Knowledge-based Product Derivation – Research Topics of the ConIPF Project

Lothar Hotz, Thorsten Krebs, Katharina Wolter

One major challenge in software industry is to cope with increasingly complex products and development processes. Nowadays, products are composed not only of hardware but also of software. The process of assembling such complex products can be supported by *configuration* mechanisms. The approach presented in this article combines mechanisms from *structure-based configuration,* which is well known in the AI community, with the *software product line* approach used in software engineering to expand the reuse of software. The aim of our approach is to enhance the capabilities of reusing components shared by different but similar products. *Features* are used for selecting the desired functionality and serve as a starting point for the product derivation process. The main objective is to manage the complexity of the derivation process, i.e. to guarantee completeness and correctness of the solution and to make sure that dependencies between artefacts are respected.

1 Introduction

In software industry, there are two contrary trends. On the one hand, software systems (must) become larger and of a higher quality because of increasing customer requirements and more complex system functionality. On the other hand, there is a need for reducing costs and shortening time-to-market in order to stay competitive.
Often the software cannot be addressed separate from the hardware environment it is embedded in – resulting in *software-intensive systems*. In this paper *Car Periphery Supervision (CPS)* systems that monitor the environment of a car are used for illustration purposes (see Section 2.1). The growing complexity and variability of technical systems replaces the development of single software products with the development of product families. *Product families* are used to describe variability and commonalities of different but similar products and enable high level software reuse, especially through the *product line* approach [Bosch 2000] (see Section 3.1). The goal of the *ConIPF* project (*Configuration in Industrial Product Families)* is to support and realise product development with methods from *structure-based configuration* (see Section 3.2). In Section 4 we focus on some major issues discovered in configuring software-intensive systems by describing requirements and their solutions.
ConIPF is a three year project that is supported by the EU under the grant IST-2001-34438. Four partners (two industrial and two university partners) are participating in the research work: Robert Bosch GmbH, Thales Naval Nederland, the University of Hamburg and the University of Groningen. In this paper we describe an intermediate result of the project and present research topics that are of interest to the University of Hamburg.

2 Aspects of Application Domains

In an analysis and characterization phase of the project several domains in the area of vehicle development were examined. As an example in this paper the CPS domain is presented (Section 2.1). From this analysis major distinctions between configuration of hardware and software components are identified (Section 2.2).

2.1 The CPS Domain

Car Periphery Supervision (CPS) systems monitor the local environment of a car. CPS systems comprise a family of automotive systems that are based on sensors installed around a vehicle. The recording and evaluation of sensor data enables different kinds of applications. These can be grouped into safety-related applications like pre-crash detection and comfort-related applications like parking assistance [Thiel et al. 2001].

Figure 1[1] shows the ranges of different sensor types that can be mounted on a vehicle. Different applications need different kinds of sensor data. For instance, while for parking assistance the range of 2.5 meters surveyed by using ultrasonic sensors is sufficient and also gives higher accuracy in that range, for pre-crash detection a longer distance has to be monitored and short range radar is required.

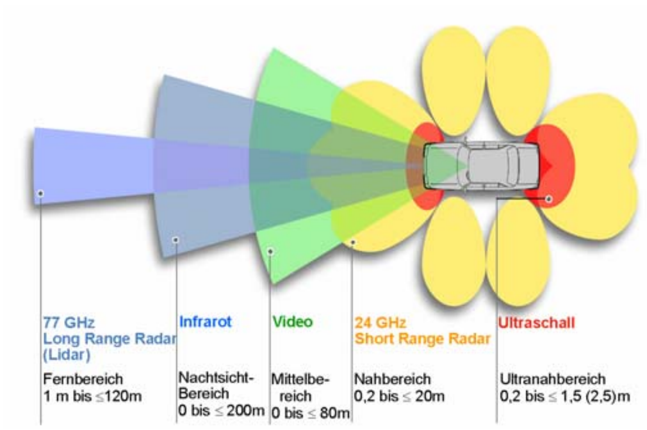

Figure 1 – Sensor Types and Supervision Ranges

2.2 Differences between Hardware and Software Configurations

Two major differences between hardware and software system configurations can be identified in the application domains surveyed in ConIPF:

- In hardware configuration there is a strict separation between the configuration and the manufacturing of the required product, i.e. collecting the desired components out of a store and assembling them. In software configuration the “manufacturing”, i.e. the implementation can be easily done on the same platform as the product configuration.
- Software is easily modifiable and therefore constantly changing. This makes configuration of software products more complex because of the increasing number of potentially unknown variants and versions of components and their dependencies.

