Kumbang Modeler: A Prototype Tool for Modeling Variability

Hanna Koivu, Mikko Raatikainen, Marko Nieminen, and Tomi Männistö

Helsinki University of Technology
Software Business and Engineering Institute (SoberIT)
P.O. Box 9210, 02015 TKK, Finland
{Hanna.Koivu, Mikko.Raatikainen, Marko.Nieminen, Tomi.Mannisto}@tkk.fi

Abstract. Variability is the ability of a system to be efficiently extended, changed, customized, or configured for use in a particular context. Several methods of modeling variability have been reported. However, tool support is also needed to take full advantage of models. We describe a prototype tool called Kumbang Modeler. Kumbang Modeler enables modeling the variability of a software product family using Kumbang conceptualization and language. The study follows the design science methodology. A user-centered design was applied in the development of Kumbang Modeler, and light-weight usability tests in the evaluation. The usability tests show that a person with knowledge of Kumbang concepts is able to correctly model the variability of a software product family using Kumbang Modeler.

1 Introduction

Variability is the ability of a system to be efficiently extended, changed, customized, or configured for use in a particular context [1]. As a successful means of managing variability, a *software product family* approach has emerged [2]. A software product family refers to a set of *product individuals* that reuse the same software assets and have a common structure called a software product family architecture [3]. The assets and the architecture of a software product family contain variability. This variability is resolved and, typically, product-specific software is also developed in order to derive the different product individuals of a software product family [4]. In a configurable software product family, the product individuals are constructed entirely on a basis of existing software only by resolving the variability [5,6].

A configurable software product family can contain a great amount of variability. In addition, this variability includes, for example, traceability relations and constraints [7]. Therefore, capturing and managing variability is challenging. Different variability modeling methods have emerged for variability management. When variability is expressed rigorously, such as in models with adequate rigor, product derivation can benefit from tool support. Tools can deduce consequences, check conformance to the model, and show when all variability is bound, for example [8]. Modeling also benefits from tools. Tools can hide the

details of the modeling language and tools can provide a viewpoint to model such that the model is only partially visible and easily navigatable. Hence, product expects can express variability without considering the syntax details of a particular language A modeling tool should also make it easier to demonstrate variability modeling to companies that could benefit from it, as well as making it more approachable.

In this paper, we describe Kumbang Modeler, which is a prototype modeling tool for Kumbang. Kumbang [9] is a domain ontology for modeling the variability in software product families. Kumbang includes concepts for modeling variability, from the point of view of the component and feature structures. The language that utilizes Kumbang conceptualization is likewise called Kumbang. The modeling tool is an Eclipse plug-in [10] for creating and editing Kumbang models. The tool stores models using the Kumbang language. In the study, we followed the design science methodology [11]. For developing the tool, we applied a user-centered design, and, in particular, the personas method [12,13]. The tool was tested for feasibility and in two lightweight usability tests.

The rest of the paper is organized as follows. In Section 2, we describe the research methods. In Section 3, we give an overview of Kumbang. Section 4 introduces Kumbang Modeler. In Section 5, we describe the validation of Kumbang Modeler. Section 6 discusses our experiences in developing Kumbang Modeler. Section 7 outlines related work. Section 8 draws conclusions and provides some directions for future work.

2 Method

The research followed the design science methodology [11]. The construction is a prototype Kumbang Modeler. The objective of the prototype was to provide the user with the ability to be able to create and edit Kumbang models through a graphical user interface. The models made using Kumbang Modeler should be saved to text files using the Kumbang language. The study built on existing Kumbang concepts, without changing them.

The development of Kumbang Modeler followed a user-centered design. The methods used were *Goal-Directed Design*, and especially *Personas* [12,13]. However, users and context, which are central in user-centered design [14], could not be studied in practice, e.g., in a case study. This was because Kumbang Modeler is a new kind of product and not used anywhere. Therefore, different use scenarios and characteristics of potential users were explored, mainly on the basis of results reported in the literature. The objective was to achieve a better understanding of the effective use of a modeling tool in an industrial setting, and the skill requirements for users. In addition, the feasibility of a user-centered design without actual users was assessed.

