# What Can the ConIPF Methodology Offer for Requirements-driven Reuse-oriented Software Development?

Lothar Hotz[1] and Thorsten Krebs[1]

HITeC c/o Universität Hamburg, Department Informatik
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{hotz|krebs}@informatik.uni-hamburg.de

**Abstract.** A vision in software development is to start from a requirements specification and automatically derive software products that realize these requirements. This short paper summarizes the tasks needed to implement this vision and discusses what the ConIPF methodology (Configuration in Industrial Product Families) can offer as a starting point for related research activities.

## 1 Introduction

In this paper we outline some first ideas to support the creation of a sophisticated requirements-driven, reuse-oriented software development framework. The major tasks of this development framework are: a modeling *language* to capture the requirements specification and transformation languages to transform the requirements stepwise into product architecture, detailed design and code artifacts, and a *query engine* for retrieving software *cases* that have been previously developed and stored. This includes the definition of a query language and identification of a similarity measure, and a *methodology* for describing the complete life-cycle of software cases. Thus, the methodology covers aspects of customer requirements, product architecture, detailed design and code artifacts. A case is defined with the modeling language and stored and retrieved with the query engine [1].

The ConIPF methodology (Configuration in Industrial Product Families) [2] covers several aspects that can deal as a starting point for supporting these tasks. This methodology enables automated development of products starting from given customer requirements and using a repository of reusable artifacts.

The remainder of this paper is organized as follows. In Section 2 we generally describe the problem current software development struggles with. Section 3 summarizes the ConIPF methodology. Section 4 explains the tasks and our first ideas for solving those in more detail. Finally, Section 5 gives a conclusion.

## 2 The Problem

It is a common trend that software systems become more and more complex. This is due to increasing customer requirements and advancing technical possibilities

in using technical and electronic systems. Companies invest huge amounts in software development in order to stay competitive.

Reuse has long been identified as a key in enhancing software development. But due to the complexity and amount of software systems produced, software development projects tends to ignore previous products. This is because different personal is working on the projects, previous solutions are forgotten about in long development cycles, and most notably because it is hard to measure similarity.

## 3 Introduction to the ConIPF Methodology

ConIPF combines the research fields of software product lines and structure-based configuration. The product line engineering helps to design, implement and derive similar but distinct products. This is achieved by different compositions of the reusable artifacts with respect to technical possibilities and given customer requirements. In structure-based configuration, *configuration models* are used, which contain a textual description of the artifacts, their capabilities (*features*), properties and relations to other artifacts. Thus, a configuration model describes commonality and variability as well as restrictions within and between artifacts of a product family. With the configuration model, all potentially derivable products are implicitly specified. The use of configuration tools automates product derivation and consistency tests on the basis of the configuration model.

Furthermore, the ConIPF methodology provides a SPEM-based derivation process[1]. This process distinguishes between *configuration* activities and *realization* activities. During configuration activities decisions about the product are made manually or automatically by the configuration tool. Those decisions are covered in a *product description*. During realization activities the product description is mapped to a product implementation. The configuration activities are formally specified with structure-based configuration techniques, whereas the realization activities are organization-specific and not mapped to specific implementation techniques.

## 4 Tasks and Approaches

### 4.1 Requirement Specification Language Definition

In the ConIPF methodology, the Asset Modeling for Product Lines (AMPL) language enables modeling of features and their structure and characteristics as well as their relations to other assets types, like software or hardware artifacts. *Features* are prominent or distinctive, user-visible aspects of the products [3] – this means they represent the products functionality. There are features common to all products and optional features that together make up commonality and variability of the product family.

[1] Software Process Engineering Meta model (SPEM), see http://www.omg.org/technology/documents/formal/spem.html

This representation can give input for developing a requirements specification. But it has to be analyzed if the expressivity of feature models suffices to describe customer requirements. For example there may be reasons for strong and weak requirements, which are requirements that have to be and should be fulfilled, respectively (see e.g. [4]). Additionally, functional requirements can specify aspects like "choose the cheapest version" or that "power consumption of all components should not exceed a certain limit". It is apparent that sophisticated representation facilities are needed to capture this kind of requirements.

