DOMAIN-CROSSING SOFTWARE PRODUCT LINES IN EMBEDDED, AUTOMOTIVE SYSTEMS

Stefan Kubica
Audi Electronics Venture GmbH
stefan.kubica@audi.de

Wolfgang Friess
AUDI AG
wolfgang.friess@audi.de

Christian Allmann
Audi Electronics Venture GmbH
christian.allmann@audi.de

Thorsten Koelzow
Audi Electronics Venture GmbH
thorsten.koelzow@audi.de

Abstract: The development of software-functions in the automotive domain is subject to multiple conditions. These conditions are for example the rising number of various functions in the car, the simultaneously increasing cost pressure and shortened development cycles. To come up with these conditions, an improvement of reuse is very promising. In this paper the point of view is that a software-function is separable in two domains, the application-domain and the standard-software-domain. Looking at the reuse activities of both domains together provides more generality and synergy-effects. A reuse approach that fits both domains is the approach of domain-crossing software product lines. This paper reports about an ongoing research project about adapting of software product lines in the specific domains and describes a concept for bringing them together. For proving the concept an accompanying tool was implemented and is introduced afterwards.

Keywords: Software Product Lines, Standard-Software, Application-Software, Reuse, Cross-Domain

1. INTRODUCTION

The demand of customers for new innovative functions in cars keeps on rising. Therefore, the car manufacturers equip their new types with a lot of new software functions. Figure 1 points out the raising complexity of functions e.g. in the infotainment sector. The manufacturer has to deal with multiple factors influencing the development of functions. On the one hand there is increasing cost pressure and shortened development cycles and on the other hand there is a demand for more quality and personalisation of the functions by customers.

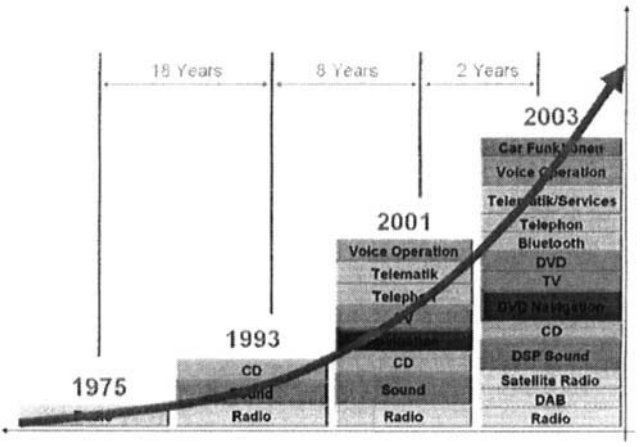

Figure 1. Increasing complexity, e.g. in the infotainment domain [Schleuter, 2002]

As described in [Endres and Rombach, 2003], reuse of software reduces cycle time and increases productivity and quality. Therefore, getting the possibility of a more efficient reuse of software-functions could solve the described problems. In this paper software reuse means, using parts of a function more than once. The challenge is to decide the kind of reuse and what is required to achieve it. A short introduction in the automotive development of functions is given. By examination the whole domain more precisely it occurs that one possible point of view is to share the whole domain of function development into two domains to handle with, the application-domain and the standard-software-domain. Figure 2 shows the two prevalent domains.
The application-domain contains the process of developing the functionality. There are activities in the automotive domain to implement a model-based development process. As described in [Langenwalter and Erkkinen, 2004] the model-based development of functions has several advantages like e.g. iterative development steps, early tests, the possibility of reuse and the generation of production code from models.

The standard-software-domain contains the system-components needed for the ECU (Electronic Control Unit). These components are e.g. the operating system, drivers and the network management. Figure 2 shows an overview of the included components. The configuration of this components depends on the requirements of the functions and on the given hardware-architecture.

