Towards Effective Visual Modeling of Complex Software Product Lines*

André Heuer, Kim Lauenroth, Marco Müller
Paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen, 45127 Essen
{andre.heuer | kim.lauenroth | marco.mueller}@paluno.uni-due.de

Jan-Nils Scheele
adesso AG
Stockholmer Allee 24, 44269 Dortmund
scheele@adesso.de

Abstract

The variability of a software product line is one of the main reasons for complexity in product line engineering. To deal with this complexity, several researchers propose an orthogonal approach for variability modeling. The central idea of these approaches is to document the variability and the product line artifacts in separate models and document the impact of the variability in the form of relationships between the variability model and the product line model.

From a theoretical point of view, the orthogonal variability modeling approach offers several benefits over other variability modeling approaches. However, the visualization of the models and an effective visual modeling with the orthogonal approach constitute a significant challenge. In this paper, we describe several challenges for the effective visual modeling based on the orthogonal approach and describe solution ideas that address these challenges. As a first step towards an evaluation of our solution ideas, we present two different empirical evaluation strategies.

1. Introduction

The variability of a software product line is one of the main reasons for complexity in product line engineering [15]. To deal with this complexity, several researchers propose an orthogonal approach for variability modeling [1], which offers several benefits over the integrated representation of variability in development artifacts [12]. An orthogonal approach documents the variability of the product line in a separate model and the effects of the variability on the product line artifacts (e.g. requirements, design, or code) by means of relationships between the variability model and the product line artifacts (see Fig. 1).

Recent examples of such orthogonal variability modeling approaches are feature-based model templates [4], feature transition systems [3], and variable I/O-automata [11]. These and similar approaches can be considered as successful approaches for documenting variability in various development artifacts. However, the purpose of these and similar approaches is the analysis or the verification of the documented artifacts. The effective visualization of the variability model or the product line artifacts is not their focus. However, an effective visualization and support for the creation of such models (which we call visual modeling) is a crucial factor for the successful industrial application of such approaches.

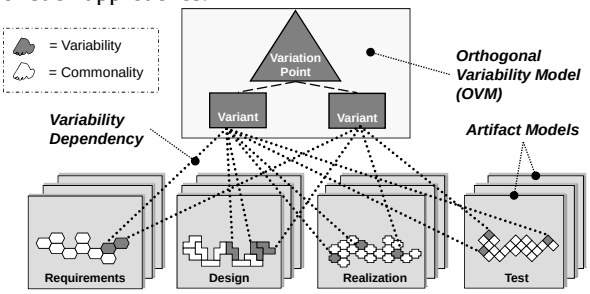

Fig. 1. Orthogonal Representation of Variability [14]

The goal of this paper is to go a first step towards an effective visual modeling of complex software product lines based on an orthogonal variability modeling approach. In order to achieve this goal, this paper provides the following contribution:

1) *Challenges for an effective visual modeling*: In order to understand the meaning of effective visual modeling, this paper discusses challenges for the visual modeling in product line engineering
2) *Presentation of solution ideas*: We are currently working on different approaches that address our identified challenges for visual modeling in product line engineering. In this paper, we present our prototypical modeling environment that implements the solution ideas for the identified challenges
3) *Discussion of empirical evaluation strategies*: Visual modeling approaches have to prove their worth during product line engineering activities. However, the evaluation of a modeling environment in a complex case study constitutes a significant effort. We therefore discuss different strategies and criteria for the empirical evaluation of the presented solution ideas in smaller experimental settings.

* This paper was partially funded by the DFG, grant PO 607/2-1 IST-SPL.

The remainder of this paper is structured as follows: Section 2 presents our identified challenges for an effective visual modeling. Section 3 discusses related work in terms of existing modeling tools. Section 4 and 5 presents our modeling environment and our solution ideas. Section 6 presents the evaluation strategies and Section 7 closes this paper with a summary and an outlook.

2. Challenges of an Effective Visual Modeling in Product Line Engineering

In this section, challenges of an effective visual modeling are discussed. We distinguish challenges related to the modeling of variability and challenges related to the modeling of product line artifacts.

2.1. Challenges related to the Modeling of Variability

The variability model of a product line can become large and complex. Industrial case studies report on product lines with several hundred and even more variation points and variants [2; 14]. From this observation, we derive the following challenge:

C1: An effective visual variability modeling approach must support the creation and visualization of a variability model with several hundred variation points and variants.

The creation of a variability model especially includes the definition of constraints between variation points and variants (e.g., one variant excludes another variant). However, representing such a relationship in a large variability model is difficult, since the corresponding variants (or variation points) may not be positioned close to each other. From this observation, we derive the following challenge:

C2: An effective visual variability modeling approach must support the definition of variability constraints in large and complex variability models.

Constraints in a variability model are not only a challenge in the creation process of a variability model, but also a challenge for the understanding of a variability model [5]. From this observation, we derive the following challenge:

C3: An effective visual variability modeling approach must support the well-arranged visualization of constraint relationships in large and complex variability models.

