Traceability for System Families

Detlef Streitferdt
Ilmenau Technical University
P.O. Box 100565, 98684 Ilmenau
detlef.streitferdt@theoinf.tu-ilmenau.de
www.theoinf.tu-ilmenau.de/~streitdf

Abstract

System families are an idea of software reuse in a specific problem domain. Existing methods have little requirements engineering support for system family development. This short paper proposes a requirements metamodel for system family development. Traceability throughout model elements is a necessary precondition for preserving the consistency of the complete family model during development and is a main issue in this paper as well as for software development in general. Family development based on the metamodel guarantees traceability by the inclusion of all development artifacts in a single and consistent model.

Keywords

System families, traceability, software reuse

1 Introduction

Software developers try to satisfy the demands for short development cycles with iterative development methods and reuse efforts. Within software development, the requirements engineering phase results in a specification document, upon which all other development activities and the contract between the supplier and manufacturer of the system are based. The outcome of a project and the problem comprehension are mainly dependent on the requirements engineering phase and its results. The system family context puts even higher demands on the requirements engineering phase, since many applications rely on developed family assets.

System families, based on the work of Parnas [5] in 1976, are the idea of software reuse in a given problem domain. Through a comparison of a set of given similar systems, developers can extract common assets found in all systems and variable assets spread over all systems. This analysis results in a flexible family architecture for the development of future variants.

Existing system family methods adopt ideas out of the domain engineering discipline, which is divided into three parts. Domain analysis forms the commonality and variability data basis. In the following domain design phase a flexible architecture is developed. Based on this architecture applications can be derived and implemented which is done in the domain implementation phase.

Feature Oriented Domain Analysis (FODA, [3]) addresses the first phase of domain engineering. By hierarchically arranging all common and variable features of a domain, relationships between different system variants are modeled. No explicit assignments of requirements to features are described in the method, what opens a gap between the model of requirements and the model features. Methods like Reuse-driven Software Engineering Business (RSEB, [1]) attempt to fill this gap with a use-case driven approach and object oriented development. RSEB reuse is achieved with component systems and a given reference architecture. The successor method FeatuRSEB [4] integrates feature modeling and RSEB to offer a family view onto the model. FeatuRSEB models the mutual affiliation of system features and their affiliation to design assets. Thus a model of variants is developed, whereas the clear association of requirements to features is still missing.

Existing approaches try to address system family aspects by variability mechanisms for modeling systems with the Unified Modeling Language (UML) as in RSEB [4], by defining text based composition rules as done with frames in [6], or by defining merge rules for composing a system architecture out of a set of models as in Subject Oriented Design [7]. In all methods a requirements model is missing which incorporates the notion of commonality and variability with the possibility of accessing other development elements. Current methods address the requirements engineering phase by high level constructs such as use-cases. This is insufficient, since not all systems are workflow oriented. Thus a lower level approach is needed.

2 Requirements Model

The proposed solution consists of a metamodel upon which a low level model of requirements can be build for developing system families. The metamodel as shown in Figure 1, has a requirement as the central model element. This requirements is an indivisible piece of text describing a small

portion of the system to be developed. A set of requirements can be assigned to a feature and features can be hierarchically modeled as described in FODA.

The left part of the figure depicts the system family model to which all requirements are related. System families consist out of a core and many variable parts. Each requirement must either belong to the core or to a set of variable parts.

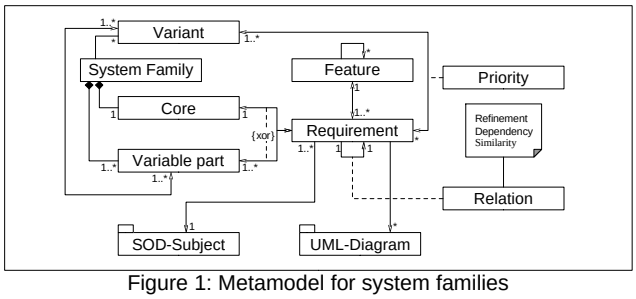

Figure 1: Metamodel for system families

A system family is intended to be the starting point for the development of many variants in the given domain. Thus the family contains a set of variants, which represent configurations of the applications to be implemented. Each variant implicitly contains the family's core and a set of variable parts. Based on the configuration of a variant the corresponding application can be rebuild at any time.

Priorities are important for planing and managing the implementation of an application. In addition to the priority of a requirement, a priority for the relation of a requirement to a variant and thus an application of the family is included in the metamodel, as stated in [2]. Identical requirements for different variants might have varying importance according to the customer's intended use of the system.

UML diagrams are a central modeling concept of the design phase. For the requirement engineering phase the diagrams are used to explain the bare text. Usage of diagrams instead of lengthy text paragraphs should be encouraged. In the later design phase the architecture of an application is developed by refining existing diagrams and adding new ones, whereas changes in an application model need to be related to the corresponding requirements.

Application development out of a flexible family architecture is done by combining model elements. The set of model elements is identified by the configuration of a desired variant. These elements are composed using rules described in Subject Oriented Design [7].

Relations between requirements can be modeled to formulate constraints. The hierarchical arrangement of requirement is referred by refinement relations. Mutual inclusion or exclusion of sets of requirements are modeled with dependency relations, and similarity relations can be used to model a coherent set of requirements.

A model derived from the metamodel contains all elements created by the development activities. Several views can be derived from this model. Depending on the role of the person in the project, the relevant data can be extracted and presented. The model can be analyzed, to generate data for management decisions.

Traceability between development artifacts is built into the metamodel. For any given model element the correspondence to other model elements and thus traceability information can be extracted out of the requirements model. For system family development a centralized data model based on the proposed metamodel is needed to enable a complete and consistent traceability. In addition tool support is needed to perform consistency checking and analyses.

3 Summary and Outlook

This research integrates the ideas of existing methods and models to form a new metamodel. The central concept of an indivisible requirement together with the model of system families enables general requirements engineering for system families.

Current work is focused on a development method for system families and the refinement of a prototype based on the eXtensible Markup Language (XML). The prototype is being used in university projects and a cooperation project with industry partners, to evaluate, refine and validate the proposed metamodel in practice.

References

1. Ivar Jacobson, Martin Griss, Patrik Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success.* Addison-Wesley-Longman, 1997.
2. Juha Kuusela, Juha Savolainen, Requirements Engineering for Product Families. *Proc. of the 1997 International Conference on Software Engineering (ICSE)*, 1997.
3. Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, Feature-Oriented Domain Analysis (FODA): Feasibility Study. *Technical Report CMU/SEI-90-TR-21*, Software Engineering Institute, 1990.
4. Martin L. Griss, John Favaro, Massimo d' Alessandro, Integrating Feature Modeling with RSEB. *Hewlett-Packard Company*, 1998.
5. Parnas D. L., On the design and Development of Program families. *IEEE Transactions on Software Engineering*, SE-2: (March 1976).
6. Paul G. Basset, *Framing software reuse: Lessons From The Real World*.(1997), Prentice-Hall.
7. Siobhán Clarke, William Harrison, Harold Ossher, Peri Tarr, Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. *Proc. of Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, 1999.