

# Handling Variants in a Product Family

Hongyu Zhang

Department of Computer Science, School of Computing,  
National University of Singapore  
Lower Kent Ridge Road, Singapore 117543  
zhanghy@comp.nus.edu.sg

**Keyword:** Product Family, Domain Engineering, Frame Technology, Software Reuse

**Classification:** First year's Ph.D. work

## 1 Introduction

In recent years, software product family approach, initiated by Parnas back in 1970s [PARN76], has emerged as one of the most promising reuse approaches to improve software productivity and quality. Instead of developing each individual product separately, the product family approach focuses on a specific problem domain, considering the family of products as a whole.

In product family approach, we identify commonalities among the members of a product family, and build generic assets (such as specification, domain model, generic architecture, components, test cases and documentation) that are reusable across product family members. For developing each individual product, we reuse the generic product family assets instead of designing from scratch.

While having much in common, members of a product family also differ in functional and non-functional requirements, design decisions, platforms and other characteristics. These variabilities give rise to variants in a product family. The generic product family assets should be able to accommodate variants (including both anticipated and unexpected variants) so that they can satisfy the needs of each specific product.

We find that the variabilities are more difficult to handle than the commonalities. Some variabilities may cut cross many product family assets; one product family asset may also be affected by many variabilities. Variants in a product family may have relationships such as alternative, optional and or-relationships. Inter-dependency among variants may further restrict the number of possible variant combinations. How to effectively handling variants in a product family, especially in a large and complex product family, to rapidly produce a specific product, is a major challenge facing both product family researchers and practitioners.

Feature model [KANG90], has been used to represent commonalities and variabilities in a domain. It provides graphical diagrams that depict structural relationship among anticipated features. However, feature models must be complemented with other notations, for example UML, to model the static and dynamic behavior of a domain. Notations traditionally used in requirement analysis can be extended with the concept of "variation point" [JACO97] and "extension

mechanisms”[RUMB99] to make them useful in domain modeling. However, as the number of variants increases, the explosion of possible variant combinations, the complex relationships and inter-dependencies among variants, and the “crosscutting” variabilities may cause the domain model to be difficult to understand.

The above-mentioned problems also occur when we build a generic architecture for a product family. A generic architecture for a product family defines an architecture shared by product family members and provides components and connectors to be reused across the members. The components and connectors in a generic architecture must be flexible enough to facilitate accommodation of variants, so that the specific product, member of a product family, can be rapidly derived from the generic architecture after possible customization. However, for a large and complex product family, without effective mechanisms and tools for managing variabilities, the customization and evolution of a generic architecture could be difficult.

Variabilities also affect other product family assets such as test cases and documentations. Tracing multiple occurrences of the same variability in different product family assets and understanding how mutually dependent variants affect each other is important. Simply by maintaining the traceability of variabilities manually is insufficient and does not scale up.

## 2 Mechanisms for Handling Variants

As early as in 1970’s, Parnas [PARN72] proposed modularization, information hiding and separation of concerns principles for handling variants in a product family. Marco Processor, Object-Oriented framework [JOHN88], Design Pattern [GAMM94], Application Generator [BATO98], Frame Technology [BASS97] and Template [CZAR00] – they all offer mechanisms to handle variants in product family. Recent work focuses on advanced separation of concerns [KICZ97, TARR99], which are proposed to address crosscutting concerns and concern compositions.

We started by exploring Frame Technology [BASS97]. Frame method and tool have an excellent record in industrial applications. A quantitative study has shown that Frame technology can lead to reduction in time-to-market (70%) and project costs (84%) [BASS97]. However, Frame technology is tightly coupled with COBOL. We have designed an XML-based language called XVCL (XML-based Variant Configuration Language) [WONG01] to overcome this limitation. XVCL supports all major frame commands and can be used to mark-up any textual contents. Being based on XML, XVCL is an open, extendable and easy to use language.

In our XVCL-based approach, we use the term *x-frame* to represent a piece of reusable asset, which is also instrumented (marked-up) with XVCL commands to accommodate variants. For example, a <SELECT> XVCL command in an x-frame indicates that only one of many anticipated changes will be included into the x-frame, a <BREAK> XVCL command indicates a “slot” in an x-frame at which possible unexpected changes may be inserted. An x-frame could also contain generic parameters and macros. Being generic and adaptable, x-frames can be easily reused. x-frames can be also organized into a *framework*, where upper layer x-frames use and adapt lower layer x-frames, forming a tree-like hierarchy (formally, it’s a semi-

lattice). The composition of the x-frames is achieved by the <COPY> and <INSERT> XVCL commands. x-frame based frameworks enable the reuse at any level of granularity.

We apply XVCL to handling variants in product family assets such as UML domain models and generic product family architecture. In domain models, an x-frame may contain adaptable fragments of use case, activity diagram or object collaboration diagram. In product family architecture, an x-frame may contain an adaptable component (or part of it such as a method, class, declarations of data structures) or a connector (defined, for example, in IDL). By using XVCL, we transform the domain model and the generic product family architecture into hierarchies of x-frames (frameworks). We can then design various tools to manipulate the x-frame hierarchies. For example, we design the *XVCL processor*, which takes the specifications that describe the customizations required for a specific product, adapts the domain model framework and the generic architecture framework, and generates customized model and code for the specific product. With the help of the XVCL-based tools, the customization and evolution of product family asset can be semi-automated, capable of scaling up.