Another important aspect for the visualization of constraints is the visualization of the effects of a constraint during the product derivation process in application engineering. In a model, it should, for example, be clearly visible if a variant is no more selectable because of a constraint dependency to another already selected variant or variation point. From this observation, we derive the following challenge:

C4: An effective visual variability modeling approach must support the visualization of effects of constraint relationships in large and complex variability models.

2.2. Challenges related to the Modeling of Product Line Artifacts

Even without taking product line engineering into account, the effective visual modeling of a system is considered to be a significant challenge [13]. Therefore, we focus in this section on challenges that are related to product line engineering and neglect challenges that also apply to the modeling of individual systems (e.g. complexity and size of design models). However, we necessarily have to take into account that product line models become at least as large and complex as models from single system engineering:

C5: An effective visual variability modeling approach must support the creation and visualization of large and complex product line models.

A product line consists of common and variable artifacts [14]. Therefore, the visualization of the variability quality (being common or variable) of a product line model element is an important challenge:

C6: An effective visual variability modeling approach must provide the information whether product line artifacts are common or variable.

However, visualizing the variability quality is not enough. The definition and visualization of the relationships between the variants of the variability model and the product line models is also an important challenge, since these variability relationships define whether an artifact is considered to be common or variable. The definition of the variability relationship is a challenge, since the variability model as well as the product line models may become large and complex. From this observation, we derive the following challenge:

C7: An effective visual variability modeling approach must support the definition of variability relationships between large and complex variability models and large and complex product line models.

Beside a visual mechanism for the definition of relationships, we consider the visualization of the variability quality of model parts in a product line models as an important challenge.

C8: An effective visual variability modeling approach must clearly visualize common and variable parts in large and complex product line models.

3. State-of-the-Art of PLE Tools and Visualization

This section provides an overview of tools that enable the modeling of a product line including the vari-

ability model and product line models and their relation.

The software pure::variants [17] is an industry-strength product designed for the specific needs of large projects. In pure::variants, variability is modeled in a feature tree. The relationship between variants and product line model elements is represented by keywords or structures, e.g. structured comments in source code, special attributes in XML, or file paths. The relationship is, however, not emphasized in the visual representation, making it hard to trace the impact of variants.

Colored Integrated Development Environment (CIDE) [9] is a tool for visualizing the implementation code which is related to different variants. In CIDE, colors are used to distinguish the variable implementation parts. Each variant is represented by a specific color, and the related implementation parts are colored accordingly. This concept can also be used to annotate e.g. whole files, directories, or structures in XML files. When variants are selected, filters can be used to hide the parts that are not relevant. The usage of different colors reduces the scalability of the approach, since people can only distinguish a limited number of colors and, as mentioned above, in product line engineering it is common to have several hundred variants.

FeatureMapper [7] uses four mapping views to visualize relationships: (1) The Realization View emphasizes the artifacts used to realize a selected variant. The other elements are shown in grey. (2) The Variant View acts like the Realization View, but additionally emphasizes all common artifacts. (3) The Context View can be used to display the relationships by colors: each variant is represented in a different color which is then also used to color related product line model elements accordingly. (4) In the Property-Changes-View, the properties, like cardinalities, are emphasized by showing the changes and hiding the parts of the domain model that are not relevant for this change. The Realization View and the Variant View emphasize the information that is in focus, while keeping the context visible. The Context View has the same disadvantages as CIDE, due to the coloring effect for distinguishing variants. The Property-Change-View addresses the issue that changed properties are not new or removed elements. This issue is not addressed by the other approaches, but not in the focus for our tool.

In Stoiber et al. [16], a set of visualization aids are presented. This approach uses decision tables for variability modeling. The visualization of the product line models directly depends on the selection of variants in the decision table. At the beginning of the decision process, the complete product line model is shown. Each decision tailors the model by removing artifacts that are not part of the derived product based on the variability model. When all decisions are made, the remaining model is the application model. Furthermore, many visualization aids are provided to support coping with the complexity of the domain model: Horizontal abstraction hides views that are not necessary for the current selection. Vertical abstraction hierarchically hides objects within views. Visual pointers are used for representing the relationships between the variants and the product line model. When a variant in the decision table is selected, the related product line model elements are highlighted, and related constraints are shown. In addition, the Impact View of a variant is a temporary view on the model that shows the affected domain model artifacts, while hiding all unrelated elements. The automatic, decision-based tailoring of the domain model in this approach facilitates the impact analysis before decision-making by selecting variants and exploring their effects in the accordingly tailored model.

The presented tools allow to model and visualize variability models and simple relations to development artifacts. The derivation of applications by selecting specific variants is also supported by the tools. However, none of these tools provide dedicated mechanisms that address the modeling of large and complex product line models nor the support of the OVM.

4. Tool Support for Model-based Product Line Engineering

This section gives an overview of our prototypical tool environment which we used to implement our solution ideas for the challenges presented in Section 2. We call our environment Remidemmi, which is the abbreviation for *R*equirements *E*ngineering and *M*anagement *i*n *D*omain *E*ngineering with *M*ulti-*M*odel *I*nteraction.