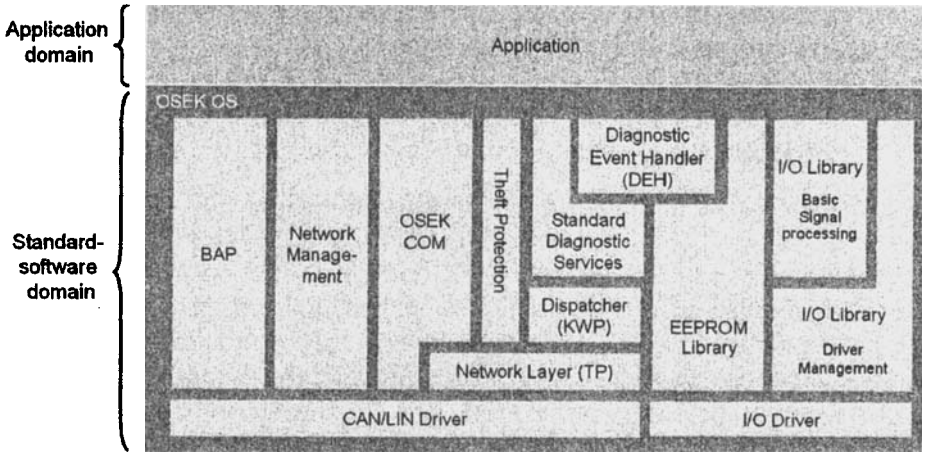

Figure 2. Standard Software Core of the Volkswagen-Group

To come up with the mentioned more effective reuse, it is necessary to support the development-process with additional software development techniques. We propose not only a separate solution for each domain, but a cross-domain solution. By combining reuse-techniques of each domain, it is possible to reach more generality and to benfit from synergy-effects. The method of software product lines offers some answers to the given problems. As described in [Northrop, 1998] software product lines help to reach goals as for example high quality, quick time to market, low cost production and low cost maintenance. To reach these goals would give the development process the improved efficiency and productivity demanded and also an instrument to handle the raising variability. The literature offers several methods for implementing a software product line, like FODA (Feature-Oriented Domain Analyse [Kang et al., 1990]), FeatuRSEB (Feature Reuse-Driven Software Engineering Business [Boellert, 2002]) and FAST (Family-Oriented Abstraction, Specification, and Translation [Weiss and Lai, 1999]). The challenge is to develop an adequate method useable for both described domains. In this paper, a concept of a software product line-framework which combines the software product lines of the application-domain with using model-parts and the standard-software-

domain with the possibility to configure the standard-software components. Furthermore, the first ready steps and the resulting tool-chain are described.

2. PRECONDITIONS FOR DOMAIN-CROSSING PRODUCT LINES

For combining the two domains of application and standard-software, first we consider the specific aspects of both.

2.1 Application Development with Software Product Lines

The model-based development in the application-domain is a first step for improving the possibility of reuse. Reasons for this are for example the independence of the model from the hardware because of using a generator for getting the specific production code. Also the possibility of distributed development by having modules and clear interfaces support reuseability. But to develop the entire application as a model, for example as Matlab/Simulink-model, does not inevitable ensure reuse. The development of various models for various variants of an application contains several disadvantages. These disadvantages are for example the number of possible model-variants of an application is increasing very fast. The problem is, that basic changes of the functionality of the application have to be made in all existing model-variants. The introduction of an additionally functionality in the application can double the number of possible variants and being able to reuse one of the existing model-variants requires expert-knowledge to get the model that fits best.
As described in [Hein et al., 2000] the software product line-approach offers some answers to the given problems. To split the functionality of an application into separate features gives the possibility to generate different variants from one common base. For getting an useable general approach, multiple steps have to be defined. In Figure 3 these steps are shown. In [Böckle et al., 2004] is described, that in general there are two parts inside the process of a software product line development.
Domain engineering includes all activities connected with the development of the software product line. These activities are:

1 Domain scoping: In this step the possible variants of the application were traced.

2 Define features: Extraction of the features and decision if a feature is a common or a variable one.

3 Deposit features with logic: The dependencies between the features were deposit with a mathematical logic. This logic is implemented with an adapted feature-tree notation.

