Model-based Configuration Support for Product Derivation in Software Product Families

Thorsten Krebs[1], Katharina Wolter[1], Lothar Hotz[2]

[1]LKI, Universität Hamburg,
Vogt-Kölln-Str. 30, 22527 Hamburg
{krebs, kwolter}@informatik.uni-hamburg.de

[2]HITeC e.V., Universität Hamburg,
Vogt-Kölln-Str. 30, 22527 Hamburg
hotz@informatik.uni-hamburg.de

Abstract. Software Product Families can be used for software mass customization. One major problem within family-based software engineering is the lack of methodological support for application engineering. The large number of decisions and dependencies between these decisions make the task complex and error-prone. Impacts of decisions are not known or overseen during application engineering. Functionality is implemented anew where reuse would have been possible because the large number of artifacts is hardly manageable. The methodology described in this paper combines the well-known research areas of software product families and model-based configuration in order to fill this gap.

The methodology is based on a configuration model that represents functionality and variability provided by the product family. Basically, the configuration model provides two layers of configurable assets, i.e. a feature layer and an artifact layer. The artifact layer reflects the (variable) structure of the product family artifacts and the feature layer is the customer view on the functionality in the artifact layer. A mapping between the feature layer and the artifact layer allows for automated inferring of the needed software artifacts for a given selection of features. We call the resulting process of application engineering enhanced with automated inferences *model-based product derivation*.

In an ideal product derivation process the features required by the customer are selected and the configuration system automatically infers all artifacts needed to provide these features. However, requirements may not be accounted for in the shared product family artifacts and can only be accommodated by adaptation or even new development. This involves adapting the product (family) architecture and / or adapting or creating component implementations. Since the software artifacts are represented in the configuration model, these modifications have to be aligned in a synchronized fashion. To support this, we developed a dependency analysis that can be performed on the model.

Introducing the methodology in selected industrial environments dealing with software product families shows that it is applicable and can be tailored to the specific needs of particular organizations. In this paper we describe first experiences made by using the methodology in industrial environments.

1 Introduction

The topics of model-based configuration from the area of artificial intelligence and mass customization have a lot in common. To analyze their relation more closely definitions for the two terms are needed. *Model-based configuration* supports the composition of products from several parts (Günter 1995) and will be introduced in more detail in Section 2.2. For the term *mass customization* several different definitions are used as Piller (2003) states. Davis refers to mass customization when "the same large number of customers can be reached as in mass markets of the industrial economy, and simultaneously they can be treated individually as in the customized markets of pre-industrial economies" (1987, p. 169). The commonality between the two research areas is the customer-specific product. Model-based configuration support can be used to derive customized products. However, the method is not restricted to mass-market products. It is also applied successfully for complex capital goods (aircraft, drive systems, etc.). Additionally, the method is more powerful than required by the customization problem, i.e. it can be used to solve more complex problems.

According to Günter (2002) three different sales scenarios can be distinguished (first published in German in Günter (2001)):

- *Click & Buy*: Only non-customizable products are offered and thus no configuration support is needed in order to ensure the consistency of products.
- *Customize & Buy:* Products can be customized but the number of valid combinations of components is still restricted. Thus, complexity and consistency problems still play a minor role.
- *Configure & Buy:* The vast number of possible combinations of components and the complex restrictions lead to a serious complexity problem. In this scenario it is not possible to list all possible products in a catalog. Model-based configuration support is needed for assembling valid products.

Similar categories are described by O (2002).

The methodology described in this paper does not fit to one of the above-introduced scenarios. Even in the last scenario a product can only be assembled from the components modeled so far. It is not possible to meet customer requirements that have not been taken into account during developing the product family. In order to meet "new" customer requirements it might be necessary to adapt existing or develop new components. Our methodology integrates these steps and the product configuration process more closely. To complete the above-described set of scenarios we add the following:

- *Configure, Develop & Buy:* This allows for assembling a product from components in the asset store and may also include components not yet developed. The main benefit is that it is possible to meet customer requirements that have not been taken into account yet. In addition to the complexity problem described for Configure & Buy in this scenario it is necessary to integrate the new components and their relations into the existing model. The facilities of known standard configuration techniques are not sufficient to fulfill this task.

With this scenario we move a step towards truly customized products. However, since the newly developed components are integrated into the product family they can be efficiently reused for all future customers.

Customization is an important topic not only for hardware products but also for software products or software-intensive systems consisting of both hardware and software. 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. Offering more and more functionality inevitably leads to the Configure & Buy scenario. Selling functionality not yet implemented (e.g. because of new customer requirements) leads to Configure, Develop & Buy. The methodology described in this paper supports this last scenario.

ConIPF (Configuration in Industrial Product Families) 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 first experiment results and experiences with our product derivation methodology. In order to validate the methodology we applied it at our industrial partners. The first results are promising and are presented later in the paper.

The remainder of this paper is organized as follows: in Section 2 we introduce the approaches our methodology is based on. Furthermore, we present the configuration tools KONWERK and EngCon, which we used to implement the methodology at our industrial partners. In Section 3 the application domain from one of our industrial partners is described. This domain is used as a guiding example. Section 4 introduces the methodology in detail. First, the configuration model we use is introduced. Second, the product derivation process is described and finally the aspect of adapting existing and developing new components is taken into account. In Section 5 we present experiments performed to validate the methodology and our experiences so far. In Section 6 we discuss related work and give a conclusion in Section 7.

2 Basic Technologies

In this section we briefly introduce the approaches our methodology is based on: i.e. software product lines (Section 2.1) and model-based configuration (Section 2.2). Additionally, in Section 2.3 we present model-based configuration tools used to implement and validate the methodology described in this paper.

2.1 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 modeled in partonomies with mandatory, optional and alternative properties.
- In application engineering existing artifacts are used to assemble specific products by analyzing requested features, selecting architecture and adapting components.

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. Application engineering is currently realized by communication facilities between developers and with standardized documents in order to capture customer requirements or to define system specifications, and to realize change management. However, a general methodology for realizing or supporting application engineering does not exist (Hein et al. 2003). Therefore, it is common to use previously developed products or platforms by a "copy and modify approach" to suit current customer needs. 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.

Our methodology fills this gap by supporting application engineering with methods from model-based configuration (see next section). The resulting application engineering process is called *model-based product derivation*.

2.2 Model-based Configuration

Configuration is a well-known approach to support 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 modeling 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 known from the customer that a product must fulfill.

The configuration itself is performed in an incremental approach, where each step represents a configuration decision and possibly includes testing, simulating or checking with constraint techniques (Günter 1995, Hotz et al. 2003). However, applying configuration methods to software systems is in an early stage. First approaches are e.g. described in Soininen et al. (1998).

2.3 Tools

Two configurators have been used to implement, validate and improve the methodology: KONWERK and EngCon. KONWERK is a configuration tool

developed partly at the University of Hamburg. EngCon is a powerful, scalable and flexible configuration platform from encoway GmbH (www.encoway.de). EngCon was used in the project ConIPF to implement and validate the product derivation methodology at both industrial partners (see also Section 5). It was also extended with new functionality that has been discovered as necessary for deriving software products. KONWERK was used to identify and perpetuate further research topics. Both tools are briefly introduced in the following:

- *KONWERK* is a kernel system for configuration tasks. A goal was the development, testing and operational introduction of the architecture of knowledge-based systems for configuration and construction of technical systems. The application of knowledge-based methods was mostly applied for problems of routine configuration, but is not limited to this. KONWERK is a modular system intended to make more areas of knowledge-based configuration (like optimization, spatial configuration, and support for vague modeling) accessible. Using Common Lisp, Common Lisp Object Systems (CLOS), mainly did the implementation and its meta-object protocol (see Kiczales et al., 1991). The user interface was implemented with the Common Lisp Interface Manager (CLIM). Thus, it is portable and runs under Windows and SunOS.
- *EngCon* is methodologically based on KONWERK. But in detail there are various differences and extensions, as e.g. further described in Hollmann et al. (2000). EngCon has a modern and flexible component architecture based on Java technology. Encoway provides a web-based modeling environment K-Build, a powerful graphical GUI builder K-Design and more development and connectivity tools (e.g. for SAP and other ERP, PDM and CRM systems) for the flexible customization of a configuration application. The configuration platform EngCon can be used in offline and online configuration scenarios. The user can configure either user-controlled or by using a configuration wizard (Ranze et al. 2002).

3 Application Domain

In this section we introduce the application domain used as a guiding example throughout the following sections of this paper. We implemented and tested the methodology at Robert Bosch GmbH (www.bosch.de) in the research unit Automotive Electronics that develops Car Periphery Supervision systems.

Car Periphery Supervision (CPS) systems monitor the local environment of a car. CPS systems are automotive systems based on sensors installed around the vehicle. The recording and evaluation of sensor data enables different kinds of high-level applications that can be grouped into safety-related and comfort-related applications (see Thiel et al., 2001).

Two examples are given in the following:

- *Pre-Crash Detection (PCD)*: Based on sensor information it is possible to estimate the time, area and direction of an impact before the crash happens. This enables adjusting trigger points of specific airbags (Pre Set) in different locations in the car

and firing a (seat) belt tensioner (Pre Fire) appropriately for the estimated crash situation.

- *Parking Assistance (PA)* supports the driver in avoiding moving the vehicle against people or stationary objects while driving slowly backward or forward. This is especially useful for vehicles that are difficult to look over.

4 The Methodology

After introducing the basic approaches our methodology is based on and the application domain of one of our industrial partners, in this section we describe our product derivation methodology in detail. The methodology is based on a configuration model where all different types of configurable assets and their relations are formalized. This configuration model is introduced in Section 4.1. The model-based product derivation process is explained in Section 4.2. In Section 4.3 we address how necessary modifications for existing assets and new development can be integrated with product derivation.

4.1 Configuration Model

Model-based configuration provides means for modeling and reasoning with configurable assets. Traditionally these are *hardware artifacts*. When considering software product lines and software-intensive systems, also *software artifacts* and *features* need to be modeled. Additionally, we identified that the *context* in which the software-intensive system will be used is of particular importance. This context can influence the set of possible solutions although it is not part of the system itself. In the CPS domain, for example, some systems cannot be used in Europe or Northern America because of legal restrictions. Furthermore, the weather conditions of some areas can restrict the choice of hardware artifacts.

Since these four asset types (features, context, hardware and software artifacts) are common to most domains of software-intensive systems we defined them and their relations in a Commonly Applicable Model (CAM). A product, i.e. the result of the product derivation has software and hardware artifacts as parts and these together realize certain features. For more details about the CAM see Krebs et al. (2004).

To implement the methodology in a specific domain the CAM is extended with domain-specific knowledge about hardware and software artifacts, the existing features and so on. The CAM and the domain specific assets are defined in the *configuration model* as *concepts*. In the following we give a description of how concepts are defined in the model-based configuration approach:

- Each concept has a *name* that represents a uniquely identifiable character string. An example from the CPS domain is the concept called `Parking Assistance`.
- Concepts are related to other concepts in the taxonomic relation. A concept has exactly one *superconcept* and possibly several subconcepts. The concept `Parking Assistance` has the superconcept `Feature` that is part of the CAM.

- Attributes of concepts can be represented through *parameters*. A parameter is a tuple consisting of a name and a value descriptor. Diverse types of domains are predefined for value descriptors – e.g. integers, floats, strings, sets and ranges. The concept Parking Assistance has one parameter called Symmetry with two possible values: `Passenger Side` or `Passenger and Driver Side`.
- Partonomies are generated by modeling *compositional relations*. Such a relation definition contains a list of parts (i.e. other concept definitions) that are identified by their names. Each of these parts is assigned with a minimum and a maximum cardinality that together specify how many instances of these concepts can be instantiated as parts of the aggregate. The CAM defines different compositional relations, e.g. `has Features`, `has Hardware`, `has Software` to emphasize on the difference between artifacts and features.

Further relations between assets that do not describe a hierarchical structure like taxonomic and compositional relations do are called *dependency relations*. A *uni- or bidirectional* dependency can be defined between concepts in arbitrary places of the ontology. Examples for this relation type are the *requires* relation for unidirectional und the *excludes* relation for bidirectional dependencies. The *realizes* relation for example is a bidirectional dependency expressing that features are realized by artifacts in an *n*-to-*m* mapping: one feature can be *realized by* one or more artifacts and the other way round.

A clear separation of potential features and artifacts of the product family on the one side and features and artifacts that are used for a specific product derivation on the other side is needed. In model-based configuration tools like KONWERK (see e.g. Günter and Hotz 1999) this is realized by distinguishing between *concepts* for describing features and artifacts in general and *concept instances* for describing features and artifacts chosen for a specific product.

4.2 Product Derivation Process

In this section we explain how software-intensive systems are derived using our methodology. One important difference between the configuration of hardware systems and software-intensive systems is that with software one can realize the product within the derivation process (i.e. compile source code, calibrate the product, etc.). Usually, the result of a configuration process is an abstract description of a hardware product. Our model-based product derivation process addresses the entire software development process. Thus, selecting features, architecture and hardware and software artifacts are part of the derivation process just like the realization of the software itself. Therefore two types of activities are involved in model-based product derivation: *configuration* and *realization*.

Configuration activities consist of making decisions about the desired product based on the customer requirements (see Section 4.1.2). A configuration tool is used to ensure a consistent, complete and correct solution. Realization activities produce the software product and other results like a specification document. This is done by selecting artifacts from the asset store, generating new artifacts, compiling new artifacts and compiling the product (see Section 4.2.2).

Configuration and realization do not describe chronological activities. In a typical product derivation process they will rather alternate. For example, once all the features have been selected which is a configuration activity the first realization step can take place, i.e. the generation of a specification document.

4.1.2 Configuration

As stated above, a configuration tool is used to support the configuration of software-intensive systems. One benefit of the configuration system is that it keeps track of the necessary decisions and the currently possible values for these decisions. A further benefit is the computation of values for decisions based on the configuration model. Typically, the user starts by selecting features that represent the functionality of the desired product. In a feature-oriented approach decisions can be made on a more abstract level and by using customer-understandable terms. As soon as the user has made the first decision, the configuration system starts to compute the impacts of these user decisions. Based on the *is realized by* relation between features and software / hardware artifacts the configuration tool infers the needed artifacts. Thus, in the model-based product derivation process the user makes some decisions and others are inferred by the configuration system.

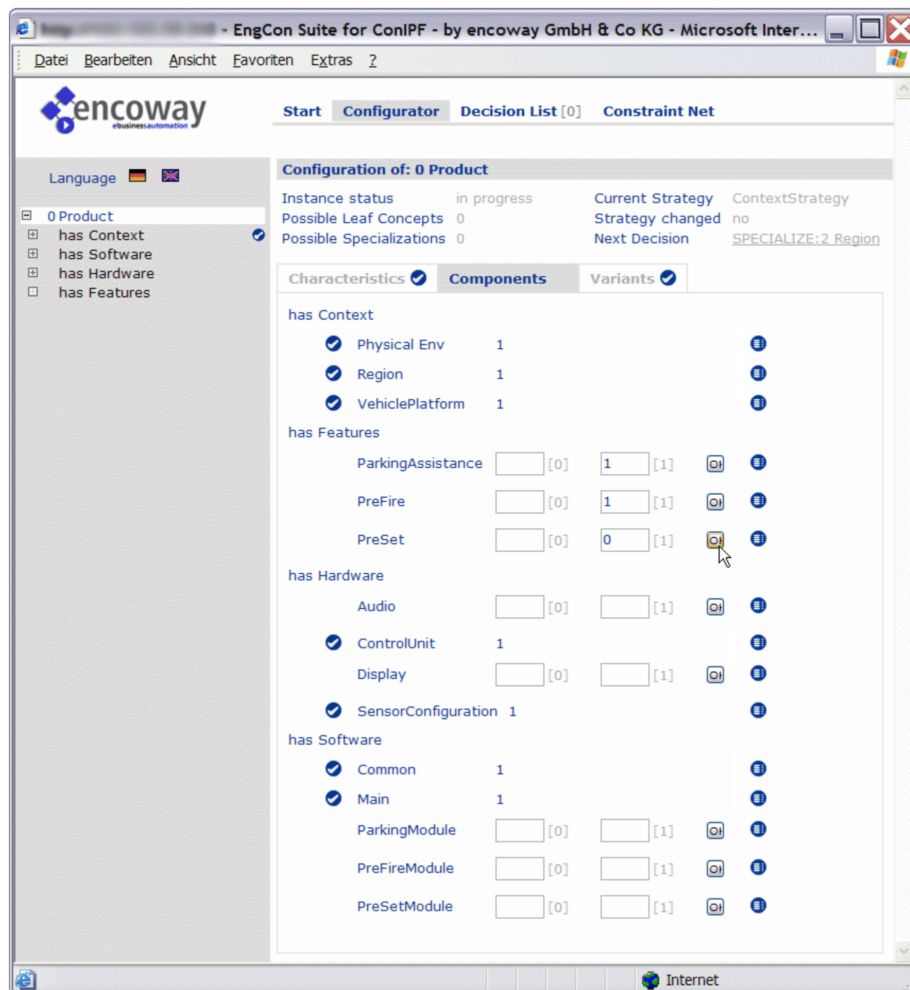

Figure 1: Selection of Features for a CPS System

To illustrate the product derivation process two screen shots of the ConIPF prototype application are given in Figure 1 and Figure 2. Usually, a configuration tool is extended with a domain-specific user interface when integrated in an industrial context. For our purposes the tool vendor encoway customized a user interface that reflects our methodology. On the left side the product derived so far is given in a tree. This tree is structured according to the relations defined in the CAM, e.g. `has Feature`, `has Context` and so on. On the right side possible decisions are given. The corresponding part of the solution is highlighted in the tree. The relations defined in the CAM are used to structure the decisions. Three optional features are displayed (`Parking Assistance`, `Pre Set` and `Pre Fire`).

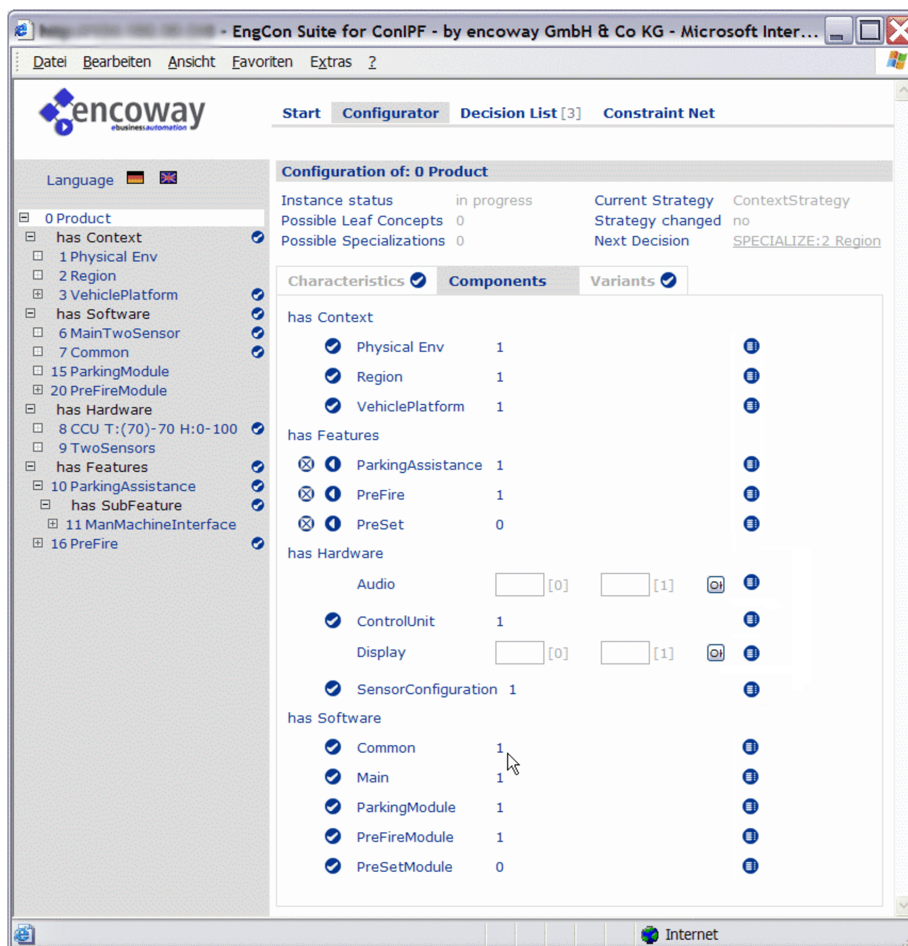

Figure 2: Software Artifacts inferred by the Configuration Tool

An example scenario from the CPS domain is: the user starts by selecting the features Parking Assistance and Pre Fire and decides not to include Pre Set (see

Figure 1). Figure 2 shows that the values for the selected features have been set and the configuration tool has inferred the corresponding software modules Parking Module (indicated by ID 15) and Pre Fire Module (ID 20) and the feature Man Machine Interface (ID 11) to be part of the CPS system.

4.2.2 Realization

EngCon produces exportable XML output that contains a description of the configuration solution. This file contains all information about all assets that have been configured; incl. context, features, and hard- and software artifacts. In Table 1 we show a sample output of the software artifacts that have been configured. The `Product` is the aggregate and references its parts via the `has Software` relation. Every asset instance has an ID and cardinalities. The `Pre Fire` Module e.g. was not selected and therefore has the cardinality 0 here.


```
- <modelInstance concept="Product" creationConcept="Product" id="0">
  - <decompositionRelationInstance name="has Software"
        definitionName="Software">
  - <decompositionElementInstance min="1" max="1" conceptName="Main">
    <instanceReference modelInstanceID="6" />
      </decompositionElementInstance>
  - <decompositionElementInstance min="1" max="1"
        conceptName="Common">
    <instanceReference modelInstanceID="7" />
      </decompositionElementInstance>
  - <decompositionElementInstance min="1" max="1"
        conceptName="ParkingModule">
    <instanceReference modelInstanceID="21" />
      </decompositionElementInstance>
    <decompositionElementInstance min="0" max="0"
        conceptName="PreFireModule" />
  - <decompositionElementInstance min="1" max="1"
        conceptName="PreSetModule">
    <instanceReference modelInstanceID="19" />
      </decompositionElementInstance>
      </decompositionRelationInstance>

    ...

  </modelInstance>
```


Table 1: (Part of a) XML configuration Output

The hard- and software artifacts are the assets that have to be assembled to generate the product. To automatically filter the relevant information from the XML structures, a XSL script can be defined which uses the concepts and relations specified in the CAM. Output of that XSL transformation is a list of software artifacts. The source files and their version numbers are extracted from a configuration management system. After compiling the source code the CPS system can be run on the target platform. In our experiments at Robert Bosch GmbH we used a demonstrator playing sample data that was recorded in a test car. Different asset combinations and different

parameter values for selected assets can be tested and compared directly. More detail about the results of our experiments is provided in Section 5. For more details about the product derivation process see Wolter et al. (2004).

4.3 Adaptation of Artifacts

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 an incompatibility is recognized, e.g. when the compilation cannot be executed successfully. Backtracking mechanisms can be applied to take back decisions and their inferences and then try different input for those configuration decisions that led to the conflict. But 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 situations existing assets (and their corresponding description in the configuration model) have to be modified during the derivation process.

In contrast to technical application domains, for configuring software products, even when existing software artifacts are reused, often modifications on those artifacts are often needed. 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 modeling 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 product family, i.e. modeling 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 model-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 modeled 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 (Krebs et al. 2004). A *dependency analysis* has been implemented to compute a graph showing the impacts a modification has. This graph can be used to set up a to-do list containing those assets that have to be modified in order to come to a consistent set of modifications to fulfill the evolution task.

5 Experiments and Experiences

Both industrial partners have chosen a business unit to apply and validate the methodology. Robert Bosch GmbH has applied the methodology in the area of the CPS domain to develop manufacturable prototypes. Thales Naval Nederland uses the new generation of Combat Management System with the aim of efficiently managing the complexity of the large number of interdependent subsystems. Both partners instantiated the methodology described in this paper, set up configuration models using the modeling facilities defined in Section 4.1 and completed diverse product derivations that led to concrete software products and could be tested on demonstration platforms.

During these experiments we made the following experiences: the Commonly Applicable Model can be used to model product lines for software-intensive systems. The asset types we identified are sufficient to model all aspects needed for automated product derivation support. During the project subsystems have been identified as helpful for modeling product lines. A subsystem can comprise multiple hard- or software assets that are used together leading to a more efficient reuse of frequently used asset compositions (see also Krebs et. al. 2004a). The methodology can be introduced in different business units and is tailorable for their specific needs. This means it is possible to only use parts of the defined modules (e.g. leave the evolution of assets out) or exchange tools, depending on what kinds of tools or representations are used in that business unit. The methodology is scalable: while at Bosch we modeled a rather small but complex domain containing about 75 assets and 35 constraints, at Thales the configuration model contains about 700 assets and 250 constraints. Two major aspects have been observed concerning modeling the product line: (1) modeling and keeping an overview over the asset store is getting harder; depending on the size of the configuration model, and (2) different possibilities to model the same aspect (e.g. modeling a requires relation as a compositional relation or as a constraint) make modeling a non-trivial task.

The model-based product derivation process consisting of configuration and realization activities shows that software products can be reliably assembled. Both industrial partners built multiple products with different feature and artifact selections that could be built and run on demonstration platforms.

A stable product family existing beforehand makes it easier to instantiate our methodology. Setting up a product family, modeling all configurable assets and their interdependencies are time-consuming tasks that need a lot of effort. The dependency analysis we introduced (and described in Section 4.3) can help in improving the overview of necessary modifications to components in the asset store and the configuration model. This means, its use is not restricted to application engineering but can also be applied for extending the asset store during domain engineering.

6 Related Work

Well-known approaches for family-based software development are PuLSE (Bayer et al. 1999), KobrA (Atkinson et al. 2000) both developed at the Fraunhofer IESE and

FAST (Weiss et al. 1999). The Software Product Line Practice Framework (see SEI) developed at the Software Engineering Institute at the Carnegie Mellon University (SEI) provides a collection of successful approaches rather than a particular methodology. In contrast to our methodology KobrA dose not provide a mapping between features and artifacts. Thus, the knowledge about how to map a specific requirement to software and hardware artifacts is not formalized in this methodology. Using PuLSE this knowledge is formalized but no tool support is given for automatically derive the needed artifacts for a selected feature (Bayer et al. 2000).

Other configuration tools that specialize on software mass customization or family-based software development are GEARS developed by BigLever Software (www.biglever.com) and pure:variants provided by pure-systems (www.pure-systems.com).

7 Conclusion

In this paper we have presented a model-based product derivation methodology for application in software-intensive domains. We have introduced the approaches our methodology is based on (i.e. software product lines and model-based configuration) and have discussed similarities between model-based configuration and mass customization. We have further shown how product derivation can benefit from a combination of these approaches. The most important aspect of our methodology and also a novelty in this research field is the combination of tool-supported configuration and realization into one process for deriving software products. We also presented first experiences gained from the experiments carried out at our industrial partners from the ConIPF project.

References

Atkinson, C., Bayer, J., & Muthig, D., (2000). Component-Based Product Line Development: The KobrA Approach, *Proc. of 1st International Software Product Line Conference*. Pittsburg, USA.

Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., & Debaud, J.M. (1999). PULSE: A Methodology to Develop Software Product Lines, *Proceedings of the 5th Symposium on Software Reusability*.

Bayer, J., Gacek, C., Muthig, D., & Widen, T. (2000). PuLSE-I: Deriving Instances from a Product Line Infrastructure. In *7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, (pp. 237-245). Edinburgh, Scotland.

Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, H., & Pohl, K. (2001) Variability Issues in Software Product Lines, *Proc. of the Fourth International Workshop on Product Family Engineering(PFE-4)*. Bilbao, Spain.

Davis, S. (1987). *Future Perfect*. Reading: Addison-Wesley.

Günter, A. (1995). *Wissensbasiertes Konfigurieren*, Infix, St. Augustin.

Günter, A., & Hotz, L. (1999). KONWERK – A Domain Independent Configuration Tool. *Proc. of Configuration (AAAI 1999 Workshop) (pp. 10-19).* Orlando, Florida.

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

Günter, A., Hollmann, O., Ranze, K. C. & Wagner, T. (2001). Wissensbasierte Konfiguration von komplexen variantenreichen Produkten in internetbasierten Vertriebsszenarien. *KI*, 15(1), 33-36.

Hein, A., MacGregor, J., & Thiel, S. (2001). Configuring Software Product Line Features, In: *Proc. of ECOOP 2001 - Workshop on Feature Interaction in Composed Systems*, Budapest, Hungary.

Hein, A., & MacGregor, J. (2003). Managing Variability with Configuration Techniques, *Proc. of the Workshop on Software Variability Management at the ICSE*, Portland, Oregon, USA.

Hollmann, O., Wagner, T., & Günter, A. (2000). EngCon: A Flexible Domain-independent Configuration Engine. *Proc. ECAI-Workshop Configuration* (pp. 94).

Hotz, L., Günter, A., & Krebs, T. (2003). A Knowledge-Based Product Derivation Process and some Ideas how to Integrate Product Development (position paper*), Proc. of Software Variability Management Workshop* (pp. 136-140). Groningen, The Netherlands.

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

Kang, K., Lee, J., & Donohoe, P. (2002) Feature-oriented Product Line Engineering. *IEEE Software, 7/8* (pp. 58–65).

Kiczales, J., Bobrow, D. G., & des Rivieres, J. (1991). *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA.

Krebs, T., Wolter, K., & Hotz, L. (2004). Mass Customization for Evolving Product Families, Proc. of International Conference on Economic, Technical and Organizational Aspects of Product Configuration Systems (pp.79-86). Copenhagen, Denmark.

Krebs, T., Hotz, L., & Wolter, K. (2004a). Pre-Packaged Variability for Product Derivation in Product Lines, Proc. of Configuration -- ECAI 2004 Workshop (pp. 3-1 - 3-3). Valencia, Spain.

O, Ying-Lie (2002). Configuration for mass-customization and e-business. 15th European Conference on Artificial Intelligence, Configuration Workshop.

Piller, F. T. (2003, May). What is mass customization. *Mass customization news*, 6(1). Retrieved March 21, 2005, from http://www.mass-customization.de/news/news03_01.htm

Ranze, K., Scholz, T., Wagner, T., G¨unter, A., Herzog, O., Hollmann, O., Schlieder, C., & Arlt, V. (2002). A Structure-based Configuration Tool: Drive Solution Designer DSD. *14. Conf. Innovative Applications of AI.*

SEI (Carnegie Mellon Software Engineering Institute). *A Framework for Software Product Line Practice*. Retrieved March 21, 2005, from http://www.sei.cmu.edu/plp/framework.html

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

Thiel, S., Ferber, S., Fischer, T., Hein, A., & Schlick, M. (2001). A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems, *Proc. of In-Vehicle Software 2001 (SP-1587)* (pp. 43-55). Detroit, Michigan, USA.

Weiss, D., & Lai, C.T.R. (1999). Software Product Line Engineering: A Family-based software Development Process, Addison Wesley.

Wolter, K., Krebs, T., Hotz, L., & Meijler, T.D. (2004). Knowledge-based Product Derivation Process, Proc. of the IFIP 18th World Computer Congress TC12 First International Conf. on AI Applications and Innovations (AIAI2004/WCC2004) (pp. 323-332). Toulouse, France.