Needed Expressiveness for Representing Features and Customer Requirements

(position paper)

Thorsten Krebs[1] and Lothar Hotz[2]

[1] LKI, Fachbereich Informatik, Universität Hamburg
Hamburg, Germany, 22527
`krebs@informatik.uni-hamburg.de`
[2] HITeC c/o Fachbereich Informatik, Universität Hamburg
Hamburg, Germany, 22527
`hotz@informatik.uni-hamburg.de`

Abstract. During the development of a software system, different representation mechanisms are used. From the initial high-level representation, subsequently development goes to more concrete detailed representations. In this paper we discuss the expressiveness of high-level representations like customer requirements and features as well as relations and restrictions between those knowledge entities. Representation facilities defined for structure-oriented configuration are addressed and their suitability for representing customer requirements and product features is discussed.

1 Introduction

During the development, a (software) system goes through multiple development phases like requirements engineering, system design, implementation, etc. Each development phase has its own representations. The initial high-level models describe requirements and features of the software system. Based on these high-level representations, first design decisions are taken and subsequently development goes to more concrete detailed representations [10].

In this paper we focus on the first high-level representations, i.e. customer requirements and features. While there has been done quite some work addressing representation of and reasoning with feature trees [14, 11, 21, 6], the mapping between the diverse representations of the distinct development phases is rather unspecified. The mapping describes both, hints to gain a pre-selection of representation entities within the next development phase and restrictions between representation entities within the same development phase. For instance the selection of customer requirements has an explicit impact on the selection of features. Features themselves on the other hand can mutually exclude or include other features.

Research communities participating in requirements engineering and feature analysis use different representation facilities. We do not want to question this given knowledge about requirements and features but to define a mapping for both of these representation entities to a common structure. For formalizing features and customer requirements we propose the modeling facilities developed in the field of *structure-oriented configuration* [15, 8, 13]. Besides giving a language for describing distinct kinds of variability, the structure-oriented configuration approach provides algorithms for reasoning about representation entities of diverse kinds. Thus, representing customer requirements and features with the same modeling facilities is expected to enhance creating a transparent and consistent product derivation process within the different development phases - i.e. the different levels of abstraction.

In the following sections we present our position on features (Section 2) and of customer requirements (Section 3) and their representation. In Section 4 we describe how to represent requirements and features with modeling facilities defined for structure-oriented configuration. After that we give a conclusion (Section 7) and present some related work (Section 6). Points for discussion are listed in Section 5.

2 Features

We follow the definition in [14] where, a *feature* is "a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems" and in [6] "a feature model is a specific domain model covering the features and their relationships within a domain". In the following, needed facilities for representing features are discussed.

Specializations allow defining more specific system properties and therefore a distinction between different groups of features. *Decompositions* provide the means for grouping related properties by placing them next to each other. Feature hierarchies typically contain facilities to express diverse variability types:

Mandatory features are present in all products that belong to the current product line.

Optional features may or may not be included in the product. If an optional feature is not part of the product, all subfeatures are also excluded.

Alternative features represent a choice between a set of features from which one and only one has to be chosen.

Multiple features capture the possibility to choose many features from a set of features, but at least one has to be chosen.

Because features can be interconnected not only by hierarchical relations (i.e. taxonomies and decompositions) but also by restrictions concerning arbitrary features, we see the need for more variability mechanisms:

Features should be uniquely identifiable. Thus, each feature should contain a non-recurring *ID* or type descriptor. A textual *description* should be available for each feature in order to build a well documented model and to be able to automatically generate documentation. The following is a list of further needed restrictions (i.e. relations concerning multiple objects at arbitrary positions in the hierarchy) between features:

Requires Features can be *required by* other features - i.e. their existence is needed for the first one. This relation can be seen as a mutual inclusion.

Excludes Features may *exclude* each other. This happens when two features can not be selected together, e.g. when system components are incompatible. This is a mutual exclusion.

Recommends A weak form of the mutual inclusion is a recommendation. The existence of features can be *recommended for* other features. This can also be seen as the semantics of a default value.

Discourages Contrary to the latter, features may be *discouraged for* other features in the system. This is a weak form of the mutual exclusion. Hence, it describes that a feature is not chosen per default.

Due to views of various granularity, features are utilized as requirements (functional and non-functional) and properties of product lines and products [17]. The different views emerge because the different roles, people dealing with the feature trees have. These roles can comprise customers, sales persons, system designers and software developers – making different views of various granularity quite natural. Properties of product lines are common to all product line members - i.e. mandatory features, which parent-features are neither optional nor connected to alternatives. Optional features are variant features which represent the permissible differences between product line members.

