Knowledge Engineering in Software Product Lines

Michael Schlick[1] and **Andreas Hein**[2]

Abstract. A software product-line is a collection of products sharing a common set of features that address the specific needs of a given business area [1]. The PRAISE project [2], partly funded by the European Commission under ESPRIT contract 28651 and pursued by Thomson-CSF/LCR (France), Robert Bosch GmbH (Germany), and the European Software Institute (Spain), has investigated product-line realisation and its assessment in industrial settings. A part of the project was dedicated to the validation and consolidation of proposed product-line technologies in real-scale industrial experiments. This paper presents an extract of the experimental results found by Bosch. The Bosch experiment has been located in the Car Periphery Supervision (CPS) domain. The focus during analysis was on feasibility of variability modelling with FODA [3]. The experiment has shown that the FODA model does not provide the necessary expressiveness to represent the different types of crosslinks that are obligatory to describe the domain. Therefore an extension was made to overcome this drawback. Moreover, it became clear that a lot of issues concerning the configuration of FODA models are far from being applicable. Here a solid theoretical foundation is needed first. This paper presents some basic findings.

1 INTRODUCTION

During the last decade, product-line architectures have become increasingly important for companies that want to capitalise on their domain expertise by systematic reuse on a large scale. They divide software development into two distinct life cycles, one for domain and the other for application engineering [4]. The purpose of domain engineering is to model the commonality and variability between its members. Reusable assets are produced by domain engineering and then specialised during application engineering to derive final products.

Requirements traceability has been recognised as being essential to reuse. Application engineering profits from traceability because the engineers understand why a system was built the way it was, and because they can better assess the impact of design modifications. Traceability is even more important for domain engineering where many decisions must be understood to be able to later derive applications from a common architecture and build components for reuse.

The ESPRIT project PRAISE [2] addresses product-line engineering with a special focus on validation of methodological support for domain engineering. Commonality in a domain is represented by requirements, whereas variability must be treated separately. FODA [3] explicitly addresses variability modelling and enables developers to see where variations occur and which decisions have to be made to create a special product. But existing FODA descriptions are not adequate for use in an industrial environment. Within PRAISE, two real-scale experiments have been performed to validate and consolidate product-line methodologies. One focus of the Bosch experiment has been on feasibility and validation of variability modelling as introduced by FODA. It has shown that the existing concepts are insufficient for applying them to real-world engineering tasks. This paper describes the necessary feature model extensions as well as the demands on tools that are to support the feature model configuration process.

The Bosch experiment has been settled in the Car Periphery Supervision (CPS) domain. Actually several business units in the automotive equipment sector produce and develop applications that extensively use information about objects located in the car's periphery. A number of sensors detect obstacles and provide the data needed for further evaluation. Depending on the required measurements, different sensor technologies, namely radar or ultrasonic sensors, can be used. Other types of sensor data are not included in the domain definition at present, although they might be of interest to the domain in the future. Applications that supervise the car periphery offer services such as Backing Aid, Blind Spot Detection, Pre-Crash Sensing and Adaptive Cruise Control.

Today these applications are built as separate systems with the consequence that each one needs its own set of sensors, controller hardware and, last but not least, software architecture and basic sensor-control services.

Sensor applications overlap in terms of their requirements on object location information. Due to the fact that only a restricted number of sensors can be positioned on the vehicle, e. g. in the bumper, shared usage of sensor signals will be inevitable in the near future. Moreover, a reduction of the number of sensors needed does of course reduce the total price of a system. A precondition for sensor sharing is that all participating applications have a common software infrastructure. This infrastructure will be shaped by the architecture of the Car Periphery Supervision domain.

All applications in the CPS domain evaluate sensor data, while simultaneously performing diagnosis and consistency checks. The corresponding services can be characterised by a common architecture with parameterisation or instantiation of its generic parts. The architecture's implementation should provide a *platform electronic control unit* as an expandable basis for the instantiation of multiple, possibly co-operating, applications in the future.

In the following, related work is presented first. Then the software product line approach is introduced, and an overview of the corresponding process and the different tasks is provided. After that we describe the expressiveness needed for a knowledge representation system that is to support product line engineering. The required reasoning is stated explicitly. Next, our actual work is presented. Finally, the major points are summarised in the conclusion chapter.