Kumbang Modeler was tested for feasibility and in lightweight usability tests. In the feasibility test, different models were developed and their correctness was validated. The lightweight formative usability tests were carried out with two different users. Both users had experience with software product families. The

users had not used or even seen Kumbang Modeler before the test. One user did not have knowledge of Kumbang, while the other was familiar with Kumbang concepts and language. The tests took roughly an hour and were done in an office room with a PC. Both tests were recorded with a video camera and screen capture software. The users were interviewed before and after the tests.

3 Kumbang Overview

Kumbang [9] is a domain ontology for modeling variability in software product families. Kumbang differentiates between a *configuration model*, which describes a family and contains variability, and a *product description*, which is a model of a product individual derived from a configuration model by resolving variability. The elements in a configuration model are referred to as *types*, while the elements in a product description are referred to as *instances*.

Kumbang includes concepts for modeling variability from both a structural and feature point of view. More specifically, the modeling concepts include *components* and *features* with *inheritance structure* and *compositional structure*, *attributes*, the *interfaces* of components and *connections* between these, and *constraints*. A compositional structure is achieved through the concepts of a *subfeature definition* and *part definition* that state what kinds of parts can exists for a feature or component, respectively. Constraints can be specified within components and features. Implementation constraints are a special class of constraints between features and components.

The semantics of Kumbang is rigorously described using natural language and a UML profile. A language based on the Kumbang ontology, likewise called Kumbang, has been defined. Kumbang has been provided with formal semantics by defining and implementing a translation from Kumbang to WCRL (Weight Constraint Rule Language) [15], a general-purpose knowledge representation language with formal semantics.

A tool called Kumbang Configurator, which supports product derivation for software product families modeled using Kumbang, has been implemented [8]. Kumbang Configurator supports a user in the configuration task as follows: Kumbang Configurator reads a Kumbang model and represents the variability in the model in a graphical user interface. The user resolves the variability by entering her requirements for the product individual: for example, the user may decide whether to include an optional element in the configuration or not, to select attribute values or the type of a given part, or create a connection between interfaces. After each requirement entered by the user, the Kumbang Configurator checks the consistency of the configuration, i.e., whether the requirements entered so far are mutually compatible, and deduces the consequences of the requirements entered so far, e.g., automatically choosing an alternative that has been constrained down to one; the consequences are reflected in the user interface. The consistency checks and deductions are performed using an inference engine called smodels [15] based on the WCRL program translated from the model. Once all the variation points have been resolved and a valid configura-

tion thus found, the tool is able to export the configuration, which can act as an input for tools used to implement the software, or used for other purposes.

4 Kumbang Modeler

This section introduces Kumbang Modeler, a prototype tool for creating Kumbang models. Kumbang Modeler has been implemented as a plug-in for the Eclipse Platform [10]. First, a short introduction to Eclipse will be given below, then Kumbang Modeler is described in terms of architecture, user interface, and usage.

4.1 Eclipse

Eclipse [10] is an integrated development environment popular among Java developers [16]. Eclipse began as an open source IDE tool for Java development, but has been extended to a multi-purpose development environment via plug-in extensions. Eclipse plug-ins are currently very popular [17].

Eclipse's development environment is called a *workbench* [18]. A user sees a workbench as one or several windows. Each window contains a menu bar, a toolbar, and one or more *perspectives*. A perspective defines what is included in the menus and toolbar. The perspective also defines a default layout, which can be changed or reloaded to undo changes. A perspective is also a container that defines the initial group and layout of a group of *editors* and *views*. An editor or a view contains the actual user interface elements. Plug-ins can consist of any of the element such as views, editors, menus, or perspectives.

4.2 Kumbang Modeler Architecture

The main elements of the Kumbang Modeler architecture are the model layer, the controllers layer, and the user interface layer. A Kumbang model is represented as Java objects at the model layer and can be imported from or exported to a text file. The controllers layer combines some display information with he model objects, provides ways to change the model and updates these changes at the graphical user interface layer. The user interface elements are at the graphical user interface layer. Kumbang Modeler reuses the model and parser from Kumbang Configurator [8].

4.3 Kumbang Modeler User Interface

User interface elements specific to Kumbang Modeler are a perspective, an editor, and three views. In addition, Kumbang Modeler uses two standard Eclipse views. The user interface design was guided by the Eclipse User Interface Guidelines [19].