3 Customer Requirements

Customers requirements are used to select properties of individual products out of a product line. Therefore, customer requirements have the same purpose as features have, i.e. specifying the properties of an individual product – but on a different level of abstraction. Customer requirements are expressed in a language the customer can understand whereas features usually describe system properties on a level of technical detail the product developers can derive the desired product from. Moreover, customer requirements often over- or under-specify the desired system properties. Because of that, customer requirements are a way to abstract from features in order to achieve a description of the product properties as accurate as possible.

Customer requirements abstract from more specific representation entities like features and artifacts (i.e. hardware and software components). Since requirements can be very specific, they are not always more general than features. But even if they refer to artifacts, they are only an abstraction of those. An example for this is given when a customer would like to have a component from a specific vendor and does not accept other products. Features represent characteristics and functionality and are therefore more general than component-specific properties like the vendor. One requirement can imply the existence of multiple features and / or artifacts, and the other way round multiple requirements can map to one feature or artifact. Thus, there is a *n:m-mapping* between customer requirements and features or artifacts.

The modeling of topic-related groups of customer requirements allows the representation of contradicting system properties. A car vendor for instance has to fulfill legal requirements (e.g. emission laws) to be able to sell his products. When a customer demands a very powerful motor and there is no such motor that is powerful and can fulfill the legal requirements, there is a conflict in the requirements specification. The customer might be satisfied with a motor slightly less powerful but conforming to the emission laws. But the car vendor is not allowed to sell his car when he creates the motor specified by the customer requirements, that does not conform to the emission laws.

Customer requirements should not be treated equally significant. Optimization in requirements engineering is gained when customer requirements are definable in various degrees of importance. As already stated in [20], the definition of *definitive* and *optional* requirements can yield to a more adequate requirements specification. Also a distinction between requirements concerning one system component (e.g. the powerful motor) and requirements that address multiple components, i.e. *complex* requirements (e.g. the price of the product that depends on all the components) can improve the quality of requirements engineering.

An approach for getting a model describing customer requirements is to start from the feature model. By enhancing the feature model with new types of features and extending property descriptions which are not yet covered by the product line the customer requirements model is a basis for future products. This is based on the fact, that often the development of new products is motivated by customer requirements. Thus, a model for customer requirements is achieved that:

- is strongly related to the features of the system and already developed products,
- is not far away from the product line, thus realistic,
- can be extended by modeling development costs for anticipated changes of artifacts, and thus, using such cost estimations for configuration, and
- can be individualized for distinct sales persons with separate focuses.

4 Representation of Features and Customer Requirements

In *structure-oriented configuration* representation languages are used for configuration of technical domains [8, 18]. These configuration approaches are currently being applied to the field of software configuration [15, 2]. In the following we give a short glance on how the needed expressiveness of features and customer requirements can be realized with these modeling facilities. The main benefit is the possibility of using this model for automated selection of appropriate features and artifacts for given customer requirements.

The knowledge needed for structure-oriented configuration can be grouped into three categories:

Domain Model A domain model describes objects by their types and properties (i.e. parameters and relations between parameters and objects). Main relation types are decompositional relations (*has-parts*), taxonomic relations (*is-a*) and restrictions between parameters of arbitrary objects – expressed by constraints.

Procedural Knowledge Procedural knowledge describes the configuration process, i.e. the order of configuration decisions to be made. It mainly consists of focusing on specific parts of the domain model and criteria for the evaluation order. Furthermore, conflict resolution knowledge can be defined (e.g. by introducing explicit backtracking points).

Goal specification A goal specification describes a priori known facts about the product to be configured.

The domain model describes knowledge about configurable products while procedural knowledge is concerned with the process of product configuration. In the following we show how these modeling facilities can be utilized for representing features and customer requirements and the relations between those as described in Section 2:

- Types are used for representing distinct kinds of software assets. For identifying a non-recurring *ID* is modeled. Also a *textual description* can be included.
- The means of mandatory, optional, alternative and multiple variability choices are already given by the modeling facilities of structure-oriented configuration. The parts of an aggregate in a decompositional relation (*has-parts*) are assigned with a numerical expression that represents their cardinality. Therefore, a mandatory feature or customer requirement is assigned with the interval [1 .. 1], an optional feature or customer requirement with [0 .. 1], an alternative is represented as multiple optional features from which exactly one has to be chosen (i.e. set to [1 .. 1] and the others set to [0 .. 0]) and multiple features and customer requirements are represented with [1 .. n].
- The requires relation can be interpreted as a form of decompositional relation so that the required features and artifacts are mandatory parts and the customer requirement or feature requiring those is the aggregate. Thus, we speak of a *generalized decompositional relation.*
- Incompatible features and customer requirements that exclude each other can be connected via a constraint. An excluding constraint therefore would narrow down the cardinality interval of the corresponding object to [0 .. 0].
- Default values exist for object parameters and cardinalities in relations between objects. These can be utilized to model recommended features and customer requirements (by using [1 .. 1] as cardinality). Discouraged features and requirements can be represented the same way – only here the default cardinality has to be [0 .. 0].