4.1. Remidemmi – Technical Overview

Our tool implementation is written in Java code and is based on the Eclipse Rich Client Platform (RCP)[1]. Eclipse is well known as an Open Source framework that offers a loosely coupled and dynamic plug-in architecture. For example, all Eclipse IDE´s (e.g. for Java or C development) are based on the Eclipse RCP framework and just a set of plug-ins that extend the core components with the functionalities needed to develop, compile, and debug Java or C code. We also used this framework and extended it according to our needs, because it allows us to still integrate new models into the current framework later on. We used additional frameworks for the development of our tool. The extensions we mainly used are listed below:

- *Eclipse Modeling Framework (EMF)*[2]. This is a modeling framework for code generation. It sup-

[1] See www.eclipse.org/rcp

[2] See www.eclipse.org/emf

ports creating meta models and generating code based on the meta model.

– *EMF Validation Framework*[3]. This framework allows for applying validation methods to instances of EMF models.
– *Graphical Editing Framework (GEF)*[4]. The GEF is a framework that supports the developer in creating visual editors, e.g. model editors.
– *Graphical Modeling Framework (GMF).* The GMF is a code generation framework that generates a GEF based editor on the basis of an EMF model.

Additionally, we used the Java Binary Decision Diagram[5] implementation (JavaBDD) as SAT-Solver implementation. JavaBDD was used because the Boolean equations of the variability model formalization can be efficiently evaluated by BDDs. JavaBDD was isolated in a separate plug-in to capsulate the functionality.

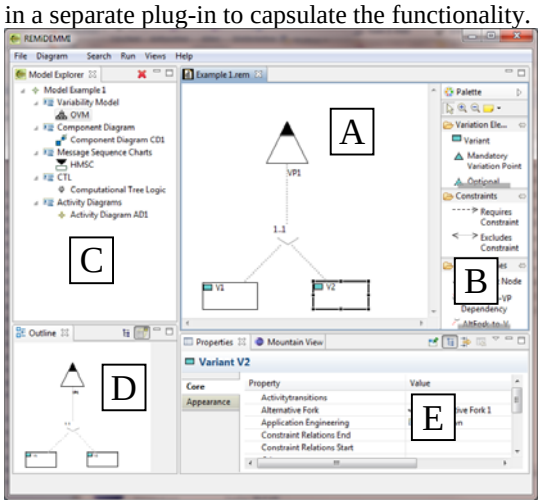

Fig. 2 Overview of the Remidemmi UI

4.2. Editors in Remidemmi

The user interface of our tool is based on a multi-window interface, i.e. different windows (so-called views) can be opened around multiple editor windows.

Fig. 2 gives an overview of the user interface of Remidemmi. The central elements in the user interface are the editors. They are placed in the middle of the window, marked with (A). The frame marked with (B) contains the tool palette for the editor, where the tools (e.g. the model elements, zoom, selection, notes, etc.) can be selected and applied to the main editor. The Model Explorer can be found in (C). It shows a hierarchical list of all types of models that are available and available instances of the models. In the lower left corner (D), the outline view is shown, which gives an overview of the current model in the editor and the possibility to navigate through the model. Beside the outline view, in the frame marked with (E), additional views are available. An important view is the properties view that allows the manipulation of the properties of the currently selected model element. Additional available views are the mountain view and the connection view that will be described in Section 5. Remidemmi provides several editors for the specification of variability in product line artifacts. In the following, the different available editors are briefly described.

Variability Model Editor. This editor supports the creation of a variability model in the OVM notation (cf. [14]). The editor is shown in Fig. 3.

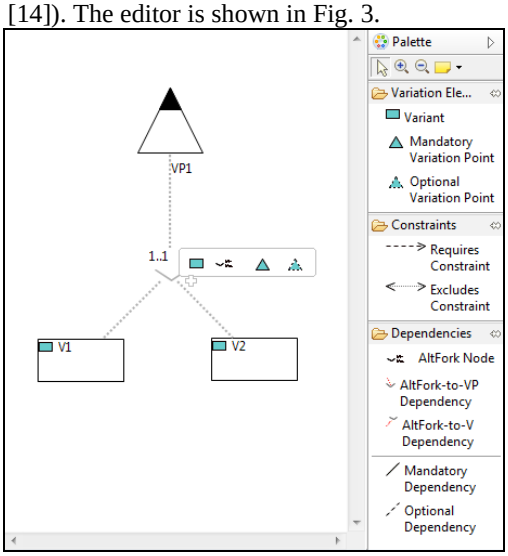

Fig. 3 The OVM editor

Component Diagram Editor. The notation for component diagrams is based on a simplified UML notation. However, we use rectangles for each component. Interfaces between components are shown as lines. Fig. 4 shows the editor.

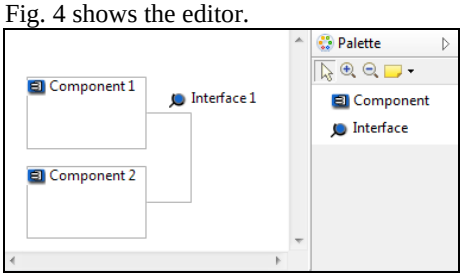

Fig. 4 The component diagram editor

I/O automaton editor. In Remidemmi, I/O automata are called state machines. They model the internal behavior of the components and their communication triggered by events. Therefore, a component (modeled in the component diagram editor) may contain a state machine. Fig. 5 shows the editor and the events view.