[1] Robert Bosch GmbH, Corporate Research and Development – FV/SLD, P.O. Box 90 01 69, D-60441 Frankfurt am Main – email: michael.schlick@de.bosch.com

[2] Robert Bosch GmbH, Corporate Research and Development – FV/SLD, P.O. Box 90 01 69, D-60441 Frankfurt am Main – email: andreas.hein1@de.bosch.com

2 RELATED WORK

The feature-oriented concept of the FODA method [3] places special emphasis on the identification and concise representation of commonality and variability in a domain. A feature is understood as “a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems”. Features can be related to another by several types of links. Together the features form the feature tree which is used to parameterise all other models. Use case modelling has been used as a high-level functional description from the viewpoint of application families in the Reuse-Driven Software Engineering Business (RSEB) [5]. The approach introduces the notions of variation points and variants into use cases and analysis types. FeatuRSEB [6] and FODAcom [7] contribute to domain analysis in that they relate variability management in feature and use case modelling. Our work draws on these approaches and extends them with procedures and experiences from our experiment. Requirements templates, as in FODAcom, are one main point within our work. We are using them for requirements derivation starting from feature configurations. An approach of applying feature modelling for controlling code generation is described by [8]. Over and above this, we are trying to apply feature modelling to software design generation.
In [9] an approach is described that goes even beyond focusing on pure product lines or product families towards building product populations. Our application domain can be situated in between these two categories.

3 DOMAIN ANALYSIS ASSETS

Domain analysis assets are a prerequisite for coping with the complexity of software systems in the long run. Especially they are the starting point for the derivation of new products from existing knowledge. Information of the models on the product level is generalised on the domain level in a way that enables developers to see where variations occur and which decisions have to be made to create a special product. As multiple products and the domain level are involved, traceability is of major concern. Traces must be established from the product models to the domain models, and between the different models of each level. Furthermore, relationships between the elements within one model may have special semantics, such as ‘consists of’, ‘alternative’, etc. These model-specific relationships must also be adequately represented.
Domain analysis for CPS primarily consists of completing the requirements, context, and feature models. The requirements model focuses on the commonalities between domain products, the context model defines the interfaces to other domains, while the feature model captures variability within the domain.
One goal of domain analysis is to build an abstract model that can be used as a starting point for the derivation of specific product variants. The feature model is predestined to play this role, as it captures the decisions that have to be made to determine the individual characteristics of a system. Application derivation begins with eliminating the domain variability step by step by instantiating a subset of all potential features. Hence we get a complete specification of one single product variant.
The overall construction process of the requirements and the feature model, depicted in Figure 1, starts with requirements modelling for every single product used to create the domain. Product requirements may also include variability when they specify multiple variants. A requirement addresses variability through parameters. Each requirement that contains parameters can be seen as a requirement template. These templates are further refined by specifying parameter types and values. The combined requirements texts and parameter definitions of the product or the domain form a unit, the requirements model. The parameter definitions section essentially binds the requirements that state commonality and features that state variability.