Note, that default values are only evaluated when this is defined in the procedural knowledge. Other calculation methods are querying the user or computation of functions. Default values can be modeled to be automatically chosen or they can be presented as a "recommended choice" to the user before taking over the value.

We expect to model features and customer requirements with these representation facilities. But not only the representation is supported by the structure-oriented approach but also appropriate reasoning algorithms which realize the underlying logical semantics of such a model [16]. By using these algorithms tools are developed which process a partial automatic configuration [1, 9]. Because this methodology is general, i.e. domain independent and abstract, it could be used for feature and customer requirement modeling and reasoning.

5 Outlook

There are quite some open questions in the field of representing features and customer requirements. In the following we give a short survey on topics of interest for the upcoming research tasks in this area.

It has to be examined whether the same modeling facilities can be utilized for representing features as well as customer requirements. On the one hand, both entities have the same relation types – e.g. *requires*, *excludes* etc. On the other hand, customer requirements are not necessarily structured in hierarchies using means of specializations and decompositions. They are rather unstructured, maybe grouped for representing semantic coherence.

Another question is how customer requirements should be modeled. Because requirements are not structured into deep taxonomic hierarchies, they would be modeled next to each other on the same level. This can be a more or less unclear structure for large amounts of requirements. But existence of customer requirements is needed for automating the mapping to features and hard- and software components.

The next question is how requirements of diverse significants (e.g. *definitive* and *optional*) can be modeled. A first idea is to keep the definition of customer requirements the same for all levels of importance and to have a different kind of mapping to the more concrete knowledge entities. This means that for instance a definitive requirement would entail a *requires* relation to the concerned features and artifacts while an optional requirement would entail a *recommends* relation for the mapping.

The last topic is followed by the question if and how a mapping from customer requirements of diverse importance to features and / or artifacts can be automated. The major difficulty here is the prioritization of requirements – i.e. the selection of the most important mappings. Furthermore, it is not yet clear how customer requirements, that cannot be fulfilled, are treated. Maybe some kind of case-based reasoning could help in enhancing feature and artifact selections in this situation.

6 Related Work

A lot of research has been done in the field of feature analysis starting with the FODA approach [14]. Similar representations have been worked out in different research areas such as using UML [4] or methods of aspect-oriented [19] and generative programming [5]. Also using features in combination with product lines – especially software product lines has already been addressed in [3, 12, 10, 11, 7]. Our approach uses the demands of expressiveness for features given in FODA and maps those to the modeling facilities of structure-oriented configuration in order to gain automated reasoning and consistency checks of the configuration model.

Customer requirements are grouped to gain semantic coherence but they are not modeled in trees like features are. Therefore, having a syntactical representation, mapping both customer requirements and features is a promising approach. Furthermore, using customer requirements as a starting point for structure-oriented product derivation in the context of product lines – as done in the ConIPF project – gives a new research area.

7 Conclusion

Customer requirements may demand the existence of other requirements. This can be formalized using the generalized decompositional relation giving the mapping a meaning of *has sub-requirements*. The same goes for features: the generalized decompositional relation can be used to model *subfeatures*. The selection of customer requirements is logically linked with the selection of product features. Since customer requirements and features both are formalized using the same modeling facilities, the generalized decompositional relation can also be used to express that requirements *map to* features.

Thus, using the same modeling facilities for customer requirements, features and the mapping between those has two major benefits:

1. The automatic reasoning methods provided by structure-oriented configuration can be used to configure products based on customer requirements. The inference machine can process both the requirements model and the feature model and can take dependencies between these two representation entities into account. The configuration task can be divided into *phases* dealing with an arbitrary number of abstraction levels (e.g. customer requirements and features or hardware components and software modules).
2. Consistency checking methods that already exist in the structure-oriented configuration approach can be used to check both the modeled customer requirements and the feature model. Furthermore, the mapping between these two representation entities can be included in consistency checks after modifying or when loading the configuration model.

