated in [SSvH13] in this application, and, as a bottom line, we would not recommend employing a heavy-weight editor framework such as GMF when the goal is merely visualizing and browsing models, but not editing.

First practical experiences with real-world models of the automotive industry have confirmed our thesis that automatically arranged models can easily be understood. The automatic layout algorithms that take into account the positioning of ports optimize the readability of the graphical models. This offers large time-saving potential for engineers who are used to work with classical, page-oriented documentation.

While the pilot users of the EHANDBOOK solution at ETAS report promising experiences, a substantial user study, evaluating the impacts on the daily work routine, has yet to be performed. We also plan to integrate further methods supporting the understanding of the models, e. g. dynamic exploration during the simulation of a model and the visualization of time-critical paths based on profiling information. Another area for future work is the further optimization of automatic layout algorithms in the context of hierarchical data-flow models and very-large-scale models.

[7] `http://www.eclipse.org/xtend/`

References

[BLL+08] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuendorffer, Yang Zhao, and Haiyang Zheng. Heterogeneous Concurrent Modeling and Design in Java, Volume 2: Ptolemy II Software Architecture. Technical Report UCB/EECS-2008-29, EECS Department, University of California, Berkeley, April 2008.

[Bra01] Jürgen Branke. Dynamic Graph Drawing. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs: Methods and Models*, volume 2025 of *LNCS*. Springer, 2001.

[BRSG07] Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. The Aesthetics of Graph Visualization. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging (CAe'07)*, pages 57–64. Eurographics Association, 2007.

[BSLF06] Robert Ian Bull, Margaret-Anne Storey, Marin Litoiu, and Jean-Marie Favre. An Architecture to Support Model Driven Software Visualization. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 100–106. IEEE, 2006.

[BT00] Stina Bridgeman and Roberto Tamassia. Difference Metrics for Interactive Orthogonal Graph Drawing Algorithms. *Journal of Graph Algorithms and Applications*, 4(3):47–74, 2000.

[DETT99] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[EJL+03] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming Heterogeneity—The Ptolemy Approach. *Proceedings of the IEEE*, 91(1):127–144, Jan 2003.

[GHL+13] John C. Grundy, John Hosking, Karen Na Li, Norhayati Mohd Ali, Jun Huh, and Richard Lei Li. Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications. *IEEE Transactions on Software Engineering*, 39(4):487–515, April 2013.

[Kla12] Lars Kristian Klauske. *Effizientes Bearbeiten von Simulink Modellen mit Hilfe eines spezifisch angepassten Layoutalgorithmus*. PhD thesis, Technische Universität Berlin, 2012.

[KSSvH12] Lars Kristian Klauske, Christoph Daniel Schulze, Miro Spönemann, and Reinhard von Hanxleden. Improved Layout for Data Flow Diagrams with Port Constraints. In *Proceedings of the 7th International Conference on the Theory and Application of Diagrams (DIAGRAMS'12)*, volume 7352 of *LNAI*, pages 65–79. Springer, 2012.

[LMB+01] Ákos Lédeczi, Miklós Maróti, Árpád Bakay, Gábor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Péter Völgyesi. The Generic Modeling Environment. In *Workshop on Intelligent Signal Processing*, 2001.

[LNW03] Edward A. Lee, Stephen Neuendorffer, and Michael J. Wirthlin. Actor-Oriented Design of Embedded Hardware and Software Systems. *Journal of Circuits, Systems, and Computers (JCSC)*, 12(3):231–260, 2003.

[Min06] Mark Minas. Generating Meta-Model-Based Freehand Editors. In *Proceedings of the 3rd International Workshop on Graph Based Tools (GraBaTs'06)*, volume 1 of *Electronic Communications of the EASST*, Berlin, Germany, 2006.

[MLC06] Gergely Mezei, Tihamér Levendovszky, and Hassan Charaf. Visual Presentation Solutions for Domain Specific Languages. In *Proceedings of the IASTED International Conference on Software Engineering*, Innsbruck, Austria, 2006.

[RMG07] Tobias Reinhard, Silvio Meier, and Martin Glinz. An Improved Fisheye Zoom Algorithm for Visualizing and Editing Hierarchical Models. In *Second International Workshop on Requirements Engineering Visualization*, pages 9–19. IEEE, 2007.

[San04] Georg Sander. Layout of Directed Hypergraphs with Orthogonal Hyperedges. In *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, volume 2912 of *LNCS*, pages 381–386. Springer, 2004.

[SB92] Manojit Sarkar and Marc H. Brown. Graphical Fisheye Views of Graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 83–91. ACM, 1992.

[SFM99] Margaret-Anne D. Storey, F. David Fracchia, and Hausi A. Müller. Customizing a Fisheye View Algorithm to Preserve the Mental Map. *Journal of Visual Languages & Computing*, 10(3):245–267, 1999.

[SFvHM10] 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.

[SSM$^+$13] Miro Spönemann, Christoph Daniel Schulze, Christian Motika, Christian Schneider, and Reinhard von Hanxleden. KIELER: Building on Automatic Layout for Pragmatics-Aware Modeling (Showpiece). In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, 15–19 September 2013.

[SSvH12] Christian Schneider, Miro Spönemann, and Reinhard von Hanxleden. Transient View Generation in Eclipse. In *Proceedings of the First Workshop on Academics Modeling with Eclipse*, Kgs. Lyngby, Denmark, July 2012.

[SSvH13] Christian Schneider, Miro Spönemann, and Reinhard von Hanxleden. Just Model! – Putting Automatic Synthesis of Node-Link-Diagrams into Practice. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, 15–19 September 2013. With accompanying poster.

[STT81] 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.

[SWFM97] Margaret-Anne D. Storey, K. Wong, F. David Fracchia, and Hausi A. Müller. On integrating visualization techniques for effective software exploration. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 38–45. IEEE, 1997.

[SZFK12] Markus Scheidgen, Anatolij Zubow, Joachim Fischer, and Thomas H. Kolbe. Automated and Transparent Model Fragmentation for Persisting Large Models. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS'12)*, volume 7590 of *LNCS*, pages 102–118. Springer, 2012.

[SZG$^+$96] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, 3:162–188, 1996.