[3] See http://eclipse.org/modeling/emf/?project=validation
[4] See www.eclipse.org/gef
[5] See javabdd.sourceforge.net

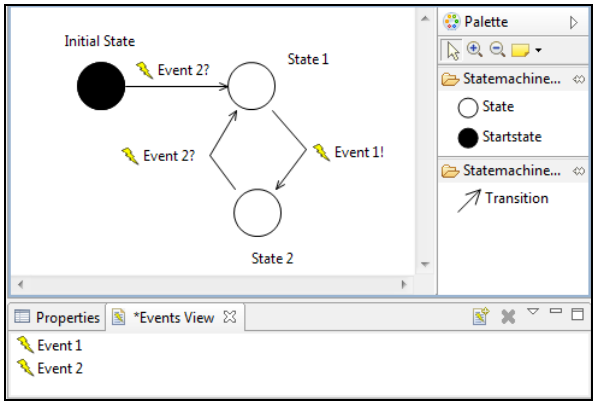

Fig. 5 I/O automaton editor and events view

Message sequence charts editors. Message sequence charts (MSC) require two editors: one for the hierarchical (h)MSC and one for the basic (b)MSCs. The model explorer allows creating one hMSC. The hMSC may contain several bMSCs. The corresponding editor can be opened by a double-click on the bMSC in the hMSC editor. Fig. 6 shows both editors (left-hand side hMSC, right-hand side bMSC).

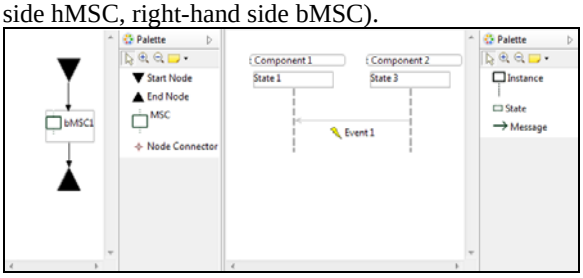

Fig. 6 hMSC and bMSC editor

Computational Tree Logic (CTL) editor. The CTL editor is a hierarchical editor that supports the composition of CTL expressions. Below a CTL expression, operands have to be created pointing to a specific state in a state machine. Every operand can be negated. In the CTL expression, the operands have to be selected as well as an operator between them. Fig. 7 shows a simple CTL expression in the editor.

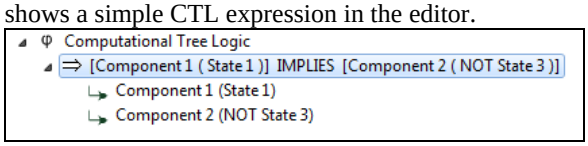

Fig. 7 CTL expression in the CTL editor

Activity diagrams editor. We used a reduced set of model elements for the activity diagrams editor. The editor only supports activities themselves and decisions. A screenshot of the editor is shown in Fig. 8.

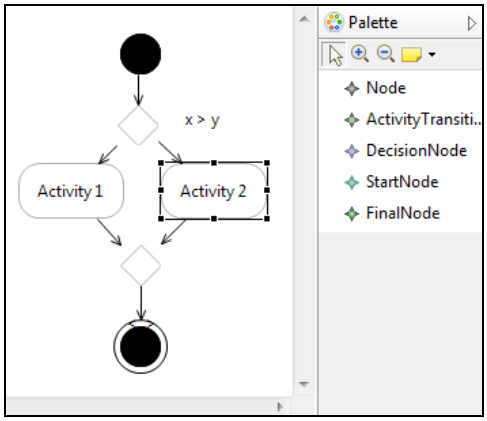

Fig. 8 Activity diagram editor

5. Addressing the Challenges

This section presents solution ideas that are implemented in our tool that are addressing the specific challenges introduced in section 2.

5.1. Modeling Variability

Variability modeling tools allow creating variability models and provide different zoom levels for visualizing the variability model. In order to get an overview of a large variability model, the engineer has to choose a high level zoom or has to look at parts of the variability model.

In order to improve the visualization of large and complex variability models including relationships, we have developed two solutions ideas: the mountain view (Section 5.1.1) and the lasso view (Section 5.1.2). For improving the visualization of constraint effects in a variability model, we have developed a coloring mechanism (Section 5.1.3).

5.1.1 Mountain View for Variability Models

The idea of the mountain view is based on abstraction and allows the engineer to decide, which elements he wants to display. The mountain view introduces abstraction layers that can be used to include or exclude variability model elements. The lowest abstraction layer shows the whole diagram with all elements of the variability model. All further layers are hierarchically ordered and impose specialization on the elements, meaning that each element on one layer also appears on all lower layers.

The layer of an element is completely user defined, there is no computation done by the editor. Thus, each user of the editor could save its own interpretation of the model. Still, if a model element is very important, it is very likely that it is on a high abstraction layer, i.e. the model elements appear on the abstraction layers depending on their importance.