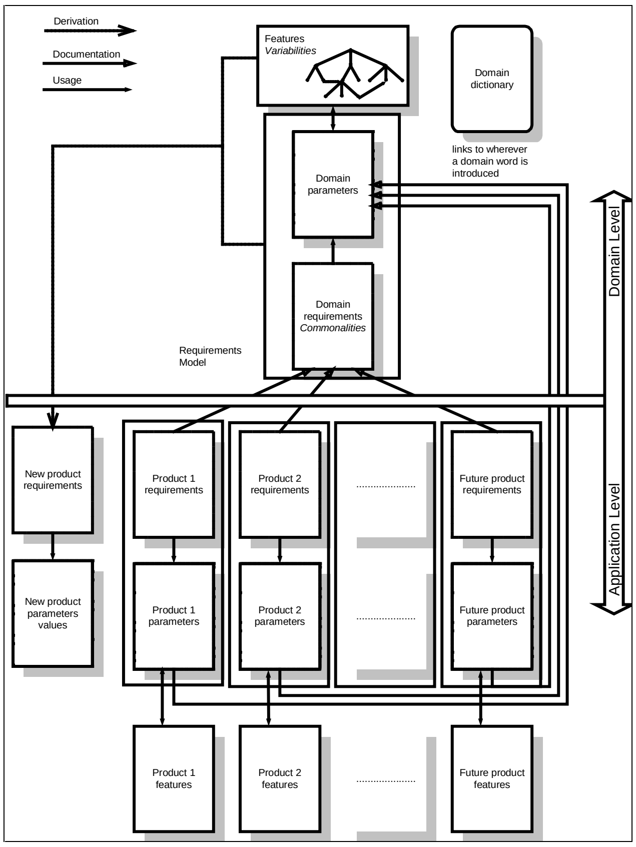

Figure 1. Building reusable domain assets

In the next step, all product requirements are abstracted to build the requirements of the domain. At this point a common domain language emerges. Subsequently, a domain dictionary that defines the vocabulary used by experts and needed to understand the domain and related products and domains is maintained during all phases of the product-line development process. To be able to understand where the abstractions come from, documentation links are established between corresponding elements on the application and domain levels.
Domain feature modelling usually begins after some requirements modelling. The requirements text structure and parameters are used to get an initial feature model that is then refined. The results from a feature modelling phase are carried back to the requirements model. This domain analysis cycle continues until the set of product requirements is sufficiently complete. An open question is where to stop analysis and where to begin architecture modelling.
Features are modelled as nodes of a tree. Basically each node corresponds to a parameter. So the feature tree describes the different possible values that can be assigned to each particular parameter.
Besides the basic tree structure we must introduce additional links between the nodes of the feature model. These links are necessary to express additional constraints between the different features of the tree. The particular constraints must be

documented so that it is possible to comprehend why, e. g., one feature requires or excludes another.

Traceability links are established from each domain requirement to the parameters it contains, and from the parameters to the corresponding nodes of the feature tree. During the derivation process these links in turn are used to generate product specifications by navigating from the nodes of the feature model via the parameters to the requirements.

After domain analysis is complete, the results (i. e. the models) must be validated through application requirements derivation. It should be possible to satisfactorily specify requirements of the products that have been used as the basis for the domain models. It is unlikely that the derivation will produce exact copies of the product requirements models. Depending on the techniques employed, the requirements text of a derived product is more or less similar to the text in the domain. The associated parameters document contains selected values for all parameters that have been covered during derivation. In order to completely derive an application, all variability must be resolved to final values. Nevertheless, variable products can be derived through partial derivation which leaves some selections open.

In summary, feature modelling based on the FODA method is advantageous in several ways:

- *Control over variability*. All variants of applications in the domain and their relationships are represented in a comprehensive and understandable form.
- *Reuse of requirements*. Generation of requirements specifications can be partially automated by the use of templates instantiated with different variants.
- *Configuration support*. As features are linked to modelling of later development phases, application derivation through selection of variants combinations can be used to support configuration right up to product code. [10]
- *Sales support*. Sales representatives have a high-level means to discuss product trade-offs with customers.
- *Support for new development*. When a new product is to be developed, feature modelling simplifies the analysis of how it differs from existing ones.

4 BASIC KNOWLEDGE ENTITIES FOR PRODUCT-LINE ENGINEERING

Three modelling primitives are used to describe the different parameters and their relations within the tree structure of the feature model; aggregation/decomposition, generalisation/ specialisation, and parameterisation. Aggregation is the abstraction of a collection of units, decomposition its refinement into the constituent units. Generalisation is the abstraction of the commonalities among a collection of units into a new conceptual unit, specialisation its refinement by incorporating distinguishing details. Finally, parameterisation is a development technique to adapt generic components in many different ways by replacing the parameters of a component with values.

In addition, two composition links enable specific relationships between features that are not yet covered. Applicability of these links is restricted to optional and alternative features. The first one is the requires-relationship indicating that a certain feature can only be selected if the feature it is connected with is selected, too. This semantics can be further refined by specifying an alternatives-relationship between the required features. The second type of link is the excludes-relationship meaning that not both of the related features can be selected in the same configuration.