The perspective for Kumbang Modeler comprises six different areas, depicted by letters a-f in Figure 1. The perspective is automatically opened when a file containing a Kumbang model is opened or a new Kumbang model is created.

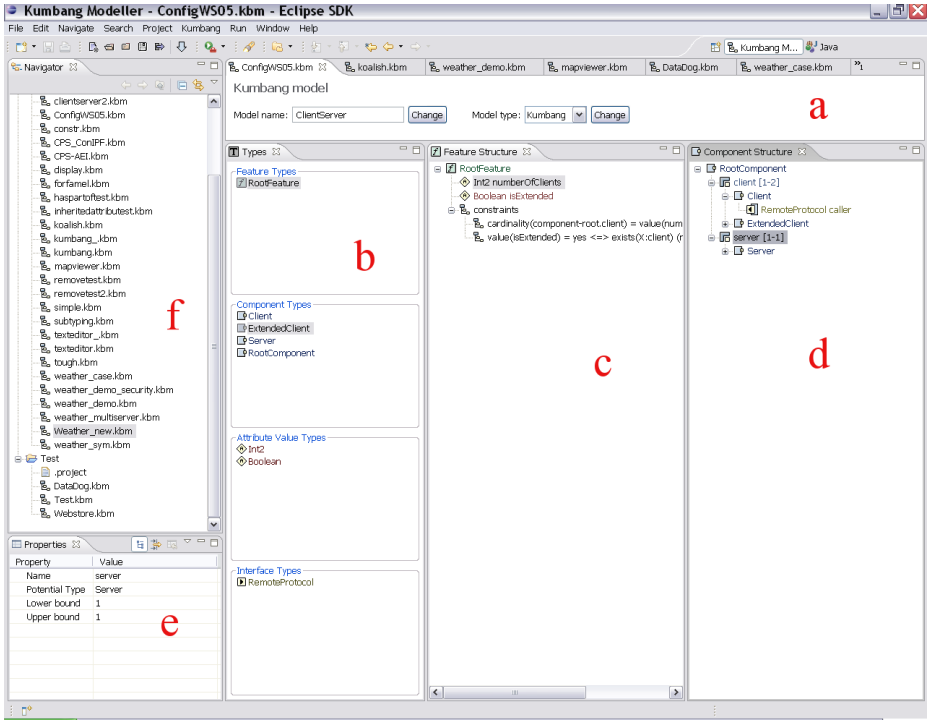
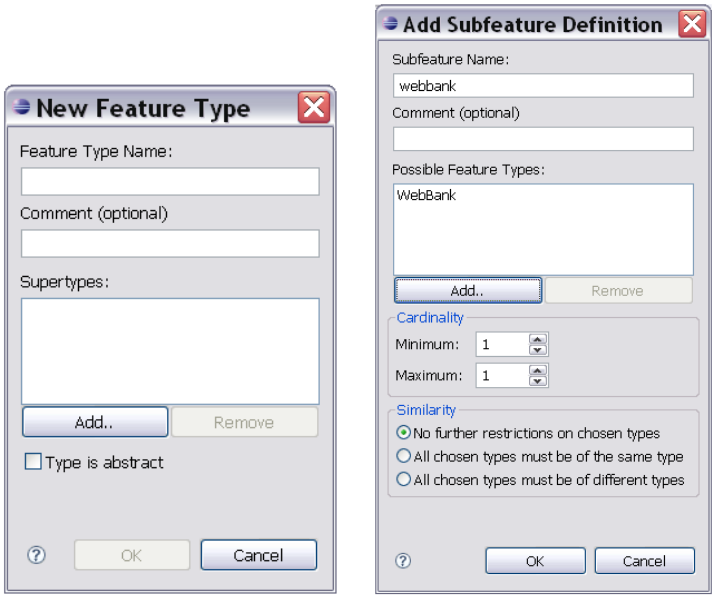


Fig. 1: The perspective for Kumbang Modeler: a) the editor area b) the type view c) the feature view d) the component view e) the properties view f) the navigator view

The editor area (Figure 1, a) shows the currently active editor and enables switching between open editors by selecting the model from the tabs. The editor area displays the model name and type; all other information is shown in the views. This gives the user more control over the user interface, as views can be resized, moved, and closed freely. The editor takes care of opening and saving the model. Several editors can be open at the same time.