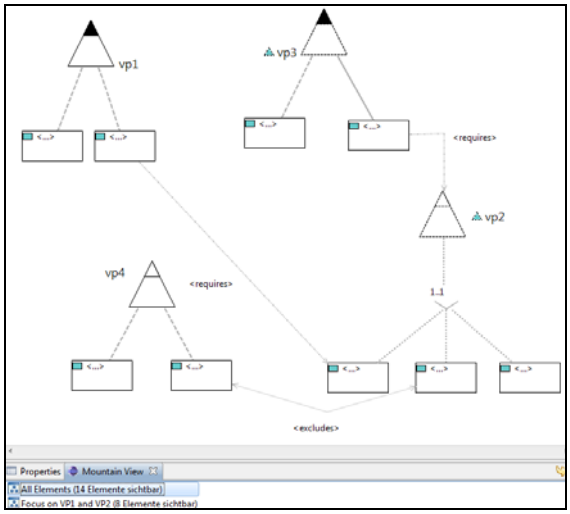

Fig. 9 Complete Variability Model on lowest layer

On the highest abstraction layer, the most important model elements can be found. This can result in a diagram with lots of empty spaces between the model elements. Fig. 9 shows an example for a model. For illustration purposes, it is kept simple. As seen at the bottom, each layer can be renamed and automatically displays the total number of elements on it. In order to reduce empty space in a variability model, each abstraction layer documents the spatial position of the model individually. Thereby, it is possible to position a model element in different places for different abstraction layers. This allows for repositioning the model elements and helps reducing the size of the variability model at each abstraction layer.

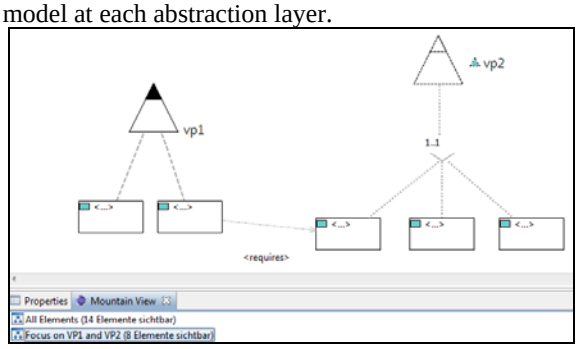

Fig. 10 Focus on two variation points

Fig. 10 shows an example for this. A new layer for the variability model in Fig. 9 was created to focus only on the variation points *vp1* and *vp2* and their variants.

This functionality allows for the definition of abstraction layers with totally different shapes. A layer's change would result in jumping model elements and would hinder the understandability of the new selected layer. To ease a layer change, the mountain view implements a floating mechanism which moves each visible element gently to its new position. Thereby, the engineer can follow visually the abstraction layer change and the new position of elements.

The mountain view idea addresses challenges C1 – C3 as follows:

- Challenge C1 is addressed by the fact that a large and complex variability model can be visualized in different abstraction levels. The creation process benefits from the mountain view since additional model elements can be defined in the context of a single abstraction layer.
- Challenges C2 and C3 are addressed since the abstraction layer can be used to define and visualize relationships in a reduced variability model.

5.1.2 Lasso View for Variability Models

In large and complex variability models related elements may reside outside the currently visible area. This especially hinders the understandability of relationships between visible and invisible model elements (e.g. a constraint relation between two variants).

The idea of the lasso view is to bring the related elements into focus of the engineer. This is accomplished by introducing context information into the diagram. The name of this view reflects the idea of fetching the related elements of a selected element and pulling them directly towards the selected element until they reside at the border of the viewable area.

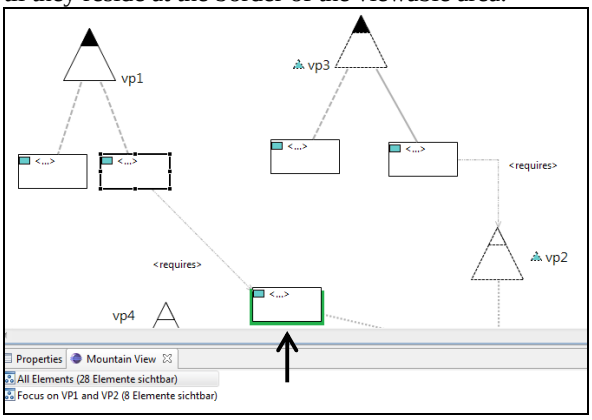

Fig. 11 Using the lasso view

Fig. 11 illustrates the use of the lasso view. By selecting the variant on *vp1*, the target variant of the requires relationship is pulled from the not-visible area of the editor to the border of the editor window and highlighted by a bold border.

Using this technique one can identify all elements affected no matter where they are located within the variability model. The engineer can now click on any element on the border and is directly transferred to the according element at its original position. Automatically all links of the new element are displayed. This way, the engineer could just jump back to where he came from or continue within the diagram.