3 Underlying Techniques

In ConIPF software product lines and structure-based configuration form the methodical foundation of research. They are briefly introduced in the following.

3.1 Software Product Lines

Software product lines provide a highly successful approach to strategic reuse of product components. The development of a product line and the development of products can be distinguished. These development tasks are identified as *domain engineering* and *application engineering* (compare [Bosch et al., 2001]):

- In domain engineering, architectures and reusable software components are developed. Exploiting commonality and managing variability is necessary and can be achieved by using feature models [Kang et al. 2002]. Features are ‘prominent or distinctive user-visible aspects of a system’ [Kang et al. 1990] and can be modelled in taxonomies with mandatory, optional and alternative properties.
- In application engineering existing artefacts are used to assemble specific products by analysing requested features, selecting architecture and adapting components. In the following this process is called *product derivation process*.

[1] The figure is made available by Robert Bosch GmbH.

Domain engineering and application engineering do not describe chronological tasks, but the distinction between developing a product line and developing products using the product line. The product derivation process is currently realised by communication facilities between participating humans and standardised documents in order to capture customer requirements or to define system specifications, and to realise change management. However, a general methodology for realising or supporting application engineering does not exist [Hein et al. 2003]. Therefore, it is a common to use previously developed products or platforms by a "copy and modify approach" to suit current customer needs. In large domains, however not every developer knows all existing code fragments and chooses from components known to him. This is rather error-prone in the sense that functionality is implemented anew where reuse would have been possible or incompatibilities between code fragments may not be detected leading to incorrect solutions.
One ConIPF goal is to fill this methodology gap and to support this process with mechanisms of structure-based configuration.

3.2 Structure-based Configuration

Configuration is a well known approach to supporting the composition of products from several parts. The configuration of technical systems is one of the most successful application areas of knowledge-based systems [Günter and Kühn 1999]. Basic modelling facilities enable the differentiation between three kinds of knowledge:

- *Conceptual knowledge* includes concepts, taxonomic and compositional relations as well as restrictions between arbitrary concepts (constraints).
- *Procedural knowledge* declaratively describes the configuration process.
- A *task specification* specifies properties and constraints that a product must fulfil.

The configuration itself is performed in an incremental approach, where each step represents a configuration decision and after each step a global mechanism can optionally be applied for testing, simulating or checking with constraint techniques [Günter 1995, Hotz et al. 2003]. Global in this case means that the entire configuration is examined. *Conflicts*[2] that are detected e.g. during the use of such global mechanisms are handled by conflict resolution mechanisms. However, applying configuration methods to software systems is in an early stage (see e.g. [Soininen et al. 1998]).

4 The ConIPF Approach

The main objective of the ConIPF project is to define and validate a methodology for knowledge-based product derivation of software-intensive systems that are based on a product family. Because of the large amount of variability in product families, the task of product derivation is time consuming and error-prone. By using structure-based configuration mechanisms, it is possible to model all assets developed during domain engineering and their relations in order to support the product derivation process. This is described in Section 4.1. Furthermore, integration of product configuration and product realisation in one process is introduced in Section 4.2. Because software is easily modified and therefore constantly changing, the aspect of evolution is of special interest and will be discussed in Section 4.3. When applying knowledge-based configuration to software-intensive systems, existing development environments have to be considered. To implement the methodology in an organization the methodology has to be tailored to the existing requirements. Furthermore, formal underpinning of the methodology for syntax and semantics (e.g. by applying description logics, compare [Möller et al. 1997]) is planned. These last two subjects are also research topics considered at the University of Hamburg but they are not further addressed in this article.

[2] A conflict is defined as a situation where the decisions made by the user, their logical impacts (e.g. computed by global mechanisms) and the configuration model are not consistent.

4.1 Consistency, Completeness, Correctness

Following manual processes like "copying and modifying" source code for similar products, the generated solutions can easily be *inconsistent*, e.g. the selected components do not fit because their interfaces do not match. Moreover, solutions can be *incomplete*, i.e. necessary components are missing in the product or solutions can be *incorrect*, i.e. components are included in the product but do not realise the needed functionality. Using structure-based configuration mechanisms provides support for such problems. When using models perfectly reflecting the product family and complete or partial mappings between features and artefacts, products are derived whose properties are consistent, complete and correct. Thus, it is ensured that the configured product is consistent, complete and correct with respect to the model.

Construction of a model that correctly reflects the product line is simplified by the following aspects:

- Because of using a product line, the underlying components are already implemented for reuse, which simplifies the formalization of variability and commonalities in a configuration model. In a more monolithically implementation the variability is more implicit and thus, more difficult to model.
- Methods like testing or verification are directly related to the product line, because they use the components, thus reflect the real world. Because methods like testing or verification are included in the ConIPF approach by means of global mechanisms the configuration model is checked during the product derivation process against the existing product line implementation.