The three views peculiar to Kumbang are *a type view*, *a feature view* and *a component view* (Figure 1 b, c and d, respectively). The type view lists currently available types. Kumbang has feature, component, interface, and attribute value types. The feature and component views show the compositional hierarchies. The hierarchies form trees with one root. The tree is composed using the sub-feature definitions within the features types and the part definitions within the components types. In addition, constraints are added to the feature and component types in the feature and component views. Implementation constraints between the feature hierarchy and the component hierarchy are placed in the feature view.

The *properties* and *navigator views* are standard views in the Eclipse IDE. The properties view (Figure 1, e) shows additional information on the currently



(a) (b)

Fig. 2: Two Kumbang Modeler dialogs: a) a dialog for adding a new feature type; b) a dialog for a subfeature definition.

selected elements. The navigator view (Figure 1, f) shows Eclipse projects and files.

Finally, Kumbang Modeler contains several dialogs that are needed when editing a model. The archetypes of the dialogs are shown below in the section describing the usage of Kumbang Modeler.

4.4 Usage

The available types, such as different feature and component types, of a model are created using dialogs. Figure 2(a) shows a dialog for creating a new feature type; similar dialogs are used for other types, although the exact fields are peculiar for the respective type. The dialogs are needed in order to set all details of the specific type. For example, a new feature type needs a name, possible supertypes, specification if the type is abstract, and an optional comment. The same dialogs that are used for creating new types are used to display and change existing types. Right-clicking in any of the views peculiar to Kumbang opens a menu from which an option for the dialog for creating new types can be selected. Alternatively, the dialog can be opened and new types created while defining compositional structures other than root, as described below.

The compositional structure of the features and components needs to proceed from the root to the leafs. When a type is dragged from the type view to an empty structural view of that type, the type becomes the root; or, when there is no root set, an existing type can be selected to be the root from a list of all possible types. Consequently, only existing type can be selected to be a root.

When the root is set, the other types can be added to the compositional structure through their respective definitions. A type can be added to the structure by dragging. Alternatively, an existing type can be selected or a new type can be created for the compositional structure during the construction of the definition. Attributes and interfaces are attached to the structure using the concept of definition, similar to the way the compositional structure is constructed. A dialog is always needed to determine the necessary information when adding a type to a structure. Figure 2(b) shows an example dialog for a subfeature definition. If a type is dragged to the tree, those values that have feasible default values are pre-filled. For instance, the cardinality of definitions has a default value of one-to-one, and the name is derived from the type, but the direction of an interface definition has no sensible default value.

The constraint language of Kumbang [20] combines predicates with Boolean algebra. A constraint can be very complex; hence, there is no simple way to manage them. In addition, flexibility is required in constructing the constraints, since length cannot be determined beforehand. The approach taken in Kumbang Modeler splits constraints into parts that can be constructed separately. There are two kinds of basic parts: *expressions*, which are predicates or functions, and *operators*, which combine the expressions or implicate relationships between the expressions. These parts are shown in a list, which is expanded every time a part is added, as seen in Figure 3(a). When a new expression is added, an expression type must first be chosen (Figure 3(b)). A similar dialog is also used for choosing operators. Each expression has a special dialog for defining the details.

5 Validation of Kumbang Modeler

5.1 Feasibility to Produce Valid Models

We tested whether Kumbang Modeler is able to produce valid models based on Kumbang concepts and written in the Kumbang language. The models were syntactically correct and could be opened also in Kumbang Configurator. In addition, Kumbang Modeler can be used to open and modify various existing Kumbang test models.