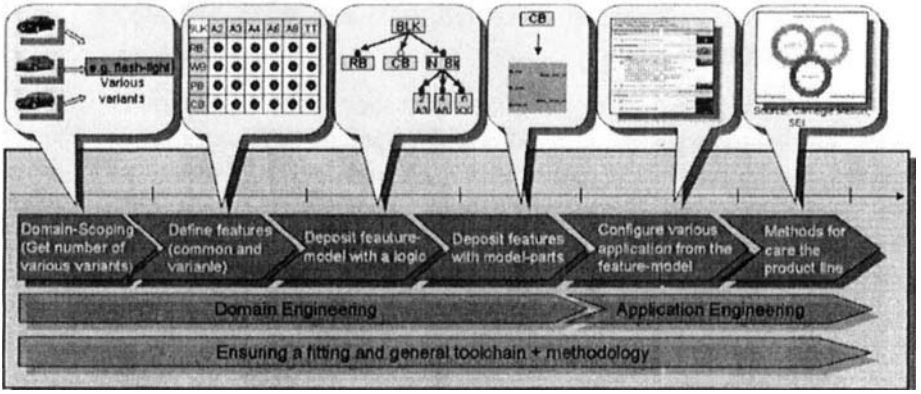

Figure 3. Concept for introducing the software product line for application-development

4 Depositing the features with model-parts. These model-parts have to represent the requirements of the respective feature.

The first two steps are realised with an adapted approach of the Fraunhofer IESE (Institut Experimentelles Software Engineering) called CaVE (Common and Variable Extraction [John and Dörr, 2003]). Within a common project the CaVE-approach was adapted and gives methods for extracting features from function-specifications methodology. The realisation of the third step, the common feature-notation for both domains, is described in a following section. Step 4 is subject to future work.

Application engineering includes all activities connected with the use of the developed software product line. These activities are:

1
- Configurator: Giving the possibility of choose features of the software product line to match the several requirements of a new application demanded.
- Composing-algorithm: Put together the model-parts connected with the selected features to an entire model.

2 Test and Care: Methods for testing the chosen combinations and for carrying the product line (e.g. adding new features to the product line)

The configurator is part of the tool-chain described in one of the following sections. The composing-algorithm is connected to the depositing of the features with model-parts and is also future work.

2.2 Standard-Software Configuration with Software Product Lines

The principle of reuse is a common way to shorten the development time, reduce costs and to increase quality also in the standard-software domain. Therefore, the standard-software, as it is used in electronic control units nowadays, is composed out of several moduls as Figure 2 shows. Only the modules needed for the specific control unit are integrated into the standard-software system. This adaption is necessary to reduce the memory consumption of the software and therefore to reduce costs.
To increase the reusabiltiy of the standard-software modules, it is necessary to set parameters of the moduls to the specific usage of the module. With this mechanism of parametrisation it is possible to reuse standard-software modules in several electronic control units. The challenge in the domain of standard-software is not to enable different variants of the system, but to manage the complexity of the whole standard-software system and the dependencies between the parameters of the modules.
The concept of modelling commonalities and variabilities of many, similar software products with feature models can help to face this challenge. The usability of a feature model-based configuration of an OSEK-conform operating system is already shown in [Czarnecki et al., 2002]. OSEK is the german abbreviation for 'Open Systems and the Corresponding Interfaces for Automotive Electronics' and stand for a joint project in the German automotive industry aiming at an industry standard for an open-ended architecture for distributed control units in vehicles.
One result is the standard for automotive operating systems. Czarnecki and collegues showed that the parameters of an OSEK operating system can represented with feature models and a configuration file can be generated with techniques of template-based code generation. The usage of product line approaches in the embedded domain is also shown by [Beuche, 2003]. He introduced an approach for composing embedded systems with feature models to represent variabilities of similar systems.