The lasso view idea addresses challenges C1 and C3 as follows:

- Challenge C1 is addressed since the lasso view can be used to navigate through a large and complex variability model.
- Challenge C3 is addressed since the lasso view allows the visualization of all elements that are related to a particular model element by moving them into the visible are of the model editor.

5.1.3 Coloring Mechanism for Constraints

For the visualization of effects of constraints in large and complex variability models (see Challenge C4), the variability modeling editor provides a coloring mechanism. If the engineer, for example, selects a set of variants the editor evaluates the variability dependencies and the constraints and gives feedback to the engineering by highlighting the model elements with different colors. Fig. 12 shows an example for this coloring.

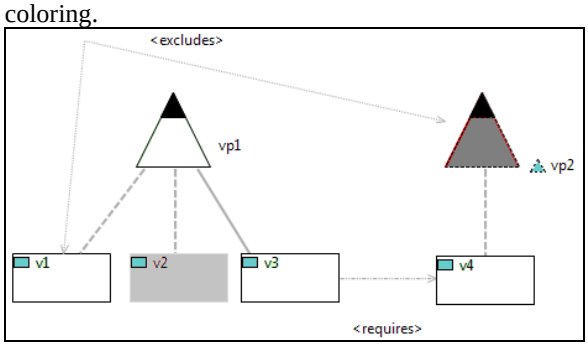

Fig. 12. Example for the Coloring Mechanism

In this example, the variant *v1* is selected. The editor evaluates all dependencies beginning from this variant. The variants, that are part of the product, respectively the variation points that were regarded, are highlighted in light green (e.g. *v1*, *vp1*, *v3*, *v4*). For illustration purposes, we used white in the figure. Variants or variation points that are not part of the product, or respectively not regarded, are highlighted in light red (e.g. *v2*, here light gray). In Fig. 12, the algorithm detects an error: if *v1* is bound, the variation point *vp1* has to be regarded. This results in a selection of variant *v3*, because it is a mandatory variant on *vp1*. Variant *v3* requires the variant *v4* that has a variability dependency to the variation point *vp2*, but *vp2* has a constraint dependency to *v1*, which was selected to be part of the product. That means, there is a problem in the variability model. Therefore, the variation point *vp2* is highlighted in dark red (here dark grey), because the dependencies on *vp2* cannot be validly solved.

5.2. Modeling Product Line Models

The orthogonal variability modeling approach requires that the product line models are documented independently from the variability model and that the impact of variability on a particular product line model element is document by relationships between product line model elements and the variability model. Thus, the creation and visualization of these relationships is the central challenge for the visual modeling of software product lines. In order to improve the visualization of the relationships, we have developed a coloring mechanism within the particular models (Section 5.2.1) and a dedicated connection view for the relationships (Section 5.2.2).

5.2.1 Coloring Mechanism for Artifact Models

In the tool, several views on the models can be open in the same time. For visualizing relationships between different models, our tool implements coloring mechanism which is based on selected model elements. When variants are selected in the variability model, the product line model elements that are related to the selected variants are highlighted to visualize the relationship.

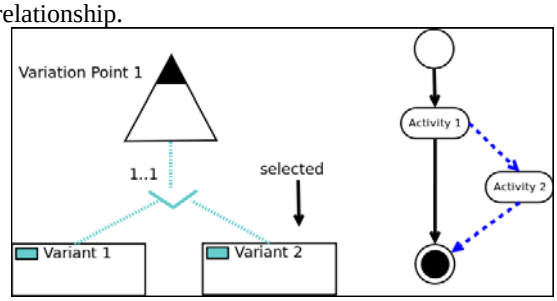

Fig. 13: Visualization of common and variable model elements

Fig. 13 shows a screenshot of our coloring mechanism. Two views are opened: the variability model on the left and an activity diagram as product line model on the right hand side. *Variant 2* is selected by the user. The activity diagram shows that one transition is common (black), two transitions (dashed) are related to the selected *Variant 2* and one transition (grey) is related to a variant that is not selected.

The coloring mechanism is also able to visualize the variants related to a product line model element. When model elements are selected, their related variants are highlighted in the variability model.

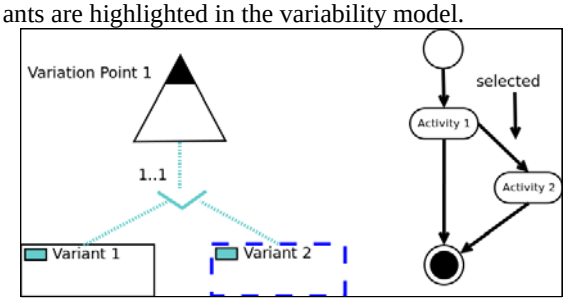

Fig. 14: Visualization of variants related to a model element

5.2.2 Connection View for Variability

The idea of the connection view is to provide an independent tree-like view on the relationship between the variability model and the various product line models.

Initially, the connection view is an empty list without any content. When an element is dropped from the editor window into the connection view, it becomes the root element of the view. Depending on the dropped element, the connection view behaves as follows:

- When a *variant* is dropped into the connection view, the view shows all product line model elements which are related to the selected variant as child elements.
- When an *artifact model* element is dropped into the connection view, the view shows the variants that are related to the selected model element as child elements.