Nevertheless, the Bosch experiment has shown that extensions of FODA are needed to represent the complexity of interconnections in the CPS domain. These extensions mainly are concerned with links that enable integrated modelling of several tree structures. However, the extended model has to introduce graph structuring. FODA already broke the single tree structure by introducing undirected constraints between features. But the required graph structures must additionally offer the possibility to distinguish between directed and undirected crosslinks, so that the directed relationships can be used to prevent automatic selection of all features connected to a common feature during application derivation. Of course, directed and undirected links must not connect the same features at the same time.

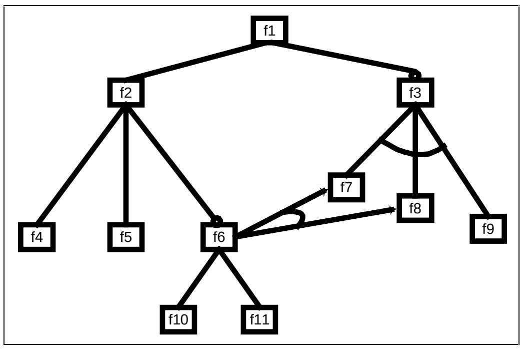

Figure 2. Sample feature tree

Figure 2 shows an example of an extended feature model. The feature tree consists of 11 features. On the top level, f1 is decomposed into a mandatory feature f2 and an optional feature f3. F2 consists of the mandatory leaf features f4 and f5 and the optional compound feature f6. The optional feature f3 is specialised by three exclusive alternatives f7, f8, and f9. In the first structure, f6 acts as an optional compound feature consisting of the two mandatory features f10 and f11. In the second structure, f6 is a root node (like f1 in the first structure). It plays the role of a mandatory alternative with the two exclusive specialisations f7 and f8. Application derivation is supposed to be mainly performed along the primary structure. For this reason, only the secondary structure employs directed links for alternative representation. In this way the selection of f7 or f8 does not automatically cause the selection of f6.

To represent product line requirements and feature models, we therefore need the following expressiveness:

- *Generic features*. It should be possible to define a generic feature in the domain model that can be instantiated several times if more than one instance of that feature is needed in the derived application model.
- *Feature attributes*. For defining a feature in more detail it must be possible to describe its properties by attributes.
- *Specialisation / Is-A links*. Alternatives can be represented by specialisation of a feature, assuming that the different sub-features are disjunctive.
- *Has Parts /Is-Part-Of links*. To represent mandatory and optional sub-features we must represent part-whole relations. Moreover, it must be possible to define a cardinality for these links. The cardinality is restricted to the interval [0 1].

- *Documentation links*. To be able to explain the choices it must be possible to link the feature descriptions to a documentation text.

Hence, to represent features we basically need two hierarchies; taxonomy and partonomy. If we want to provide multiple tree structures, we have to support multiple inheritance. It must be pointed out that multiple inheritance is not indispensable. It can be replaced via expressive constraints. Nevertheless, there is a high risk for combinatorial explosion.

The different tools for software requirements engineering and management that are available on the market do not support the product line idea. Existing tools do not provide a knowledge representation that supports the reuse of requirements nor do they support the reasoning that is necessary for application derivation.

Our experiment has shown that applying the software product line approach to a real-world problem is not possible if appropriate tools are missing. This is especially due to the following characteristics of our problem:

- A high degree of variability with respect to lines of code.
- A high amount of secondary links within the feature model.

Within the PRAISE project we spent a lot of time on modelling the domain requirements and features. Especially we tried to encapsulate the variability and to reduce the number of secondary links by adequate representation, but we could not achieve a significant improvement. Consequently, a high degree of variability and a substantial amount of constraints between the single variants seem to be typical for embedded real time software products and especially for sensor fusion projects. Nevertheless, we think that the product line approach is the appropriate technology to build the software for those sensor fusion products. So we have to find a better way to represent and reason about our domain models.

Knowledge based systems and especially configuration systems may be a solution to overcome the shortcomings of the requirements management tools.

5 REASONING FOR APPLICATION DERIVATION

One possibility to handle the application derivation problem would be to use a configuration system for the knowledge representation and the needed reasoning. Due to the hierarchical structure of our problem, the hierarchical structure of our model, hierarchy oriented configuration techniques can be applied.