To enable a feature model-based configuration for standard-software the given concepts have to be extended to model not only the operating system, but all standard-software modules. Beside the necessity of realizing a generation for different configuration files another challenge has to be solved. By extending the concept to several moduls the importance of considering relations and dependencies between different parameters is increasing. The possibilities of current feature model notations to describe relations are rather limited and have to be extended. An approach to formalize relations in feature models is shown in [Streitferdt et al., 2003]. There, an adaption of OCL (Object Constraint Language [OMG, 2003]) is used to describe relations and dependencies. The

application of such concepts to the standard-software domain is still missing and one aim of our ongoing research.

3. REALISATION

A common feature model notation, fitting the requirements of both domains is the first step to bring them together. A concept for a common notation and a tool are introduced in the following.

3.1 Concept

The previous sections have shown, that the two domains have many commonalities but also many differences. These commonalities and differences must be addressed by a common feature model notation to enable a common modelling of the two domains. For the application domain there are three main specialities. Because of the model-based development in this domain, it must be possible to establish a link between features and the corresponding model fractal which implements the feature. Beside that, it must be possible to define unique interfaces of the model. One way to realize this, is to model the interface signals themselves as features. To integrate a model-based product line for applications in a real world development process, it must furthermore be possible to link process documents like requirements or test cases to the features. The link of requirements is necessary to enable the automatic creation of a product concept catalogue for the generated application.

As shown in section 2.2, the purpose of feature modelling in the standard-software domain is to model the parameters of the different moduls. These parameters often have to be defined as values. An example will illustrate this problem. One parameter of a CAN driver modul could be the size of the communication buffer. So to configure this modul, the parameter *CAN-BUFFERSIZE* has to be defined with a value within a given range. Beside that, there could be a higher level modul, for example a transport protocol, which communicates with the can driver. Then the *TP-BUFFERSIZE* of the second module has to have the same value as the first parameter for compatibility reasons. This short example shows, that in the standard-software domain, there are parameters and relations between parameters. This has to be addressed by a feature model notation.

To enable a common modelling we combined several aspects of existing feature model notations and extended them:

- The basis for our notation is the orignial FODA notation. Feature modeling was proposed as part of the Feature-Oriented Domain Analysis method (FODA). The idea of the FODA notation is to model optional and mandatory features of a system in a hierarchical tree.

- The possibility of modelling feature parameters was added by Czarnecki with the introduction of attributes [Czarnecki et al., 2002]. These attributes are a way to represent a choice of values belonging to a feature. For our notation, it is also possible to add parameters to the features.
- We also added the principle of feature cardinalities from Czarnecki [Czarnecki et al., 2004]. With cardinalities it is possible to model how many instances of a feature must be implemented in the system.
- A possibility, which is not given by existing notations is to add references to all kinds of files to a feature. This is necessary to add documents from the development process to the features, like mentioned before. Also for feature relations it is possible to add references to files in which the relation is described in a formal way.

This proposal of a common notation is integrated in an experimental tool described in the following section. A common editor for feature modelling of both domains and a common output file to store the selected features is also part of a common method for a product-line based development of applications and standard-software.

3.2 Cross-Domain SPL-Tool

An important point to ensure the usability of a product line-based approach for the development of electronic control units is an adequate tool support. On this account we have developed a tool to support the requirements of a common notation for several domains like they are shown in the previous section. Such a cross domain software product line tool (CDS-Tool) is introduced in this section. An overview of the tool structure is shown in Figure 4.

The CPS-Tool has several features differentiating it from other tools, like Consul [Beuche, 2003] or the FeaturePlugin for Eclipse [Antkiewicz and Czarnecki, 2004]. First of all, the tool is integrated in the company software process. This means, the tool is part of the development work flow. To avoid isolated application relevant artefacts like specification documents, model files or code fragments are included in the software product line approach. The tool makes allowance for this requirement by applying such artefacts to each single feature. During feature specification in the domain engineering for each specified relation between single features (e.g require, exclude relation) the basic development artefacts can be deposit. This preparation has the advantage that feature dependencies and constraints can be verified by the deposit specification documents. These documents are stored in the project database. Any change in these documents is directly observable.