Since the connection view follows the tree paradigm for the visualization, the elements related to a child element can also be visualized by opening them.

Fig. 15 illustrates the visualization capabilities of the connection view by an example. First, Variant 2 was dragged and dropped into the view. By expanding the entry of the variant, all related artifacts are shown, e.g. variable transitions as shown in Fig. 15.

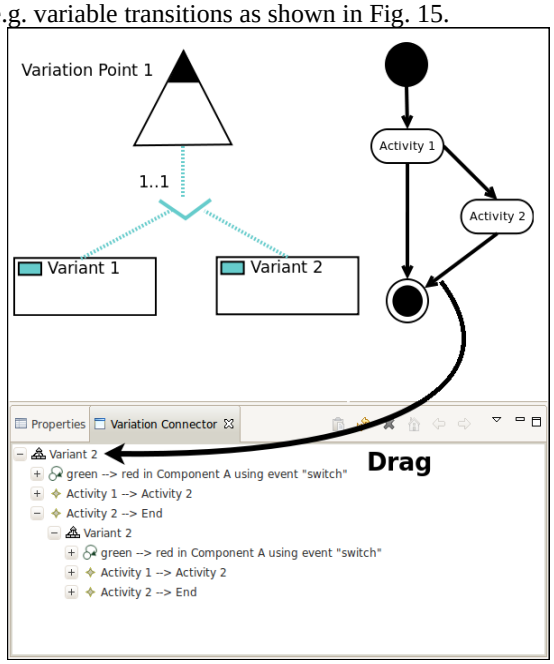

Fig. 15: Visualization and definition of variability relations in the connection view.

The connection view serves not only as a visualization mechanism. The connection view also supports the definition of relationships between the variability model and product line model elements within the whole tree by dragging them on the variant in the connection view.

Fig. 15 illustrates this: to add a new variation dependency, e.g. from the transition from *Activity 2* to the final node to *v2*, the transition has to be dragged and dropped onto the *Variant 2* in the connection view as illustrated in Fig. 15

The connection view addresses challenges C5-C7 as follows:

- Challenge C5 is addressed by the fact that the connection view can be used as independent view on the variability relation. The tree-like visualization approach furthermore enables to browse through the relationships between the variability model and the various product line model elements
- Challenge C6 is addressed by the connection view since it visualizes information about the variants that are related to a product line model element.
- Challenge C7 is addressed by the fact that the connection view also enables the definition of relationships at various levels of the relationship tree.

5.3. Conclusion

In this chapter we have presented solution ideas for the visual modeling of software product lines. The mountain view idea and the lasso view idea are to the best of our knowledge novel ideas. However, the idea of using colored model elements or the tree-like approach of the connection view itself is not a novel idea in product line engineering visualization research (see Section 3).

Nevertheless, we believe that the integration of these approaches with our presented visualization approaches for variability modeling (mountain view and lasso view) will greatly improve the visual modeling in product line engineering. In the following section, we will discuss possible strategies for the empirical evaluation of our claim.

6. Empirical Evaluation Strategy

Visual notations and modeling approaches in computer science are intended to support human beings involved in software engineering [6]. In our case, the goal of the visual modeling approach is to support human beings involved in product line engineering. The central measurement of a visual modeling approach is the cognitive effectiveness [13] which can be defined as the processing speed, ease and accuracy of the human mind [10]. In the following, we will present two possible experimental settings that measure the cognitive effectiveness of our presented solution ideas.

6.1. Task-oriented Experiments

We plan to define a series of experiments in which the participants perform task related to product line engineering and variability modeling with our tool prototype. We distinguish three kinds of task: creational task, modification tasks and analytic task.

In a creational task, the participant has to create a new model. Examples for such a creation task are:

- The creation of a variability model for a product line including the definition of new variation points, variants, and constraints
- The creation of product line models including the definition of common and variable parts of these models by defining variability relationships between the variability model and elements of the product line model

In a modification task, the participant has to modify a given model based on a given list of modifications (e.g. specified in natural language). Examples for such a modification task are:

- The modification of existing variability models including the redefinition of constraints, or the renaming of variation points or variants
- The modification of existing product line models including the definition of new common are variable model elements, or the modification of a the variability quality of a model element, i.e., a common element becomes variable and vice versa.

In an analytic task, the participant has to analyze a given variability model and/or a given product line model. Examples for such an analytic task are:

- The participant has to check if a set of given variants can be selected together or not.
- The participant has to check for contradictions in the variability model, e.g. two variants require each other and exclude each other at the same time.
- The participant has to check if variable elements of a product line model can be selected together or not.

These tasks will be performed several times with different participants (e.g. student, researchers, engineers from industry). Each participant group is split in two subgroups. They are using two different versions of our tool prototype. Version a) contains the variability modeling editor including the previously described extensions. Version b) contains the editor without the extension.

We plan to measure the time that was necessary to complete each task. We expect to show that our extensions will reduce the execution time for each task thereby increase the speed of the engineer.

After the task execution, we plan to analyze the quality of task results, especially the quality of the analytic task. We expect to show that our extensions improve the quality of the task results.