In order to obtain models reflecting the product family there is a need to model all reusable assets in the product family. The first step is to describe the different kinds of assets that should be assembled during configuration and the relations between these assets. Assets under consideration are features, architecture, software and hardware components as well as their parameters and relations to other assets (e.g. requires, excludes, realises, etc.). Regarding software-intensive systems there are several levels of abstraction that need to be modelled for different tasks like features for modelling product capabilities, architectures on a high level and system functionality on a lower level. Furthermore, for software-intensive systems, hardware and relations between software and its hardware environment are also important in order to assemble a complete and consistent product.

In ConIPF, structure-based configuration and constraints are used to model and then configure CPS systems and software-intensive systems in general. Therefore, a domain-independent language is developed and used. Its modelling facilities are general enough to model and process the above described aspects of software-intensive systems.

4.2 Knowledge-based Product Derivation Process

The result of an application engineering process and software development in general is a concrete software artefact (i.e. a product) whereas configuration creates an abstract description of a product. In contrast to hardware domains, it is possible for software to realise the product within the derivation process (e.g. to compile source code, to calibrate the product, etc.). Therefore two processes are involved in knowledge-based product derivation: configuration and realisation. Since these two processes are dependent on each other they need to be synchronized.

The knowledge-based product derivation process defined in ConIPF addresses the entire development process. Thus, selecting features, architecture and hardware and software components are part of the derivation process just like the realisation of the software itself. Integration of configuration and realisation means for instance that code is generated and compiled during the configuration process. Code generation and code compilation can be initiated by means of global mechanisms (see Section 3.2). The point in the derivation process when compilation should be initiated is defined by procedural knowledge. This

ensures that it is not possible to start the compilation before all relevant configuration decisions have been made.
As described in Section 4.1, it is possible to guarantee a consistent and complete product using structure-based configuration. However, it is very important to document how e.g. a feature is realised and which components (e.g. sensors or software modules) are used for this because the application engineer then can comprehend the relation between customer requirements and the specific parts of the solution.
In structure-based configuration features and their interrelations, the structure of possible products and the mapping between features and products are defined in the configuration model. Therefore the relations between features and system components during the derivation process as well as for the solution can be documented.
Although the configuration process always leads to a consistent solution during the process there can be inconsistent partial solutions called conflicts. This can happen when e.g. the user selects certain features or components that cannot be combined. A conflict also occurs when a global mechanism recognizes an incompatibility, e.g. when the compilation cannot be executed successfully. To cope with such situations a history of all user decisions and the computed impacts is necessary to be able to select decisions that can be backtracked or modified for resolving this conflict. Because structure-based configuration uses an incremental configuration approach, all decisions and their chronological order are known and can be used to resolve conflicts.
It is possible that a conflict cannot be resolved - i.e. no correct solution can be generated for the customer requirements and the given configuration model. In such a situation evolution comes into play. This means, existing assets (and their corresponding description in the configuration model) are modified during the derivation process.

4.3 Evolution

All kinds of configurable assets are subject to evolution – including features, software and hardware components. For extending the configuration model, knowledge about the newest and / or all versions of the corresponding components is necessary.
In traditional application areas of knowledge-based systems (i.e. technical domains like aircrafts, drive systems, etc.), development and manufacturing of components is final. This means, all components are present, but have to be selected and parameterized by processing the configuration model. Due to easily changeable artefacts in the software domain a different situation exists. During the product derivation process, even when existing software artefacts are reused, modifications on those artefacts are often needed [Krebs et al. 2002]. The configuration model describes all members of a product family that can be derived using knowledge-based configuration techniques. Thus, the configuration model describes admissible configurations. This can be extended by anticipating future evolution to a certain extent e.g. by modelling planned features [Hein et al. 2001]. But eventually there are unpredicted requirements (like bug fixes) or other situations where evolution planning is not practical.
Evolution during domain engineering is the task of extending the configuration model, i.e. modelling new variants and versions of components or modifying existing ones. Methods of knowledge acquisition are sufficient for this task. During application engineering, the model is usually fixed for structure-based configuration techniques. This means possibilities for dynamically modifying the configuration model have to be taken into account to cope with new functionality during product derivation. Generating solutions that lie outside the modelled solution space is addressed in *innovative configuration* [Günter 1995].
The configuration model can be used for supporting evolution. It reflects all existing components and their dependencies. Thus, the impacts of a component modification or an addition of a component of a specific type can be computed by examining the configuration model. E.g. a component with fewer relations to other components can be evolved more easily than a component with more dependencies to others [Krebs et al. 2003].
Knowledge acquisition is the central aspect for modelling new concept descriptions as well as modifying existing concepts. So far, knowledge acquisition has only been taken into account for building a fixed configuration model. The same acquisition techniques can also

be applied to identifying and modelling new concept definitions or modifying existing concept definitions. With these tasks, a special emphasis has to be put on consistency because changes to the model can have impacts on the currently developed partial configuration.