Like in aspect-oriented programming [KICZ97], we also attempt to isolate different computational concerns (both functional and non-functional) into separate x-frames to achieve separation of concerns. Simultaneously, we can also isolate the crosscutting variability into separate x-frames [ZHAN01]. The composition of x-frames also achieves composition of multiple concerns and their variants. This advanced form of separation of concerns eases the maintenance and evolution of the product family assets, and facilitates reuse.

### **3 Current Status and Future Work**

We are currently working with our project partners on applying XVCL and its processor to handling variants in the Computer Aided Dispatch (CAD) product family. CAD systems are mission-critical systems that are used by police, fire & rescue, health service, port operations and others. They are designed to help the operators to dispatch resources (e.g. police units) to handle incidents. Our initial experimentation on handling variant requirements in CAD domain model is reported in [JARZ01].

We believe our approach offers the benefits of effectively organizing product family assets, better support for customization and ease of evolution. In the future, we shall:

#### *Experiment on a larger scale*

We shall continue developing the CAD product family with our project partners, and extending the scope of the experimentation. We shall also apply our approach to other industrial projects in different domains.

#### *Evaluate our approach*

We shall evaluate our approach and compare it to other works such as Object-Oriented framework [JOHN88], Aspect-Oriented Programming [KICZ97] and Template [CZAR00]. It's important to explore the similarities and the differences between an x-frame and an object, an aspect and a template. We shall also design methods and tools for measuring the reusability of the x-frames and frameworks.

#### *Explore various types of variants and variant dependencies*

So far, we have been dealing with relatively simple class of the variants – functional variant requirements in our experimentation. We shall continue experimenting with a large number of variants including non-functional variant requirements. The total number of possible variant combinations will be less if variants are inter-dependent. Methods and tools should be developed to check the variant dependencies. Only after we identify a valid set of variants can we correctly customize the product family assets.

#### *Develop an XML-based integrated environment*

We shall enhance our XML-based language and tools. For example, in order for reusers to better understand and evaluate our x-frame based framework, we are developing a framework analysis and query tool, based on the Static Program Analyzer (SPA) techniques [JARZ98] and XQL [ROBI98]. The goal is to develop an XML-based integrated environment for organizing and reusing product family assets effectively.

## **Acknowledgements**

This work is supervised by Dr. Stan Jarzabek (stan@comp.nus.edu.sg) at Department of Computer Science, School of Computing, National University of Singapore, and supported by research grant NSTB/172/4/5-9V1 funded under the Singapore-Ontario Joint Research Programme by the Singapore National Science and Technology Board and Canadian Ministry of Energy, Science and Technology, and by NUS Research Grant R-252-000-066.

## **References**

- [BASS97] Bassett, P. *Framing Software Reuse - Lessons from Real World*, Yourdon Press, Prentice Hall, 1997
- [BATO98] Batory, D., Lofaso, B. and Smaragdakis, Y. JST: Tools for Implementing Domain-Specific Languages. *Proc. 5<sup>th</sup> Int. Conf. on Software Reuse*, Victoria, BC, Canada, 1998
- [BAUM00] Baum, L., Becker, M., Geyer, L. and Molter, G. Mapping Requirements to Reusable Components using Design Spaces, *Proc. Int'l Conf. on Requirements Engineering*, Schaumburg, Illinois, USA, June 2000
- [CZAR00] Czarnecki, K. and Eisenecker, U. *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, Reading, 2000

- [GAMM94] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns*, Addison-Wesley, 1994
- [JACO97] Jacobson, I., Griss, M. and Jonsson, P. *Software Reuse – Architecture, Process and Organization for Business Success*, Addison Wesley Longman, 1997
- [JARZ98] Jarzabek, S. Design of Flexible Static Program Analyzers with PQL, *IEEE Transactions on Software Engineering*, Vol. 24, No.3, March 1998
- [JARZ01] Jarzabek, S. and Zhang, H. XML-based Method and Tool for Handling Variant Requirements in Domain Models, To appear, *Int. Symposium on Requirements Engineering (RE'01)*, Toronto, Canada, August 2001
- [JOHN88] Johnson, R. and Foote, B. Designing reusable classes, *Journal of Object-Oriented Programming*, 1, 2, pp. 22-35, 1998
- [KANG90] Kang, K. et al. "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Nov. 1990
- [KICZ97] Kiczales, G. et al. Aspect-Oriented Programming, *Proc. European Conference on Object-Oriented Programming (ECOOP)*, Finland, June 1997
- [PARN72] Parnas, D. On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, 15(12):1053-1058, December 1972
- [PARN76] Parnas, D. On the Design and Development of Program Families, *IEEE Transaction on Software Engineering*, March 1976
- [ROBI98] Robie, J., Lapp, J. and David, S. XML Query Language (XQL), The W3C Query Languages Workshop, Boston, 1998
- [RUMB99] Rumbaugh, J., Jacobson, I. and Booch, G. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999
- [TARR99] Tarr, P., Oshser, H., Harrison, W. and Sutton, S. N Degrees of Separation: Multi-Dimensional Separation of Concerns, *Int. Conference on Software Engineering, ICSE'99*, Los Angeles, 1999, pp. 107-119
- [WONG01] Wong, T.W., Jarzabek, S., Soe, M.S., Shen, R. and Zhang, H. XML Implementation of Frame Processor, *Proc. Symposium on Software Reusability, SSR'01*, Toronto, Canada, May 2001
- [ZHAN01] Zhang, H., Jarzabek, S. and Soe, M.S., XVCL Approach to Separating Concerns in Product Family Assets, To appear, *Generative and Component-based Software Engineering (GCSE 2001)*, Erfurt, Germany, September 2001