5.2 Usability Evaluation of the Prototype

Kumbang Modeler was tested through lightweight formative usability tests with two different users by a predefined modeling task defined as a scenario. The first user, who knew Kumbang very well, had very little trouble making a model according to the scenario. She did have some suggestions for improving the user

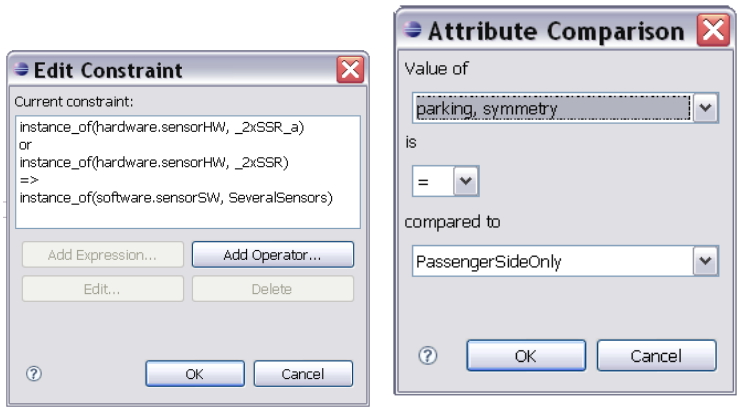


Fig. 3: A dialog for editing a constraint and a dialog for choosing an expression to a constraint.

interface, however. Most of these proposals were implemented before the second usability test. The second user had no previous knowledge of Kumbang before participating in the test. He had trouble understanding the need for relation between types and definitions used for compositional structure. This made him very frustrated; when he wanted to add a feature to the model, he did not understand why he had to make both a feature type and a subfeature definition to achieve this. However, he was able to produce an acceptable model. Table 1 summarizes the main usability changes made to Kumbang Modeler on the basis of the usability tests.

Problem	**Action taken**
The icon used in an interface definition did not show whether the interface was provided or required.	Instead of a single interface icon, different icons was designed for a required and provided interface.
Subfeature definitions could not be moved in the feature structure.	Subfeature definitions were made draggable.
User was irritated with having to name both types and definitions.	Lower case version of the type name was made to be the default definition name.
User was irritated with having to do too many steps when wanting to make a definition with a type that did not exist yet.	Several steps were combined into one.

Table 1: Changes made after the feedback from the usability tests

6 Discussion

6.1 Feasibility of Tool Support for Variability Modeling

Kumbang Modeler seems feasible for modeling variability in software product families. In addition, Kumbang Modeler seems to have advantages over writing a model by a text editor. For example, the produced models are syntactically correct, and the tree structure seems to make navigating within the hierarchy easier.

Kumbang Modeler was developed as a plug-in for Eclipse, which seems to be a practically applicable platform for such a modeling tool as Eclipse is currently a popular development environment. In addition, different plug-in extensions are also relatively widely used and easy to install. Many developers are thus familiar with Eclipse as a development environment, and with plug-in extensions for Eclipse.

6.2 Validation of Kumbang Modeler

Two tests for Kumbang Modeler were carried out: the test of the validity of the models produced and the usability tests. However, these tests have weaknesses. First, Kumbang Modeler does not check anything other than the syntactical correctness of a produced model. However, we are currently implementing advanced checks for Kumbang Modeler. The advanced checks ensure, for example, that a model does not contain cycles in the inheritance, part, or subfeature structures; a model contains all the references needed, such as the type declarations for the types used in a part definition, and at least one configuration can be found such that all interfaces can be connected; for every required interface, a provided interface exists; and constraints are not in conflict with each other. Second, the usability tests were lightweight and were not carried out using a real product. Hence, more usability tests are required in industrial settings.

6.3 Feasibility to use Kumbang Modeler to Model Variability

Kumbang Modeler seems to be feasible for product experts to express the variability of a software product family as Kumbang models. The user who knew Kumbang concepts was able to use the tool without difficulties. The user seems to be, however, required to have some knowledge of Kumbang and software product families, as the second usability test showed. Nevertheless, a thorough understanding of Kumbang syntax and semantics did not seem to be needed. The requirement of understanding Kumbang concepts is not necessarily a problem, since the tool is meant for highly specialized use. However, more tests are needed, as argued above.

The difficulties the other user had with producing a model using Kumbang Modeler seemed to be more related to Kumbang concepts than Kumbang Modeler as a tool. The user had no previous knowledge of Kumbang and was, in fact, used to modeling software product families differently. Especially troublesome

were those concepts that are not widely used in other modeling approaches. Three issues especially caused difficulties: the type and instance differentiation, the part and subfeature definitions in the compositional structure, and terminology. These are discussed in more depth below.

First, many feature modeling methods, for example, do not differentiate between types and instances. However, in Kumbang they are used in order to distinguish between a family model that contains variability and an instance model in which variability is resolved, enable several instances of the same type, and enable feature type reuse in different places of the model [21]. In addition, software product family engineering distinguishes between family and instance, e.g., in a form of family and instance development processes, or reusable and reused assets. Consequently, differentiating between types and instances seems reasonable.

Second, despite the fact that subfeatures and the compositional structure of components are used in most modeling methods, the subfeature and part definitions, which are slightly different in Kumbang, caused difficulties. The subfeature and part definitions are regarded as required in Kumbang [21]. However, especially in simple structures, such as in a subfeature structure without variability, it seems that in many cases, default values can be used for the details of the subfeature definitions. The user interface was simplified, e.g., by making a lowercase version of the type name the default name of the part definition. Although the subfeature definitions, for example, can be simplified in the user interface by using default values, they are still needed in order to express complex variability.

Third, the terminology of Kumbang Modeler was confusing. However, in software product family engineering the same term is often used to refer to different concepts or several terms are used to refer the same concept. For example, feature modeling methods do not terminologically distinguish between feature types and feature instances or product derivation can be also called instantiation, deployment or configuring. Hence, the terminology in general is ambiguous and not established. A person used to one terminology can get confused when she needs to use another terminology.

6.4 Variability Modeling

Issues concerning the nature of variability arose during the study. For example, Kumbang uses constructs that can be used to model complex variability. However, much of the variability in usability tests was so simple that using Kumbang constructs meant inserting information that was laborious, and default values would have been feasible. In order to enhance tool support, the nature of variability in software in terms of, for example, the amount and complexity of variability needs to be studied in more depth. This could then be used to develop tools that meet the actual requirements for modeling variability. For example, syntactic sugar on top of modeling concepts could be developed in order to hide complex structures. However, the rigor of the models should not be lost. The models should be based on a well-founded conceptual foundation.

6.5 User-Centered Design

We faced problems with the user-centered design approach during the development of Kumbang Modeler. The users of Kumbang Modeler do not exist and, hence, cannot be studied. We tried to study the literature in order to capture, e.g., the skills of potential users, but little is reported in the literature. Another option would have been to carry out a user study of software product family engineers in general, but this was considered to be beyond the scope of this study.

Goal-Directed Design considers necessary-use scenarios to be less important than daily-use scenarios. However, Kumbang Modeler is mainly a prototype tool and thus the threshold for using it for modeling should be low. Therefore, the creation of new models is just as important in Kumbang Modeler as modifying existing ones, although only modifying a model can be considered a daily use scenario. Therefore, Goal-Directed Design was not directly applicable in Kumbang Modeler design.

The usability tests brought about the same problem as with the overall development of the tool, namely a lack of real users who would use the tool in an actual, industrial environment. However, we assumed that such users could have three kinds of knowledge: knowledge of the specific configurable product family, configurable product family concepts, and modeling concepts. Since configurable product families are hard to find, in usability tests we used two kinds of user: both had knowledge of configurable product family concepts and one knew Kumbang.

7 Related Work

Several software variability modeling methods have been developed in addition to Kumbang, such as feature modeling [22], orthogonal variability modeling [23], and COVAMOF [24]. Different kinds of tools have been developed for the modeling methods; a review of a set of tools is provided in the ConIPF methodology [25].

In addition, there are variability modeling tools that are not used for software products. Instead, the tools are originally meant for modeling mechanical and electronic products. Examples of such tools are the Wecotin [26] and EngCon [27] modelers.

Tools can be also used in other phases of the software life cycle. In ConIPF methodology [25], tools are used for requirements engineering, modeling, configuring, realization, and software configuration management. Different tools can be used in different phases. Kumbang currently has tool support for the modeling and configuration phases. Different tools or new tools for Kumbang need to be developed for the other phases of development.

8 Conclusions

In this paper we described Kumbang Modeler, which is a tool for modeling the variability of a software product family. Modeling is based on Kumbang conceptualization. Hence, Kumbang Modeler enables modeling both from a structural and feature point of view. The study followed design science methodology. We applied a user-centered design in developing Kumbang Modeler; more specifically, the Goal-Directed Design and Personas methods. Kumbang Modeler was tested for feasibility to produce correct models and in lightweight formative usability tests.