5 Outlook

The newly developed methodology will be applied in experiments at both industrial partners. Data will be collected during those experiments and assessed. Topics of assessment are appropriateness of the knowledge representation, tools and the derivation process as well as improvements in the process compared to previous approaches and accommodation of reuse, adaptation and evolution strategies.

References

[Bosch 2000] J. Bosch, *Design & Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Addison-Wesley, 2000.

[Bosch et al. 2001] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink and K. Pohl, Variability Issues in Software Product Lines, In: *Proc. of the Fourth International Workshop on Product Family Engineering(PFE-4)*, Bilbao, Spain, October 3-5, 2001.

[Günter 1995] A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.

[Günter and Kühn 1999] A. Günter and C. Kühn, Knowledge-Based Configuration - Survey and Future Directions, In: Proc. of *XPS-99: Knowledge Based Systems*, Würzburg, Germany, 1999.

[Hein et al. 2001] A. Hein, J. MacGregor, and S. Thiel, Configuring Software Product Line Features, In: *Proc. of ECOOP 2001 - Workshop on Feature Interaction in Composed Systems*, Budapest, Hungary, June 18, 2001.

[Hein et al. 2003] A. Hein, J. MacGregor, Managing Variability with Configuration Techniques, in *Proc. of the Workshop on Software Variability Management at the ICSE*, Portland, Oregon, USA, May, 2003.

[Hotz et al. 2003] L. Hotz, A. Günter, and T. Krebs, *A Knowledge-Based Product Derivation Process and some Ideas how to Integrate Product Development (position paper)*, In Proc. of Software Variability Management Workshop, page 136-140, Groningen, The Netherlands, February 13-14, 2003.

[Kang et al. 1990] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, *Feature-oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

[Kang et al. 2002] K. Kang, J. Lee and P. Donohoe, *Feature-oriented Product Line Engineering*. In: IEEE Software, 7/8, pages 58–65, 2002.

[Krebs et al. 2002] T. Krebs, L. Hotz, and A. Günter, Knowledge-based Configuration for Configuring Combined Hardware/Software Systems, In: *Proc. of 16. Workshop, Planen, Scheduling und Konfigurieren, Entwerfen (PuK2002)*, Freiburg, Germany, October 10-11, 2002.

[Krebs et al .2003] T. Krebs, L. Hotz, C. Ranze and G. Vehring, Towards Evolving Configuration Models, In: *Proc. of 17. Workshop, Planen, Scheduling und Konfigurieren, Entwerfen (PuK2003)*, pages 123-134, Hamburg, Germany, September 15-18, 2003.

[Möller et al. 1997] R. Möller, C. Schröder and C. Lutz, *Analyzing Configuration Systems with Description Logics: a Case Study*, University of Hamburg, 1997.

[Soininen et al. 1998] Soininen, T., Tiihonen, J., Männistö, T. and Sulonen, R., Towards a General Ontology of Configuration, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998/12)*, pages 357-372, Cambridge University Press, USA, 1998.

[Thiel et al. 2001] S. Thiel, S. Ferber, T. Fischer, A. Hein and M. Schlick, A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems, in *Proc. of In-Vehicle Software 2001*, pages 43-55, Detroit, Michigan, USA, March 5-8, 2001.

Contact

Katharina Wolter
Universität Hamburg, FB Informatik
Vogt-Kölln-Str. 30
22527 Hamburg, Germany
tel: +49 (0)40 42883 2610
email: kwolter@informatik.uni-hamburg.de

Lothar Hotz is a researcher at the Hamburger Informatik Technologie Center (HITeC) located at the University of Hamburg. He participated in several projects related to topics of configuration, knowledge representation, constraints, diagnosis, qualitative simulation, parallel processing and object-oriented programming languages.

Thorsten Krebs is a researcher at the Laboratory for Artificial Intelligence (LKI) at the University of Hamburg. He has participated in developing the configuration tool *EngCon* at the Centre for Computing Technologies (TZI) at the University of Bremen.

Katharina Wolter is a researcher at the Laboratory for Artificial Intelligence (LKI) at the University of Hamburg, where she also did her masters thesis. Her primary research interests are in the area of knowledge-based configuration and human-computer interaction.