Methods for Modelling the Variability in Software Product Families[1]

Timo Asikainen

Helsinki University of Technology, Software Business and Engineering Institute,
P.O. Box 9210, FI-02015 TKK, Finland
timo.asikainen@tkk.fi

Abstract. Variability is the ability of a system to be efficiently extended, changed, customised or configured for use in a particular context. There is an ever-growing demand for variability of software. Software product families are an important means for implementing software variability. A software product family may contain very large numbers of individual products. Consequently, methods for representing the variability and efficiently reasoning about it are needed. This thesis studies such methods: the goal of the thesis is to define a solid conceptual basis for modelling the variability in software product families, and to provide the concepts formal semantics in such a way that reasoning on the models is possible using existing inference tools. Major parts of the work have already been completed and documented in a number of publications.

1 Introduction

Variability is the ability of a system to be efficiently extended, changed, customised or configured for use in a particular context [1]. There is a growing demand for variability of software, and a significant research interest in the topic, as exemplified by the workshops and special issues devoted to it, see, e.g., [2]. Products that incorporate variability are useful for various purposes: for example, such products can address multiple user segments, allow price categorisation, support various hardware platforms and operating systems, offer different sets of features for different needs, and cover different market areas with different languages, legislation, and market structure. Addressing these concerns without variability would be very difficult, if not impossible.

Software product families, or *software product lines*, as they are also called, have become an important means for implementing variability [3]. A software product family may contain very large numbers of individual products. Consequently, methods for representing the variability and efficiently reasoning about it are needed.

This thesis aims at developing methods for modelling the variability in software product families. The most important modelling concepts and constructs in the methods developed stem from existing methods used for representing the variability in

[1] PhD work, 4th year

software product families. However, the methods introduced in the thesis are provided with a more solid conceptual foundation and richer sets of modelling concepts.

The remainder of this position paper is structured as follows. Next, in Section 2 we provide a brief overview of the related previous work. The research problem is defined in Section 3, along with research questions and goals. Thereafter, in Section 4 we discuss the results achieved so far. An outline for further work follows in Section 5.

2 Previous work

This section provides an overview of the previous work on modelling variability in software product families and identifies an area of research in which more work is needed.

Numerous methods for modelling the variability in software product families have been proposed. In general terms, a *decision model* specifies the decisions that must be made to produce an individual product in the family and the order of these decisions [4]. Such decisions are often termed *variation points*.

A practically important class of variability modelling methods is based on modelling the common and variable *features* of a product family. An example of such a method is FODA (Feature Oriented Domain Analysis) [5]. A number of methods for modelling variability in *product family architectures* have been reported; Koalish [6] and xADL 2.0 [7] are examples of such methods. Arguably, variability models based on features or architecture can be considered to be instances of decision models: both of these span a set of decisions that must made in order to produce an individual product in the family. *Domain-specific languages* may also be used to express variability in software product families [8].

Variability has also been studied in the domain of traditional products, i.e. mechanical and electrical ones. This domain of research is called *product configuration*, or *configuration* for short, and it studies how a general design of a product can be modified in prescribed ways to produce product individuals that match the specific needs of customers [9]. In contrast to methods for modelling variability in software product families, the results achieved in product configuration domain include a number of conceptualisations of the domain [10, 11]. The conceptual work done in the domain has lead to a large number of successful applications [9, 12, 13].

Although there are a relatively large number of studies on variability of software, there is still need for further research on the topic. The conceptual foundation of the modelling methods is in many cases unclear: in many methods, the concepts and their interrelations are not defined at all, or in an unsatisfactory manner; conceptual work similar to that done in the product configuration domain could alleviate this condition. The semantics of the modelling concepts is in most cases not rigorously defined. Many practically relevant aspects have not been studied in depth. Configuration of individual systems over multiple stages [14] or *binding times* is widely acknowledged to be an important topic. Yet most existing methods for modelling variability do not account for multiple binding times. *Constraint languages* used in expressing dependencies between different decisions or variation points are either simplistic, including

only constraints of the form *A requires B* and *A excludes B*, or described cursorily, e.g., by referring to existing constraint languages, such as OCL [15], without studying the applicability of these languages to variability modelling in any detail. Also, we do not know any feature modelling methods that would account for the *evolution*, i.e., changes over time, of variability models.