The deposit artefacts advise the engineer if his reached decisions are conform

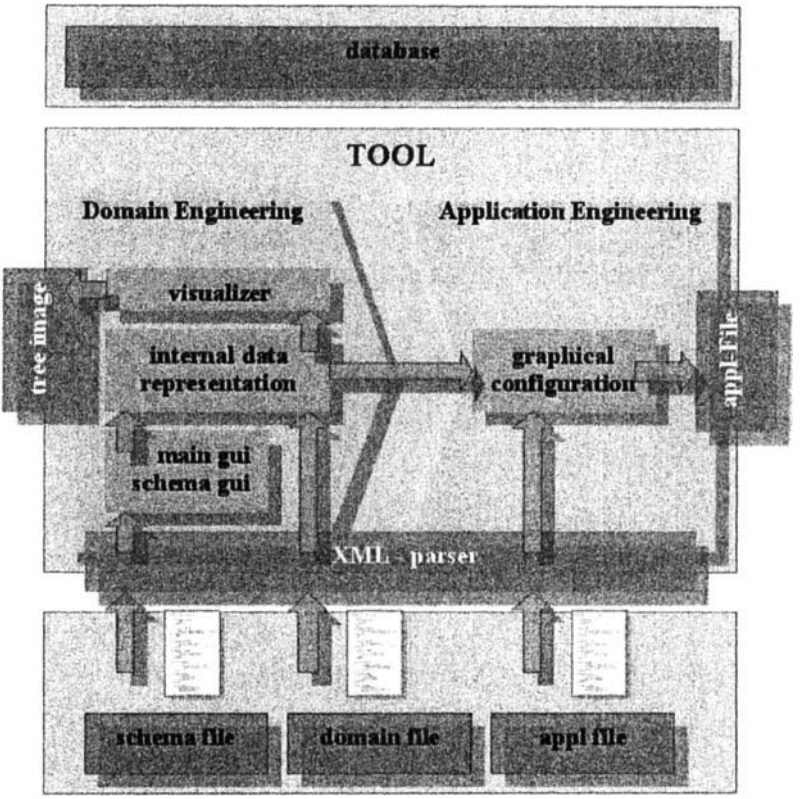

Figure 4. Overview of the structure of the CDS-Tool

to the specification documents. Currently this step must be executed by hand; in the future a constraint checker will automate this step. Beneath the deposit conditions for feature relations it is possible to supplement feature parameters with documents (as file references or simple logical terms). All these additional feature specifications (condition, implementation, description files) help to ensure traceability in the software development process. Figure 5 shows the editor to add references to several files in the tool. They help to facilitate project version management and support product maintainability. To increase the acceptance of the tool from developers, it is moreover possible to view a feature model in a graphical tree-view. With all of this, the requirements from section 3.1 are addressed by the CDS-Tool.

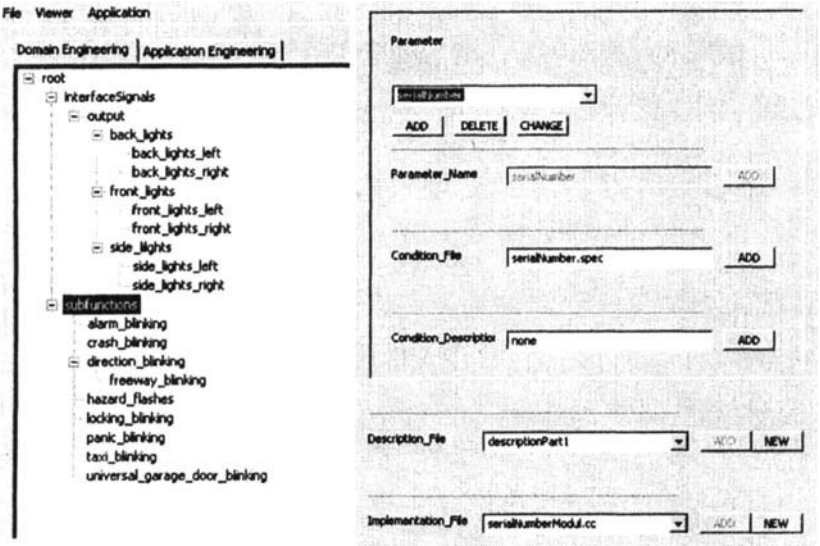

Figure 5. Treeview and Dialog of the Domain-Engineering-Part of the CDS-Tool

4. CONCLUSION AND OUTLOOK

This paper presented an ongoing research out of the field of adapting software product lines to the automotive domain. After an introduction to the different domains in this area, we showed that a common feature model notation is the first step to bring different domains together and gave a proposal for such a notation.