One shell that does provide the necessary modelling means is Konwerk [11]. There the different configuration objects can be modelled within a taxonomy and a partonomy. Moreover, Konwerk offers constraints to model the parts of the secondary structure that cannot be represented in hierarchies due to their restriction to simple inheritance. Furthermore, Konwerk provides the necessary reasoning services for deriving the product requirements from the domain model. These reasoning services can be described as a navigation in the different hierarchies, called structure-oriented configuration, combined with additional constraint propagation. An important issue is that the system can compute whether the configuration is finished, hence whether a complete description of a product has been made. This capability is based on the fact that in Konwerk it is assumed that an instance of a configuration object has finally to be an instance of a leaf or concrete concept of the taxonomy. Based on the elementary reasoning services a configuration strategy can be defined that steers the configuration process.

Konwerk like some other configuration systems uses a frame language as underlying knowledge representation formalism [12]. The problem about those configuration systems is that they are based on the closed world assumption; they assume that the knowledge they reason about is complete with respect to the problem. This assumption is not suitable for our problem. Within software product line domain engineering we cannot build models that are complete for all parts of our problem. E. g., in the CPS domain we have complete knowledge about the requirements and features of the sensor technology, but we cannot model complete knowledge about the human machine interface of CPS products, because this knowledge is very customer-specific. It must be possible to represent and to reason about incompleteness, and it must be possible to provide a mixed reasoning.

During requirements derivation it must be possible to configure a requirements text and check its completeness with respect to the given knowledge base. Furthermore, it must be possible to compute the parts that have to be further specialised by additional requirements. In the next enhancement of the domain knowledge base, the relevance of these additional requirements for further projects can then be checked, and they can be integrated into the domain model.

One possibility to overcome the shortcoming of the closed-world assumption is to introduce the explicit notation of concrete and abstract features. If during requirements derivation we have created an instance of an abstract feature that could not be further specialised by an instance of a concrete feature, we know that this feature is still incomplete and has to be further specialised independently of the domain model. In this case, variability is represented by under-specification.

Another possibility would be to employ a knowledge representation formalism that is able to reason under the open world assumption, e. g. description logics. Configuration systems can be build in description logics [13]. Furthermore, it is possible to provide reasoning under different assumptions about the world in description logics. Nevertheless, it must be proven that a description logics approach is not oversized.

6 FUTURE WORK

Based on the ‘4+1 View Model of Architecture’ [14] we are currently modelling the CPS domain architecture using UML [15]. Like in requirements modelling we have to represent variability. Furthermore we must guarantee the traceability from the feature model to the different parts of the architecture. Again we lack sufficient tool support as even the underlying knowledge representation and reasoning have not been described yet. Nevertheless, the problems seem to be similar. Based on our experience with OMOS [16] we think that it is appropriate to model variability in logical architectures using inheritance, and to apply structure-oriented configuration methods for application derivation.

In the next step we are going to describe the overall expressiveness and reasoning services that are needed for the software product line approach, so that we will finally be able to identify a corresponding knowledge representation formalism. Consequently we must find an adequate tool that we can integrate into our tool chain as basic knowledge representation and reasoning mechanism.

Our goal is to compute the effort needed to build a product already while defining its requirements. One possibility to achieve this is to use the type of configuration problem (routine, innovative, or creative configuration) [17] as an indicator for the complexity of the software engineering task.

7 CONCLUSION

We have shown that for applying the software product line approach, and for domain analysis in particular, strong tool support is needed. Current tools are not sufficient as they lack basic knowledge representation and reasoning capacities. We have proposed appropriate knowledge representation and configuration systems to fill this gap. Finally, systems that can deal with incomplete knowledge have been considered especially useful.

ACKNOWLEDGEMENTS

We would like to thank our colleagues from the Robert Bosch GmbH business units for their contribution to models realisation, and our PRAISE project partners Thomson-CSF/LCR, and ESI for the technical discussions.

REFERENCES

[1] European Software Institute: Product-line architectures and technologies to manage them. ESI-1998-RECORD-ED1, 1998.

[2] ESPRIT Project 28651: PRAISE – Product-line Realisation and Assessment in Industrial Settings. IT RTD Project Programme, 1998, http://www.esi.es/Projects/ Reuse/Praise/.