Finally, we plan to compare the time and quality results for each task and expect to show that our extension improve the speed and the quality at the same time.

6.2. Eye Tracking Studies

Eye tracking technologies allow for the measurement of human eye movement, e.g. during the process of reading a text or a diagram. The result of an eye tracking study is typically the path of the human eye movement and the time spent for watching a certain part of a text or a diagram. This information can be used to draw conclusions about the accuracy and ease of a visual task.

We plan to evaluate our solution ideas for visual variability modeling, in particular the mountain view and the lasso view with eye tracking studies based on various tasks related to the variability models. These tasks can be reused from the experiments presented in Section 6.1. Similar to the experiments presented in Section 6.1, these tasks will be executed with both versions of our tool (with and without extensions). The execution of each task is document with an eye tracking tool. Additionally, the mouse movements and the keyboard interaction will be measured.

We expect to show that our extensions will significantly reduce the eye movement of the participants and thereby significantly reduce the cognitive complexity in terms of accuracy and ease for the creation and modification of large variability models.

We further plan to perform similar forms of eye tracking studies to evaluate the cognitive complexity for the other task.

7. Summary and Outlook

We have presented eight challenges for the visual modeling of complex product line models based on the orthogonal variability modeling approach. In order to address these challenges, we have presented different solution ideas that support the visual modeling of product lines and illustrated their implementation in our prototypical tool environment.

We consider our solution ideas as an important step towards an effective visual modeling in product line engineering. Our first experiences from the application of the tool prototype indicated that our solution ideas greatly improve the visual modeling of product lines. In our future work, we plan to examine the benefits of our solution ideas in more detail and plan to conduct a series of experiments and case studies. A description of our empirical evaluation strategy is also part of the paper.

8. References

[1] Bachmann, F.; Goedicke, M.; Leite, J.; Pohl, K.; Ramesh, B.; Vilbig, A.: Managing Variability in Product Family Development. In: Proceedings of 5th Intl. Workshop on Product Family Engineering (PFE-5), 2003.

[2] Bosch, J.: Software Variability Management. In: Proceedings of the 26th International Conference on Software Engineering, pp. 720-721, 2004.

[3] Classen, A.; Heymans, P.; Schobbens, P-Y.; Legay, A. and Raskin, J-F. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines (to appear). In 32nd International Conference on Software Engineering, ICSE 2010, 2010.

[4] Czarnecki, K.; Pietroszek, K.: Verifying feature-based model templates against well-formedness OCL constraints. In Proceedings of the Conference on Generative Programming and Component Engineering, pp. 211-220, 2006.

[5] Deelstra, S., Sinnema, M., Bosch, J.: A Product Derivation Framework for Software Product Families. In: van der Linden, Frank (ed.): Proceedings of the 5th International Workshop on Product Family Engineering, Siena, Italy, LNCS, Vol. 3014. Springer, Heidelberg, pp. 473-484, 2003.

[6] Harel, D.: On Visual Formalisms. Communications of the ACM, Vol. 31, No. 5, pp. 514-530, 1988.

[7] Heidenreich F., Savga I., Wende C.: On Controlled Visualisations in Software Product Line Engineering. In Proc. SPLC Workshop on Visualization in Software Product Line Engineering (ViSPLE), 2008.

[8] http://www.visuresolutions.com/

[9] Kästner C., Trujillo S., Apel S.: Visualizing Software Product Line Variabilities in Source Code. In Proc. SPLC Workshop on Visualization in Software Product Line Engineering (ViSPLE), 2008.

[10] Larkin, J.; Simon, H.A.: Why a Diagram Is (Sometimes) Worth Ten Thousand Words. Cognitive Science, Vol. 11, No. 1, 1987, pp. 65-100.

[11] Lauenroth, K.; Pohl, K.; Töhning, S.: Model Checking of Domain Artifacts in Product Line Engineering. In: Proc. of the ACM/IEEE Intl. Conference on Automated Software Engineering, pp. 269-280, 2009.

[12] Metzger, A.; Pohl, K.: Variability Management in Software Product Line Engineering. In: Knight, J.; Emmerich, W.; Rothermel, G. (Eds.): Proceedings of the 29th Intl. Conference on Software Engineering (ICSE 2007), Minneapolis, 20-26 Mai 2007, Companion Volume, ACM, pp. 186–187, 2007.

[13] Moody, D.: The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering, Vol. 35, No. 6, pp. 756-779, 2009.

[14] Pohl, K.; Böckle, G.; van der Linden, F.: Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Heidelberg, 2005.

[15] Sinnema, M.; Deelstra, S.; Nijhuis, J.; Bosch, J.: Managing Variability in Software Product Families. In: Proceedings of the 2nd Workshop on Software Variability Management, 2004.

[16] Stoiber R., Reinhard T., Glinz M.: Visualization Support for Software Product Line Modeling. In: Proceedings of the SPLC Workshop on Visualization in Software Product Line Engineering (ViSPLE), 2008.

[17] Variant Management with pure::variants, Pure-systems GmbH , Technical White Paper, 2006, URL: http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf