KNOWLEDGE-BASED PRODUCT DERIVATION PROCESS

Katharina Wolter, Thorsten Krebs
LKI, Fachbereich Informatik, Universität Hamburg
Hamburg, Germany
{kwolter|krebs}@informatik.uni-hamburg.de

Lothar Hotz
HITeC c/o Fachbereich Informatik, Universität Hamburg
Hamburg, Germany
hotz@informatik.uni-hamburg.de

Theo Dirk Meijler
Department of Mathematics and Comupter Science, Rijksuniversiteit Groningen
Groningen, The Netherlands
t.d.meijler@bdk.rug.nl

Abstract Software development in general is a design task. Software product lines divide this task into two processes: during domain engineering artifacts are developed for reuse while during application engineering these artifacts are assembled to build software products. Thus, application engineering is similar to a configuration task. In this paper, we describe how structure-based configuration can be used to support application engineering. The presented process extends efficiency of application engineering and enhances the quality of the results.

Keywords: Structure-based configuration, product lines, derivation process

1. Introduction

Reuse is an important means of reducing development costs and time and of increasing quality for software systems. Software product lines introduced in software engineering provide a successful approach to enhancing reuse. They are used to develop a product family, i.e. a set of systems that "share a common, managed set of features satisfying the specific needs of a particular customer or market segment" (Bosch et al., 2001). The product line approach divides the development of systems into two processes: domain engineering and application engineering. In domain engineering reusable artifacts are developed for the concerned product family. Commonalities and variability in the considered product space can be described by means of feature models. In application engineering products are composed using these feature models and artifacts. This process of assembling artifacts by selecting the required features is called product derivation.

While there are different methods of supporting domain engineering in software product lines there is a lack of support for application engineering (see e.g. Hein and MacGregor, 2003). Usually, artifacts are combined in a 'copy and modify approach' based on former cases - i.e. old products. The large number of decisions and dependencies between these decisions make product derivation complex and error-prone: impacts of decisions are not known or are overlooked, functionality is implemented anew where reuse would have been possible because the large set of artifacts is hardly manageable.

The work described in this paper is part of the ConIPF project. One goal of this project is to fill the methodology gap described above. We describe how structure-based configuration form AI can be used to support the process of product derivation in software product lines. The main benefit of using mechanisms from structure-based configuration is to automatically derive needed artifacts form product features and thus, make the variability manageable. The resulting process is called knowledge-based product derivation process (KBPD-process).

The remainder of this paper is organized as follows: in Section 2 methods that form the basis of the KBPD-process are introduced and an example domain is described. Next, we describe the KBPD-process (Section 3) and we show how this process can be mapped to structure-based configuration (Section 4). Finally a discussion of related work is given and a short summary concludes the paper.

2. Background

Software Product Lines

In domain engineering, exploiting commonality and managing variability are necessary and can be achieved by using feature models (see Kang et al., 2002). Features are "prominent or distinctive user-visible aspects of a system[...]" (Kang et al., 1990). In application engineering, existing artifacts are used to assemble products by analyzing requested features and selecting an architecture. Domain engineering and application engineering do not describe chronological tasks, but the distinction between developing a product line and developing products using this product line.

Structure-based Configuration

Configuration supports the composition of complex products from several parts. Different approaches to configuration can be distinguished (see e.g. Stumptner, 1997; Sabin and Weigel, 1998). The work presented here is based on structure-based configuration (see Günter and Cunis, 1992; Günter, 1995). The knowledge needed to support configuration is defined in a configuration model. Basic modeling facilities enable the differentiation between three basic knowledge types:

- Conceptual knowledge includes concepts to model domain objects, taxonomic and compositional relations between domain objects, as well as restrictions between arbitrary concepts (constraints).
- Procedural knowledge declaratively describes the configuration process – i.e. the order in which configuration decisions are processed.
- A Task specification describes the desired product.

The configuration process is performed incrementally. Each step represents a configuration decision and the calculation of its logical impacts. After each step, global mechanisms can be applied e.g. for consistency checking. Global means that the entire partial configuration is examined. The result of a configuration process is a complete configuration: i.e. a description of the created solution. During the configuration process situations can occur where the decisions made by the user, their logical impacts and the configuration model are not consistent. Such a situation is referred to as a conflict.

Example Domain

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 (see Thiel et al., 2001) like pre-crash detection or parking assistance:

Pre-Crash Detection. Based on sensor information it is possible to estimate the time, area and direction of an impact before the crash happens. This enables e.g. adjusting trigger points of airbags in different locations in the car appropriately for the estimated crash situation.

Parking Assistance. The driver is supported in avoiding moving the vehicle against people or stationary objects while driving slowly.

Requirements for the Product Derivation Process

The most important requirements that an industrial realistic KBPD-process must meet are: synchronization of configuration process and realization process, supporting the user in traversing the configuration decisions, allow for process tailoring, integration of calibration, supporting efficient change of earlier decisions and connection of KBPD-process with evolution. In the following we give details for the requirements essential for the topic of this paper.

Synchronization of configuration and realization. The result of an application engineering process and software development in general is a concrete software artifact (i.e. a product) whereas configuration creates an abstract description of a product. Therefore two processes are involved in knowledge-based product derivation: configuration and realization. Since these two processes are dependent on each other they need to be synchronized.

Guidance in proceeding. The product derivation process consists of a large number of decisions with several dependencies. In order to manage the complexity of the process it should be structured into subtasks according to the user's idea of the task. Furthermore, in order to find a solution and to avoid conflicts it is on the one hand reasonable to make those decisions first that most restrict the solution space. On the other hand it is not appropriate in most cases to force a procedure on the user because explorative behavior is important in order to create a solution. In design people do not usually proceed according to the strategy of stepwise refinement with a top-down breadth-first decomposition(see Guindon, 1990). Therefore, a flexible process is needed in which the user can make decisions, check their effects on other decisions and then possibly change decisions as (s)he likes. A flexible process is also very important when the product derivation is done in a sales conversation with the customer.

3. The KBPD-Process

The KBPD-process assumes that a product line has already been set up - i.e. software artifacts already exist and their capabilities are described by means of features. The features, the available artifacts and the relations between these two are modeled in a configuration model. Based on a "is implemented by" relation between features and artifacts it is possible to infer the required software and hardware artifacts after the user has selected a feature (see Figure 1). In the CPS domain, for instance, the user can select features like parking assistance on the feature level instead of sensors and applications on a lower level of abstraction.

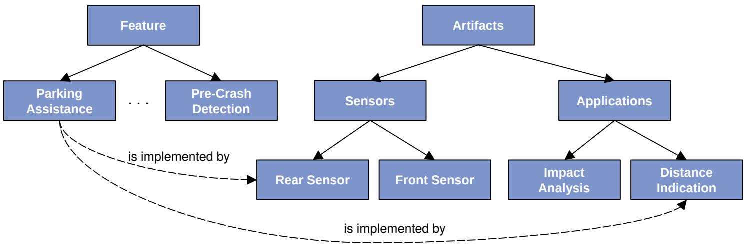

Figure 1. Features and Artifacts modeled in the configuration model.

The scenario of use for the KBPD-process looks as follows: at least the first part of the product derivation is done by a sales person, possibly during a conversation with the customer. The derivation might be completed later by one or more other persons who might work in other departments. Results of the KBPD-process are the concrete software artifact as well as a description of the entire solution. This description specifies the software part of the solution and also the hardware and the realized features.

The KBPD-process is based on three main concepts: phases, roles and activities. Phases are used to structure the product derivation on a high level according to the workflow in an organization. In our experiences the product derivation is subdivided into the phases: acquisition, engineering, calibration. Roles define persons with specific responsibilities and abilities and are used to define who is supposed to carry out which part of the product derivation. The following roles need to be distinguished: sales person, application engineer, integrator. Activities are used to structure the work within one phase into subunits that are familiar to the user. In the following we first describe phases and next activities are discussed.

Phases

In Figure 2 the KPBD-process and its three phases are illustrated. In the first phase, the acquisition phase, a sales person together with the customer derives a first partial configuration corresponding to the customer needs. Most decisions in this phase are on a high level of abstraction e.g. about features or the target environment. However, based on the "is implemented by" relation many decisions about technical details can already be inferred by the configuration system. During the engineering phase the number of nodes or control units needed and the assignment of processes to these is determined. Furthermore, technical aspects that are not of direct interest to the customer or that require specific knowledge about the realization can be decided. Finally, in the calibration phase a set of parameter values that make the software function optimally in its run-time environment is iteratively determined.

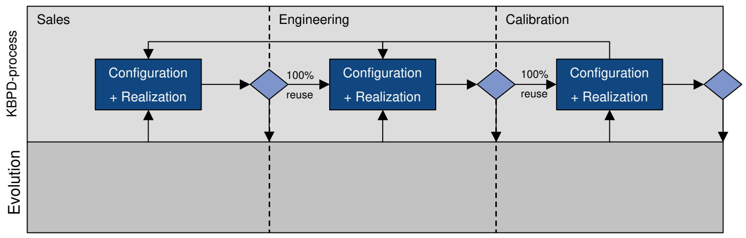

Figure 2. Overview of the KBPD-process and its relation to evolution.

The KBPD-process handles the best case where 100% of reuse is possible. If, on the other hand, a selected feature or feature combination cannot be realized based on the available artifacts a conflict occurs. The user can then either change the decisions made so far (upper arrows back to earlier phases in Figure 2) or the set of realizable products has to be extended (illustrated by the arrows downwards). The last case is called evolution and requires the adaptation of the configuration model and development and/or adaptation of artifacts.

Activities

Activities are used to structure the work within one phase. Two different activities can be distinguished: configuration activities and realization activities. Configuration activities are used to group the large number of decisions into units that belong together from the users point of view. Realization activities produce results based on decisions made so far. Each activity is assigned to exactly one phase. In Figure 3 a more detailed image of the KBPD-process is given. The sales phase consists of five configuration activities and one realization activity. The Pre-select activity is used to group together those decisions that strongly restrict the solution space in the considered domain and that therefore should be made in the very beginning. In Environment restrictions caused by the environment of the product are defined. Examples in the CPS domain are the size and shape of the car. Configure environment is about external aspects which are not part of the product. In contrast to this, Hardware and Software Constraints consists of constraints regarding the hardware

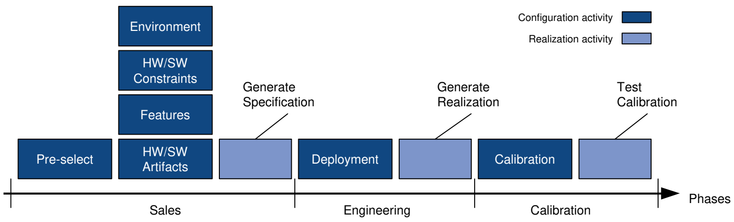

Figure 3. Activities in the KBPD-process.

and software artifacts of the product. In the CPS domain for instance, one could specify that a certain kind of sensor should be used. In the Features activity, the main capabilities and attributes that the product should provide are determined. An example is the selection of features like pre-crash detection or parking assistance in the CPS domain. In contrast to all activities described so far Hardware and Software Artifacts consists of decisions on the technical level. One main objective is to infer as many decisions as possible on this lower level from the decisions of the other activities described earlier. Based on these activities a first product specification can be generated in the realization activity: Generate Specification.

In the engineering phase there is only on configuration activity: Deployment which has already been described above in the description of the engineering phase. When this activity is completed the activity Generate realization can be executed. Finally, in the calibration phase there is again only one configuration activity: Calibration. Based on the parameter values specified in this activity the last realization activity is used to initiate the test of the product in its run-time environment.

Since realization activities are always based on decisions it is necessary to ensure that these decisions have already been made when the realization is started, i.e. configuration process and realization process need to be synchronized. The order of activities is from the left to the right in Figure 3, where activities in the same position on the phases axis can be done in parallel, i.e. the user can switch between them. Thus, as much flexibility as possible is offered. To provide guidance the activities described above should be displayed to the user and feedback should be given concerning the decisions made so far and decisions still to be made. Furthermore, the configuration system should displays which activities the user can currently select and which activities cannot be selected yet. When more than one activity can be selected the configuration system provides a default sequence which is reasonable in most cases. Thus, the user can follow this sequence if no other procedure is preferred or necessary at the moment.

4. Mapping to Procedural Knowledge

In this section, we describe how the above defined KBPD-process can be implemented by using mechanisms of structure-based configuration. In the ConIPF project we verify this by enhancing the facilities of the procedural knowledge definable for the tool KONWERK (Günter and Hotz, 1999). Phases are not mapped to procedural knowledge since they are used to describe the relation between the KBPD-process and an organization.

Activities. In the following we show how configuration activities and realization activities can be defined by means of strategies available in KONWERK. All decisions the user can make at a certain time are collected in an agenda. Strategies define which decisions are put on the agenda, the order in which the decisions can be made, and how the value for a decision is determined. Configuration activities group decisions that belong together. This can be achieved by using a strategy with a focus on the corresponding decisions. In Figure 4 (left) an example definition for the activity configure environment is given.

```
Strategy
  Name: Configure Environment
  Strategy Class: Depth-First-Search
  Focus: Environment
  Execution: User Query
  Priority: 95
  Preconditions: Pre-select
  Global Mechanisms:
    Name: Constraint Propagation
      Execution: after each step
```

```
Strategy
  Name: Generate Specification
  Priority: 90
  Preconditions:
   Configure Environment,
   Configure HW/SW-Constraints,
   Configure Features,
   Configure HW/SW-Artifacts
  Global Mechanisms:
    Name: Generate Specification
      Execution: user-initiated
```

Figure 4. Configuration activity (left) and Realization activity (right)

Realization activities do not consist of decisions but produce results based on the decisions made so far. These results are not computed by the configuration system itself but by external global mechanisms initiated to produce e.g. a specification document. In Figure 4 (right) a strategy is shown with the external global mechanism generate specification which can be initiated by the user. Since both, configuration and realization activities, are represented in the procedural knowledge as a strategy, they can be synchronized by means of preconditions. With preconditions for each strategy a set of other strategies can be defined that must have been completed in order to select the strategy (also shown in Figure 4).

Guidance in proceeding. A default sequence for all activities can be defined using priorities assigned to the strategies (see Figure 4). If no specific strategy is selected by the user, strategies are processed according to their priority. While the default sequence can be easily defined, support for displaying all decisions grouped by their activity and giving feedback about which activities are selectable at the moment is limited.

Tailoring. In the ConIPF project the products to be derived are software-intensive systems, i.e. systems that also consist of hardware-artifacts, especially real-time embedded systems like the CPS system.

These systems require a calibration step where the software is iteratively tuned to its run-time environment. However, the KBPD-process is a reference process that can be tailored to the specific needs of an organization, e.g. the calibration phase can be left out for non-embedded systems. Since tailoring is part of the methodology instantiation it is not discussed in detail here.

5. Related Work

Software development in general is a design task where new artifacts are created. Software development methodologies like the RUP (see Rational Software, 1998) or CommonKADS (see Schreiber and Akkermans, 2000) for knowledge-intensive systems comprise the design, modeling and implementation for one system. This is different with the KBPD-process where the design, modeling and development for a whole product family has already been done in domain engineering. The remaining task in application engineering can therefore be described as routine configuration. While configuration of technical systems is one of the most successful application areas of knowledge-based systems (see e.g. Günter and Kühn, 1999) applying configuration methods to software systems is still in an early stage (see e.g. Soininen et al., 1998). New demands like the integration of configuration process and realization process need to be fulfilled. The KBPD-process is related to the application engineering process PuLSE-I (see Bayer et al., 2000). Both model the necessary decisions but in contrast to PuLSE-I in the KBPD-process impacts of decisions are computed by the configuration system.

6. Summary

The advantages from both methodologies (i.e. reuse strategies from the product line approach and consistency checks from structure-based configuration) can be integrated in a process well suited for deriving software-intensive systems. Supporting the product derivation by means of structure-based configuration makes the derivation more efficient since a large number of decisions can be inferred by the system and improves the quality of results since dependencies cannot be overlooked by the user. We showed how the KBPD-process can be defined by means of procedural knowledge for KONWERK. Thus, the requirements mentioned in Section 2 can be met. It is possible to structure the derivation process in subtasks by using strategies, to offer a default sequence for all the activities by means of priorities and to synchronize configuration process and realization process using preconditions. However, the guidance in product derivation could still be improved by offering a better visualization of the whole derivation process.

Acknowledgments

This research has been supported by the EU under the grant IST-2001-34438, ConIPF - Configuration in Industrial Product Families.

References

Bayer, J., Gacek, C., Muthig, D., and 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, pages 237–245, Edinburgh, Scotland.

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

Guindon, R. (1990). Designing the Design Process: Exploiting Opportunistic Thoughts. Human-Computer Interaction, 5:305–344.

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

Günter, A. and Cunis, R. (1992). Flexible Control in Expert Systems for Construction Tasks. Journal Applied Intelligence, 2(4):369–385.

Günter, A. and Hotz, L. (1999). KONWERK - A Domain Independent Configuration Tool. Configuration Papers from the AAAI Workshop, pages 10–19.

Günter, A. and Kühn, C. (1999). Knowledge-based Configuration - Survey and Future Directions. In Puppe, F., editor, XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg.

Hein, A. and MacGregor, J. (2003). Managing Variability with Configuration Techniques. In Proc. of the Workshop on Software Variability Management at the International Conference on Software Engineering, Portland, Oregon, USA.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, S. (1990). Feature-oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021.

Kang, K., Lee, J., and Donhoe, P. (2002). Feature-oriented Product Line Engineering. In: IEEE Software, 7(8):58–65.

Rational Software (1998). Rational Unified Process, Best Practices for Software Development Teams. WWW.

Sabin, D. and Weigel, R. (1998). Product Configuration Frameworks Ũ A Survey. IEEE Intelligent Systems, pages 42–49.

Schreiber, G. T. and Akkermans, H. (2000). Knowledge engineering and management: the CommonKADS methodology. MIT Press.

Soininen, T., Tiihonen, J., Männistö, T., and Sulonen, R. (1998). Towards a General Ontology of Configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), 12, pages 357–372.

Stumptner, M. (1997). An Overview of Knowledge-based Configuration. AI Communications, 10(2):111–126.

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