### 4.2 Modeling and Transformation Languages for Software Cases

AMPL allows to model features and reusable artifacts in hierarchical structures. With this, the commonality and variability of products can be represented. The artifacts selected during product configuration and their characteristics and relations to other assets (like features) describe the solution of a software case.

For achieving such a solution, AMPL defines a mapping between the feature layer and an artifact layer. This mapping describes a "realizes"-relation between the two knowledge entities. Artifacts *realize* features, and vice-versa features *are realized* by artifacts. Since multiple artifacts may be needed to realize some specific functionality and an artifact may realize multiple features, this relation is a n-to-m relation. When a feature is selected for a product, the corresponding artifacts are automatically selected, too. Thus, the mapping keeps track of the transformation history between customer selections and architecture or design decisions.

Furthermore, the AMPL language contains facilities to model subsystems that can consist of hardware and software artifacts. This might be a valuable input for defining architecture and / or design patterns for identifying groups of code that are commonly used together or form logical units.

These representation facilities can be used as a starting point for modeling the layers of a software case. The representation of the product architecture and its detailed design and the transformations between the steps from requirements to architecture, from architecture to design, and from design to code are an open issue. It is apparent that a simple mapping between the layers is not sufficient for unambiguously transforming one layer into the next layer.

### 4.3 Software Case Query Language

A major part of human expertise is believed to be past experiences. Case-based reasoning provides a model for representing experience in so-called *cases* (i.e. former software development problems and their solutions) and reusing them for solving new developments ([5]). There are two challenges: how to represent software cases (see 4.2) and how to retrieve them from a case base and identify which case fits best to a given new situation. Similarity has to be defined for this task.

Appropriate cases should be identified based on the customer requirements, not only the code artifacts of the solution. Thus, the requirements specification

of the current case has to be compared to the requirements specifications of the stored cases. Using feature models to represent customer requirements, they are modeled in hierarchical structures. These structures can be used to identify similarity more precisely. For example a subfeature of another feature is very similar to the latter because they are both of the same type (one is a specialization of the other). Two siblings are also similar because both inherit characteristics of their common parent. Two features in different places of the tree structure do not have anything in common.

### 4.4 Software Development Methodology

For enabling practitioners to use the requirements specification language, the modeling and transformation languages and the retrieval mechanisms, a methodology is needed that describes how to use these techniques. As described in Section 3 the ConIPF methodology provides several techniques that can be used for variability handling. However, including model-driven software development approaches, probably as specific realization activities is a challenge.

## 5 Summary

The ConIPF methodology provides several mechanism for deriving products. To summarize, the following questions arose: Can features be used for modeling complex requirements? How can variability aspects (like represented in AMPL) be integrated with model-driven approaches? How can similarity of software cases be measured?

## Acknowledgments

This research has been supported by the European Community under the grant IST-2006-33596, Requirements-Driven Software Development System (ReDSeeDS).

## References

1. Smialek, M.: Can use cases drive software factories? In: 2nd International Workshop on Use Case Modeling (WUsCaM-05) – (MoDELS 2005), Montego Bay, Jamaica (2005)
2. Hotz, L., Wolter, K., Krebs, T., Deelstra, S., Sinnema, M., Nijhuis, J., MacGregor, J.: Configuration in Industrial Product Families - The ConIPF Methodology. IOS Press, Berlin (2006)
3. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021 (1990)
4. Thäringen, M.: Wissensbasierte Erfassung von Anforderungen. In Günter, A., ed.: Wissensbasiertes Konfigurieren. Infix (1995)
5. Sasikumar, M.: Case-based Reasoning for Software Reuse. In: Knowledge Based Computer Systems-Research and Applications (International Conference on Knowledge-Based Computer Systems), Bombai, India, Narosa Publishing House, London (1996) 31–42