3 Research Problem, Questions, and Method

The research problem is the *study and development of methods for modelling the variability and commonality in software product families*. In more detail, the thesis aims at answering the following research questions.

1. What concepts are suited to modelling variability and commonality in software product families?
2. What is the formal or rigorous semantics of these concepts?
3. What kind of languages can be built on these concepts?
4. What kind of tools can support the use of these languages and methods?

Related to the fourth point, there should be support for two tasks: the *modelling* and the *configuration task*. The former pertains to creating a model of the variability in a software product family. The latter, in turn, pertains to finding a *configuration*, i.e., a description of an individual product in the family, matching a given set of requirements at hand.

The research method applied in this thesis is a *constructive* one [16]. In short, applying the constructive research method pertains to building an artefact that solves a domain problem in order to create knowledge about how the problem can be solved and the solution artefact compares with previous solutions to the same problem.

4 Results achieved

In this section, we provide an overview of the results achieved so far.

The results achieved so far can be classified based on the underlying modelling concepts; a distinction between results on *feature modelling*, *architecture description*, and results *integrating* these two views can be made.

Forfamel is a method for modelling the variability in software product families from a feature point of view. The conceptual basis of Forfamel is defined in [17]. Forfamel includes the definition of the concepts of the method, and their informal but rigorous semantics. Forfamel synthesises a number of existing feature modelling methods, which gives it a solid foundation. Further, it previous work on features with a number of concepts and constructs from the product configuration domain. Forfamel is provided with formal semantics by translating it to Weight Constraint Rule Language (WCRL) [18], a general purpose knowledge-representation language similar to logic programs [19]. Although general-purpose, WCRL has been designed to allow the easy representation of configuration knowledge about non-software products and shown to suit this purpose [20]. This suggests that WCRL is a reasonable

choice for the knowledge representation formalism of our approach as well. Further, an inference system *smodels*[2] operating on WCRL has been shown to have a very competitive performance compared to other problem solvers, especially in the case of problems including structure [18].

Further, [21] shows how an existing prototype product configurator, WeCoTin [22], can be used to provide tool support for modelling and configuring the features of a software product family; it should be noticed that the feature modelling concepts studied in this paper are not those of Forfamel, but another synthesis from previous feature modelling methods. The configurator provides support for both the *modelling* and *configuration task*. The paper shows that existing tools, originally intended for describing the physical structure of non-software products, can be applied to software products.

As for architecture description, [23] contains an analysis of three architecture description languages (ADLs), and compares them with a configuration ontology originally developed for non-software products [10]. The outcome is that the ontology is able to capture most, but not all of the concepts of the ADLs. Hence, [23] shows that configuration modelling concepts provide a basis on which architecture-based modelling methods for configurable software product families can be built on, but is not as such applicable to modelling architectures.

Further, [24] contains the definitions of a conceptualisation, i.e., a domain ontology, called *Koalish* for modelling architecture of configurable software product families. In more detail, Koalish is based on Koala [25], a component model and architecture description language (ADL), developed at Philips Consumer Electronics. Koala is, to the best of the author's knowledge, the only ADL that has been applied in the industry. Hence, its practical success gives Koalish a solid foundation. Koalish extends Koala with concepts and constructs for modelling variability. Finally, similarly as for Forfamel, Koalish is provided with formal semantics by translating it to WCRL.

The definition of a language based on Koalish is contained in [6]; this language is likewise called Koalish. In addition, an approach for managing configurable software product families is outlined. The approach is based on providing tool support for the modelling and deployment tasks. The modelling task was defined in Section 3 above. The deployment task, in turn, consists of the configuration task and the additional steps required to turn the description of an individual product into a concrete product. Together, the language and the outlined process form a solid basis on which tools supporting architecture-based configurable software product families.