The results showed the feasibility of modeling variability with Kumbang Modeler. At least some knowledge on the applied Kumbang variability concepts is required to use the tool. We faced problems with the user-centered design because actual users were not available. The usability tests, nevertheless, showed that despite the fact that variability can be modeled with the existing methods, more studies are needed to show that modeling is efficient and convenient. For example, much of the variability can be simple and details of more complex constructs to model variability can then be hidden or default values can be used. However, modeling also seems to need complex structures. In addition, in order for, e.g., tool-supported derivation to be possible, modeling should be based on rigorous foundations.

Kumbang Modeler provides the missing tools support for Kumbang. That is, with Kumbang Modeler, the captured variability of a software product family can be modeled, whereas with the existing Kumbang Configurator, expressed variability can be bound during product derivation. Hence, other tools, such as a generator, are still needed.

Acknowledgements

The authors acknowledge the financial support of Tekes, the Finnish Funding Agency for Technology and Innovation.

References

1. Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Software — Practice and Experience **35** (2000)
2. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
3. Clements, P., Northrop, L.M.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)
4. Bosch, J.: Design and Use of Software Architecture. Addison-Wesley (2000)
5. Bosch, J.: Maturity and evolution in software product line: Approaches, artefacts and organization. Lecture Notes in Computer Science (Proc. of SPLC2) **2379** (2002) 257–271

6. Männistö, T., Soininen, T., Sulonen, R.: Configurable software product families. In: ECAI 2000 Configuration Workshop, Berlin. (2000)
7. Thiel, S., Hein, A.: Modeling and using product line variability in automotive systems. IEEE Software **19**(4) (2002)
8. Myllärniemi, V., Asikainen, T., Männistö, T., Soininen, T.: Kumbang configurator—a configuration tool for software product families. In: IJCAI-05 Workshop on Configuration. (2005)
9. Asikainen, T., Männistö, T., Soininen, T.: Kumbang: A domain ontology for modelling variability in software product families. Advanced Engineering Informatics **21**(1) (2007)
10. Eclipse Foundation: Eclipse platform. http://www.eclipse.org/ (2006) Visited December 2006.
11. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly **28**(1) (2004)
12. Cooper, A.: The Inmates Are Running the Asylum. Macmillan Publishing Co. Inc. (1999)
13. Cooper, A., Reimann, R.: About Face 2.0: The Essentials of Interaction Design. John Wiley & Sons, Inc. (2003)
14. ISO/IEC: 9241-11 ergonomic requirements for office work with visual display terminals (vdt)s - part 11: Guidance on usability (1998)
15. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002)
16. Goth, G.: Beware the march of this ide: Eclipse is overshadowing other tool technologies. IEEE Software **22**(4) (2005) 108–111
17. Murphy, G.C., Kersten, M., Findlater, L.: How are java software developers using the eclipse ide? IEEE Software **23**(4) (2006) 76–83
18. Eclipse 3.2 Documentation. http://help.eclipse.org/help32/index.jsp (2006) Visited December 2006.
19. Edgar, N., Haaland, K., Li, J., Peter, K.: Eclipse user interface guidelines, v. 2.1 (2004) http://www.eclipse.org/articles/Article-UI-Guidelines/Index.html. Visited December 2006.
20. Asikainen, T.: Kumbang language, technical report (2007, to appear)
21. Asikainen, T., Männistö, T., Soininen, T.: A unified conceptual foundation for feature modelling. In: Proceedings of the 10th International Software Product Line Conference. (2006)
22. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute (1990)
23. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
24. Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J.: Covamof: A framework for modeling variability in software product families. In: Proceedings of Software Product Line Conference (SPLC). (2004) 197–213
25. Hotz, L., Wolter, K., Krebs, T., Deelstra, S., Sinnema, M., Nijhuis, J., MacGregor, J.: Configuration in Industrial Product Families. IOS Press (2006)
26. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A practical tool for mass-customising configurable products. In: In Proceedings of International Conference on Engineering Design (ICED 03). (2003)
27. Hollmann, O., Wagner, T., Guenter, A.: Engcon - a flexible domain-independent configuration engine. In: Configuration Workshop at ECAI-2000. (2000)