

Efficient Variability Treatment based on XML

Martin Becker

System Software Group
University of Kaiserslautern
Kaiserslautern, Germany
mbecker@informatik.uni-kl.de
www.gss.informatik.uni-kl.de/Mitarbeiter/mbecker/

Keywords: System Family Engineering, Variability Evolution, XML

Classification: 3 year's work

1 Introduction

For the last three decades software reuse has been the most promising approach to improve the efficiency of software development as regards time, cost and quality. Meanwhile it has shown that the ad-hoc reuse of software artefacts is often doomed to failure primarily due to searching and adapting problems. On the other hand, respectable benefits have been made with software especially developed for reuse, e.g. software that is developed for a distinct domain of applications [Bosc00]. Software product line or generally system family engineering approaches try to exploit the reuse potential in such domains systematically.

Domain or family-oriented software development [Parn76] means to identify commonalities and variabilities for a set of considered systems and provide appropriate software solutions in regard to their expected reuse potential. Comparing the methods used to develop single systems on the one hand and system families on the other, the obvious difference is that we have to deal with variabilities in the latter case and we have to accomplish this efficiently and consistently to be successful. Special variability aware methodologies to develop and evolve such system families are required. While the initial engineering of such families is already supported quite well – especially on the feature level – satisfactory support for the instantiation and evolution of system families is still lacking.

In this thesis, a method to support the evolution of variability in system families in a consistent and efficient manner is developed and validated. Based upon a comprehensive variability model and a technique to specify variability in parameterised or so-called generic software artefacts, the evolution of the variability, i.e. assimilation of further variants, change of the variability resolution, restructuring the variability etc., due to changing requirements is facilitated. To this end, the potential of XML [W3C98], related languages and tools to realise and evolve variability in a uniform manner is analysed and exploited.

2 Problem

Variability plays an important role for the success of software reuse, especially in the context of system family engineering, as software that offers variabilities can be deployed in a broader range of contexts [Bass97]. It paves the way out of the dilemma to build general and in principle reusable software that often does not meet the special demands of the target systems sufficiently or to build specialized software that cannot be deployed in other contexts. However, variability is often not explicitly addressed by the software development methods. The fact that variability has to be evolved during a system family's lifecycle due to additional or changing requirements further aggravates the situation.

Considering a system family and the set of artefacts used to specify it on the different levels of abstraction, e.g. requirements, architecture, components etc., variability crosscuts the artefact hierarchy (cf. fig. 1); variability usually bears an impact on software artefacts on several levels of abstraction (vertical dimension), but can also affect more than one artefact on the different levels of abstraction (horizontal dimension). This results in interdependencies of the artefacts that also have to be considered during the development and evolution of the variabilities. To facilitate their consistent evolution efficient impact analysis, traceability, change propagation and assimilation is required.

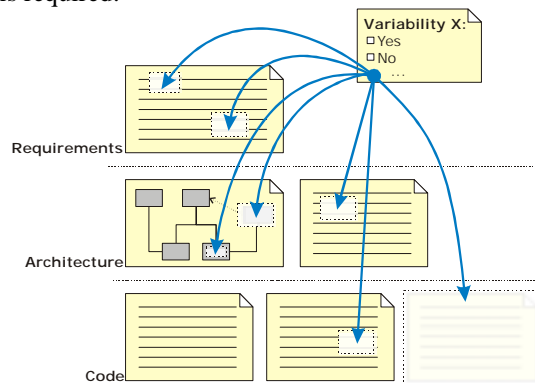


Fig. 1. Crosscutting impact of Variabilities.

Up to now, the resolution of variability, i.e. the replacement of so-called variation points in the artefacts with concrete solutions during the product instantiation phase, usually is only applied at the code level, although reuse on higher levels of abstraction bears a great reuse potential, too. This often leads to inconsistent documentations, as changes in the code are not propagated into the higher level descriptions. But generic and generative approaches can also be applied on the higher levels of abstraction. The approaches developed and evaluated in this thesis address the above mentioned problems and represent our vision of XML-based variability treatment.

3 Variability Modelling and Specification

Variability treatment is the fundamental difference between single system and system family engineering. A lot of variability related information exists at the design and development time of the variable artefacts that constitute the system family. Some of this information is included in the artefacts, the rest of this information tends to vanish. Such a representation obstructs the efficient resolution and evolution of variability, which is crucial for the successful reuse of the system family.

An explicit variability model acting as a carrier of all variability related information like specifications, interdependencies, origins etc., can play an important, maybe the central role in successful system family engineering. It can be the entry point for all variability related tasks, as configuration, instantiation and evolution. Inspired by the experience on the field of decision modelling [Baum00B,Baum00] a comprehensive variability model to capture the respective information has been developed. Besides the more decision related information, e.g. the possible choices, their dependencies and resulting restrictions etc., information about the origins, the binding times, the impact and the resolution support of variabilities can be captured in the model. Concerning the representation of the model, several approaches are sensible. A XML-based approach seems to offer the most flexible access to various information included in the model for the respective tools. specify

The adaptations that have to be accomplished in the generic software artefacts in order to instantiate them into deployable artefacts are not specified in the variability model. They are stated along with the (clearly identifiable) variation points in the generic artefacts, as they adhere more to the variation points than to the variabilities that parameterise them. But how to specify those adaptations? As mentioned above, the specifications on the higher levels of abstraction bear a considerable reuse potential, and therefore should also be addressed by the developed methods. The range of considered artefacts therefore reaches from requirements specifications over architectural descriptions to source code. The greatest common denominator to specify the considered artefacts was XML. Although perhaps being inconvenient to most programmers, representing source code in XML offers some further opportunities that result from the clear separation of structure, content and representation and the linkage support offered by XML. Some of these opportunities, like view-based editing and aspect-oriented specifications using XSLT will also be discussed in the thesis.