The next step of our work is to complete the tool chain for the whole generation process. For this, we have solve the problem of composing model fractals of a model-based application family to different family-members. In the domain of standard-software, we have to generate configuration files out of the feature model to use existing standard-software.

In parallel we will go on with case studies to gain more experience in combining models of different domains. Especially interactions between the two domains are of special importance to get the biggest benefit of the synergy effects mentioned in the introduction. The standard-software offers its features, modelled in the feature model, to the application. So on the other side, the application features have to express their requirements to the features of the standard-software to enable a semi-automatic preselection of standard-software features. Beside that, there must be a mapping of these requirements to the features of the standard-software. The concept of such a cross-domain middleware will be part of our future work.

REFERENCES

Antkiewicz, Michal and Czarnecki, Krzysztof (2004). FeaturePlugin: Feature Modeling Plug-In for Eclipse. http://www.swen.uwaterloo.ca/ kczarnec/etx04.pdf.

Böckle, G., Knauber, P., Pohl, K., and Schmied, K. (2004). *Software Produktlinien - Methoden, Einführung und Praxis*. dpunkt.verlag GmbH, Heidelberg.

Beuche, Danilo (2003). *Composition and Construction of Embedded Software Families*. PhD thesis, Otto-von-Guericke Universität Magdeburg.

Boellert, K. (2002). *Objektorientierte Entwicklung von Software-Produktlinien zur Serienfertigung von Software-Systemen*. Technical University of Illmenau, Illmenau.

Czarnecki, Krzysztof, Bednasch, Thomas, Unger, Peter, and Eisenecker, Ulrich W. (2002). Generative Programming for Embedded Software: An Industrial Experience Report. In *GPCE*, pages 156–172.

Czarnecki, Krzysztof, Helsen, Simon, and Eisenecker, Ulrich W. (2004). Staged Configuration Using Feature Models. In *SPLC*, pages 266–283.

Endres, A. and Rombach, D. (2003). *A Handbook of Software and System Engineering*. Pearson Addison Wesley, England, Harlow.

Hein, A., Schlick, M., and Vinga-Martins, R. (2000). *Applying Feature Models in Industrial Settings*. P. Donohoe, Software Product Lines - Experience and Research Directions, Kluwer Academic Publishers.

John, I. and Dörr, J. (06/2003). *Elicitation of Requirements from User Documentation*. Proceedings of REFSQ'03, Klagenfurt.

Kang, Kyo C., Cohen, Sholom G., Hess, James A., Novak, William E., and Peterson, A. Spencer (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie Mellon University, Software Engineering Institute.

Langenwalter, J. and Erkkinen, T. (02/2004). *Entwicklung von Embedded Systemen fuer Automobile*. auto & elektronik, Heidelberg.

Northrop, L. (1998). *Essentials of successful product line practise*. Ground System Architecture Workshop, California.

OMG (2003). UML 2.0 OCL Specification. http://www.omg.org/docs/ptc/03-10-14.pdf.

Schleuter, W. (01/2002). *Herausforderungen der Automobil-Elektronik*. Köln: IKB Unternehmerforum, Köln.

Streitferdt, Detlef, Riebisch, Matthias, and Philippow, Ilka (2003). Details of Formalized Relations in Feature Models Using OCL. In *ECBS*, pages 297–304.

Weiss, D. M. and Lai, C. T. R. (12/1999). *Software Product-Line Engineering: A FamilyBased Software Development Process*. Addison-Wesley Pub Co.