Kumbang [26] is an approach integrating Forfamel and Koalish. Thus, Kumbang enables modelling variability simultaneously from a feature and an architecture point of view and the interrelations between these two views using constraints. The work includes a UML (Unified Modeling Language) stereotype illustrating the modelling concepts of Kumbang and those of UML. A number of case products inspired by real-life software product families have been modelled using Kumbang by our research team. Kumbang provides a sufficient level of support to capture the intent of

[2] See `http://www.tcs.hut.fi/Software/smodels/`

the product families. The cognitive effort required to create the models has been moderate.

Finally, Kumbang Configurator is a prototype tool that supports the configuration task for Kumbang, and hence also Forfamel and Koalish, models [27]. The configurator includes an implementation of the Kumbang. The configurator has performed well when applied to the case software product families mentioned in the previous paragraph.

5 Further work

This section discusses further work needed to complete the thesis. It is still unclear which extensions will be included in the thesis; it is unlikely that all of them would be included.

Further work should take place in three main areas. First, it is possible to extend the conceptual basis with new modelling concepts and constructs. Second, theoretical studies can be carried out to add rigour to the possibly extended modelling concepts. Finally, empirical studies can be carried out to demonstrate the practical applicability of Kumbang.

There are a number of possible ways to extend the conceptual basis of Kumbang. An essential extension is to define a constraint language to be used with Kumbang: constraints are needed to specify dependencies both within a single view and between views. Such a constraint language should resemble existing languages such as the Object Constraint Language (OCL) [15] or xPath (see http://www.w3.org/TR/xpath) and should be an integral part of the modelling method in the sense that it is both possible to check the constraints and efficiently search for a configuration that satisfies the constraints in the configuration model.

It is also possible to extend Kumbang with concepts and constructs for modelling the evolution of software product families, similarly as has been done in xADL 2.0 [7].

An issue often discussed in conjunction with variability are *binding times*: a configuration is not produced during a single step but during multiple steps where the output of the previous step serves as an input for the following steps [14]. However, the notion of binding times and their semantics has not yet been thoroughly studied or understood. Hence, augmenting the modelling methods developed in the thesis could both improve their usefulness and contribute to the area of research.

Another possibility is to extend the modelling concepts in such a way that the user would define the views used in a particular model. That is, the set of views in the modelling method would not be fixed to, e.g., a combination of feature and architectural views. Instead, the number of views, the properties of each view, and the possible interrelations between views could be specified to match the particular requirements of the domain at hand. This extension is motivated by the fact that the number and characteristics of the views required to model the variability in a software product family depends on the particular domain and family at hand. It seems that no single set of views suits all domains.

An example of a practical domain with more than two views is car periphery systems at Robert Bosch GmbH [28]. In this domain, four views are used: the environment in which the device is located, the features of the software, the architecture of the physical device in which the software is embedded in, and the architecture of the software itself and how it is deployed to physical components.

To make the theoretical foundation of Kumbang, or an extended method more solid, the method could be provided with even more rigorous formal semantics than has been done so far for Kumbang. Such semantics could also be used to perform theoretical complexity analysis and other relevant properties of the methods.

Demonstrating the practical applicability of the results requires testing the methods empirically with real software product families in real software development contexts. The tests should concern both their expressive power and usability. The same method, applied to a sufficiently wide range of different kinds of configurable software product families, can also be used to analyse their scope of applicability.