[3] Kang, Kyo C., Cohen, Sholom G., Hess, James A., Novak, William E., and Peterson, A. Spencer: Feature-Oriented Domain Analysis (FODA). Feasibility Study. Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute, 1990.

[4] Lalanda, P.: Domain Specific Software Architecture. CEC deliverable P28651-D2.1, Esprit project PRAISE, http://www.esi.es/ Projects/Reuse/Praise/. December 1998.

[5] Jacobson, I., Griss, M., and Jonsson, P.: Software reuse – architecture, process and organization for business success. Addison Wesley Longman, 1997, ISBN 0-201-92476-5.

[6] Griss, M. L., Favaro, J., and d'Alessandro, M.: Integrating feature modeling with the RSEB. Proceedings of the 5th International Conference on Software Reuse (ICSR'98), IEEE Computer Society Press, Victoria BC, Canada, June 2-5, 1998, ISBN 0-8186-8377-5.

[7] Vici, Alessandro Dionisi, Argentieri, Nicola, Mansour, Azza, d'Alessandro, Massimo, and Favaro, John: FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry. Proceedings of the 5th International Conference on Requirements Engineering (ICRE'98), IEEE Computer Society Press, Victoria BC, Canada, June 2-5, 1998, ISBN 0-8186-8377-5.

[8] Czarnecki, K., and Eisenecker, Ulrich W.: Synthesizing objects. In: Proceedings of the 13th European Conference on Object-Oriented Programming (ECOOP'99), Springer, 1999.

[9] Van Ommering, R.: Beyond Product Families: Building a Product Population. In Proceedings of the Third International Workshop on Software Architectures for Product Families (IWSAPF-3), Universidad de Las Palmas de Gran Canaria, Spain, March 2000.

[10] Weiss, David M., and Lai, Chi Tau Robert: Software Product-Line Engineering. A Family-Based Software Development Process. Addison-Wesley, 1999.

[11] Günter, A.: Wissensbasiertes Konfigurieren, infix, Sankt Augustin, Germany 1995

[12] Cunis, R.: Das 3-stufige Frame-Repräsentationsschema - eine mehrdimensional modulare Basis für die Entwicklung von Expertensystemkernen. Infix, St. Augustin, Germany 1992.

[13] McGuinness, D., and Wright, J.: An Industrial-Strength Description Logic-Based Configuration Platform. In IEEE Intelligent Systems, July/August 1998, pp. 69-77.

[14] Kruchten, P.: The 4+1 view model of architecture. In: IEEE Software 12 (6), November 1995, pp. 42-50.

[15] Egyed, A.: Integrating architectural views in UML. Technical Report USC/CSE-99-TR-514, University of Southern California, Center for Software Engineering, 1999.

[16] Hermsen, W., and Neumann, K. J. : Application of the Object-Oriented Modeling Concept OMOS for Signal Conditioning of Vehicle Control Units. 2000 SAE International Congress and Exposition. Detroit/U.S.A., 6.-9.3.2000.

[17] Gero, J.: Design Prototyps: A Knowledge Representation Scheme for Design. AI Magazine, 11(4), 1990, pp. 26-36.

[18] Lalanda, P.: Product-line Software Architecture. CEC deliverable P28651-D2.2, Esprit project PRAISE, http://www.esi.es/Projects/ Reuse/Praise/. March 1999.

[19] Vinga-Martins, R., and Süßlin, S.: Requirements traceability. CEC deliverable P28651-D2.3, Esprit project PRAISE, http://www.esi.es/Projects/Reuse/Praise/, March 1999.

[20] Hein, A., MacGregor, J., Schlick, M., and Vinga-Martins, R.: Lessons Learned. CEC Deliverable P28651-D3.4, Esprit Project PRAISE, http://www.esi.es/Projects/ Reuse/Praise/. March 2000.

[21] Vinga-Martins, R.: Requirements traceability for product-lines. Workshop on Object Technology for Product-line Architectures, ECOOP'99, 15.6.1999.

[22] Hein, A., Schlick, M., and Vinga-Martins, R.: Applying feature models in industrial settings. To Appear In: Proceedings of the First Software Product Line Conference, August 28.-31., 2000, Denver, Colorado.