A small language to identify the variation points and to specify the required adaptations in the XML-files has been developed. It supports internal choices of pre-defined solutions and the construction of simple generators as well as the inclusion of externally generated or user-defined solutions. Using this language to describe the adaptation in a uniform manner, independent from the content of the files, the developed methods to support the realisation and evolution of variability as presented in the following section can be applied to a broad spectrum of specifications.

4 Evolutionary Support

The evolution of variabilities and the respective variation points in system families can be driven by several motivations:

- a yet unimplemented (no resolution support) but foreseen feature has to offer better resolution support
- a further variability has to be added, as a previously common (mandatory) feature becomes optional, for instance
- a new, formerly unconsidered feature has to be supported
- variable implementations have to be maintained
- further artefacts have to be added (, e.g. on lower levels of abstraction)

Each of those evolution scenarios involves considerable efforts and potential inconsistencies that diminish the overall benefit and thus endangers the successful reuse of software aimed at by system family engineering.

The goal of the developed method is to ease the treatment of variability in the above mentioned cases. To reach this, we pursue the following approaches:

Explicit variation points – the variation points in the generic artefacts and their dependencies on the variabilities are explicitly specified and therefore can be identified and evaluated by tools [Beck00]. The impact of a variability's evolution thus can be quite easily assessed and the corresponding evolution requests can be propagated to the respective developers.

View-based editing – the evolution of variants can be alleviated to a large extent, if variability does not have to be considered explicitly. After the developer has specified the variant he wants to develop through a corresponding configuration profile, he can work with a partial resolved artefact (unnecessary variability details are removed). The committed modifications are afterwards automatically merged in the generic artefact, i.e. the resolution specifications in the respective variation points are adapted.

Assimilation of variants – unforeseen adaptations can be assessed for their relevance and can be identified and assimilated into the generic artefacts if sensible. For this, the difference of the structure and content has to be analysed, which will be supported by an appropriate tool.

Explicit evolution of derived artefacts – evolving generic artefacts often bears the danger of evolving formerly derived artefacts implicitly [Bosc00]. The derivation and the evolution therefore are augmented to avoid this problem. If only some of the derived artefacts should be evolved, additional variability has to be added consistently. This also can be supported to a large extent by tools.

Compact artefact representation – artefacts derived from generic artefacts will be represented as compact as possible; they are represented by a reference to the generic artefact they stem from, the configuration profile and additional deltas, if the artefacts have been adapted after the derivation. Doing so, the common specifications fragments are only stored once and can be evolved easier.

The method and the corresponding tools will be validated in different domains and on different levels of abstraction, e.g. in the domain of embedded operating systems and in the domain of business applications. Besides the code level, the treatment of variability on the architectural level will also be analysed, which seems to be challenging due to the presence of artefacts with graphical representations.

5 Summary and Outlook

In this thesis a method and tools are presented that address some of the problems which arise from the necessity to handle variabilities in system families (e.g. in software product lines) efficiently and consistently. By concentrating on XML-based specifications, the results of this thesis are expected to be applicable to an increasing number of different specification types.

The thesis is planned to be completed in summer 2002.

6 References

- [Baum99] Baum, L.; Becker, M.; Geyer, L.; Gilbert, A.; Molter, G.; Tamara, V.: Supporting Component-Based Software Development Using Domain Knowledge, 4th World Multiconference on Systemic, Cybernetics and Informatics (SCI2000), 2000
- [Baum00] Baum, L.; Becker, M.; Geyer, L.; Molter, G.: Mapping Requirements to Reusable Components using Design Spaces, Proc. of the IEEE Int'l Conference on Requirements Engineering (ICRE-2000), Schaumburg/Chicago, USA, June 19-23, 2000
- [Bass97] Bassett, P.G.: Framing Software Reuse - Lessons From the Real World, Yourdon Press Computing Series, 1997
- [Beck00] Becker, M.: Generic Components: a symbiosis of paradigms, 2nd International Symposium on Generative and Component-Based Software Engineering (GCSE'00), 2000
- [Bosc00] Bosch, J.; Högström, M.: Product Instantiation in Software Product Lines: A Case Study, 2nd International Symposium on Generative and Component-Based Software Engineering (GCSE'00), 2000
- [Parn76] Parnas, D. L.: On the Design and Development of Program Families, IEEE Transactions on Software Engineering, VOL. SE-2, No. 1, March, 1976
- [W3C98] W3C: Extensible Markup Language (XML), <http://www.w3.org/XML/>, 1998