References

1. Svahnberg, M., van Gurp, J., Bosch, J.: A Taxonomy of Variability Realization Techniques. Software - Practice and Experience 35(8) (2006) 705-754
2. Bosch, J.: Software Variability Management (introduction to special issue on software variability management). Science of Computer Programming 53(5) (2004) 255-258
3. Clements, P. C. and Northrop, L.: Software Product Lines - Practices and Patterns. Addison-Wesley, Boston (MA) (2001)
4. Weiss, D. and Lai, C. T. R.: Software Product Line Engineering: A Family Based Software Development Process. Addison-Wesley, Boston (MA) (1999)
5. Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, S. A.: Feature-Oriented Domain Analysis (FODA) - Feasibility Study. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
6. Asikainen, T., Soininen, T., Männistö, T.: A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families. In: Proceedings of the 5th International Workshop on Product Family Engineering (PFE-5). Lecture Notes in Computer Science 3014. Springer (2003) 225-249
7. Dashofy, E., van der Hoek, A., Taylor, R. M.: A Comprehensive Approach for the Development of Modular Software Architecture Description Languages. ACM Transactions on Software Engineering and Methodology 14(2) (2005) 199-245
8. Tolvanen, J.-P., Kelly, S.: Defining Domain-Specific Modeling Language to Automate Product Derivation: Collected Experiences. In: Obbink, J. Henk and Pohl, Klaus (eds.): Proceedings of the 9th International Software Product Line Conference (SPLC 2005) (2005) 198-206
9. Soininen, T., Stumptner, M.: Introduction to Special Issue on Configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17(1-2) (2003) 1-2
10. Soininen, T., Tiihonen, J., Männistö, T., Sulonen, R.: Towards a General Ontology of Configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4) (1998) 357-372
11. Felfernig, A., Friedrich, G., Jannach, D.: Conceptual Modeling for Configuration of Mass-Customizable Products. Artificial Intelligence in Engineering 15(2) (2001) 165-176
12. Faltings, B., Freuder, E. C.: Special Issue on Configuration. IEEE Intelligent Systems 14(4) (1998) 29-85

13. Darr, T., Klein, M., McGuinness, D. L.: Special Issue on Configuration Design. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4) (1998) 293-397
14. Czarnecki, K., Helsen, S., Eisenecker, U. W.: Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practices 10(2) (2005) 143-169
15. Object Management Group: OCL 2.0 Specification. ptc/2005-06-06 (2005)
16. Kasanen, E., Lukka, K., Siitonen, A.: The Constructive Approach in Management Accounting Research. Journal of Management Accounting Research 5(1993) 243-264
17. Asikainen, T., Männistö, T., Soininen, T.: A Unified Conceptual Foundation for Feature Modelling. In: Proceedings of the 10th International Software Product Line Conference (SPLC 2006) (2006)
18. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. Artificial Intelligence 138(1-2) (2002) 181-234
19. Asikainen, T. Modelling Methods for Managing Variability of Configurable Software Product Families. Licentiate thesis, Helsinki University of Technology, Department of Computer Science and Engineering. (2004)
20. Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing Configuration Knowledge with Weight Constraint Rules. In: Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning (2001)
21. Asikainen, T., Männistö, T., Soininen, T.: Using a Configurator for Modelling and Configuring Software Product Lines Based on Feature Models. In: Männistö, Tomi and Bosch, Jan (eds.): Proceedings of Software Variability Management for Product Derivation - Towards Tool Support, a workshop in SPLC 2004. Helsinki University of Technology, Espoo, Finland (2004) 24-35
22. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A Practical Tool for Mass-Customising Configurable Products. In: Proceedings of the International Conference on Engineering Design (ICED'03), Stockholm, Sweden (2003)
23. Asikainen, T., Soininen, T., Männistö, T.: Towards Managing Variability Using Software Product Family Architecture Models and Product Configurators. In: van Gurp, Jilles and Bosch, Jan (eds.): Proceedings of Software Variability Management Workshop. IWI preprint 2003-7-01. University of Groningen, Groningen, The Netherlands (2003) 84-93
24. Asikainen, T., Soininen, T., Männistö, T.: A Koala-Based Ontology fro Configurable Software Product Families. In: Configuration Workshop of 18th International Conference on Artificial Intelligence (IJCAI) (2003)
25. van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The Koala Component Model for Consumer Electronics Software. IEEE Computer 33(3) (2000) 78-85
26. Asikainen, T., Männistö, T., Soininen, T.: Kumbang: A Domain Ontology for Modelling the Variability in Software Product Families. Advanced Engineering Informatics (accepted for publication) (2006)
27. Myllärniemi, V., Asikainen, T., Männistö, T., Soininen, T.: Tool for Configuring Product Individuals from Configurable Software Product Families. In: Männistö, Tomi and Bosch, Jan (eds.): Proceedings of Software Variability Management for Product Derivation - Towards Tool Support, a workshop in SPLC 2004. Helsinki University of Technology, Espoo, Finland (2004) 106-109
28. Thiel, S., Ferber, S., Fischer, T., Hein, A., Schlick, M.: A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems. In: Proceedings of In-Vehicle Software 2001 (SP-1587) (2001) 43-55