The benefits presented above become more significant when thinking about a configuration process which uses further representation entities like high level product features and low level implementation features (e.g. performance of algorithms) and also using models of software modules and hardware components. Using the structure-oriented configuration approach, all these representation entities can be modeled, processed and kept consistent with the same modeling and reasoning facilities. Implementing and testing this will be one task within the European project ConIPF (Configuration of Industrial Product Families).

Acknowledgments

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

References

1. V. Arlt, A. Günter, O. Hollmann, T. Wagner, and L. Hotz, 'EngCon - Engineering & Configuration', in *Proc. of AAAI-99 Workshop on Configuration*, Orlando, Florida, (July 19 1999).
2. T. Asikainen, T. Soininen, and T. Männistö, 'Towards Managing Variability using Software Product Family Architecture Models and Product Configurators', in *Proc. of Software Variability Management Workshop*, pp. 84–93, Groningen, The Netherlands, (February 13-14 2003).
3. J. Bosch, *Design & Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Addison-Wesley, May 2000.
4. M. Clauss, 'Generic Modeling using UML Extensions for Variability', in *DSVL 2001.* Jyvaskylae University Printing House, Jyvaskylae, Finland, (2001).
5. K. Czarnecki and U.W. Eisenecker, *Generative Programming Methods, Tools, and Applications*, Addison-Wesley, 2000.
6. A. Ferber, J. Haag, and J. Savolainen, 'Feature Interaction and Dependencies: Modeling Features for Re-engineering a Legascy Product Line', in *Proc. of 2nd Software Product Line Conference (SPLC-2)*, Lecture Notes in Computer Science, pp. 235–256, San Diego, CA, USA, (August 19-23 2002). Springer Verlag.
7. L. Geyer and B. Becker, 'On the Influence of Variabilities and the Application Engineering Process of a Product Family', *SPLC2*, (August 2002).
8. A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.
9. A. Günter and L. Hotz, 'KONWERK - A Domain Independent Configuration Tool', *Configuration Papers from the AAAI Workshop*, 10–19, (July 19 1999).
10. J. van Gurp, J. Bosch, and M. Svahnberg, 'On the Notion of Variability in Software Product Lines', in *Proc. of Working IFIP/IEEE Conference on Software Architecture (WICSA)*, (2001).
11. A. Hein, J. MacGregor, and S. Thiel, 'Configuring Software Product Line Features', in *Proc. of ECOOP 2001 - Workshop on Feature Interaction in Composed systems*, Budapest, Hungary, (June, 18 2001).
12. A. Hein, M. Schlick, and R. Vinga-Martins, 'Applying Feature Models in Industrial Settings', in *Software product lines - Experience and research directions*, ed., Donohoe P., pp. 47–70. Kluwer Academic Publishers, (2000).
13. L. Hotz and A. Günter, 'Using Knowledge-based Configuration for Configuring Software?', in *Proc. of the Configuration Workshop on 15th European Conference on Artificial Intelligence (ECAI-2002)*, pp. 63–65, Lyon, France, (July 21-26 2002).

14. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, ‘Feature-oriented Domain Analysis (FODA) Feasibility Study’, *Technical Report CMU/SEI-90-TR-021*, (1990).
15. T. Krebs, L. Hotz, and A. Günter, ‘Knowledge-based Configuration for Configuring Combined Hardware/Software Systems’, in *Proc. of 16. Workshop, Planen, Scheduling und Konfigurieren, Entwerfen (PuK2002)*, ed., J. Sauer, Freiburg, Germany, (October, 10-11 2002).
16. R. Möller, C. Schröder, and C. Lutz, ‘Analyzing Configuration Systems with Description Logics: a Case Study’, in *http://kogs-www.informatik.uni-hamburg.de/~moeller/publications.html*, University of Hamburg, (1997).
17. S. Robak and B. Franczyk, ‘Feature Interaction and Composition Problems in Software Product Lines’, in *Proc. of Feature Interaction in Composed Systems – ECOOP 2001 Workshop*, Budapest, Hungary, (June 18-22 2001).
18. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, ‘Towards a General Ontology of Configuration’, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), 12*, 357–372, (1998).
19. A Speck, M. Clauß, and B. Franczyk, ‘Concerns of variability in ”bottom-up” pruduct-lines’, in *Proc. of Second Workshop on Aspect-Oriented Software Development*, Bonn, Germany, (February 2002).
20. M. Thäringen, ‘Wissensbasierte Erfassung von Anforderungen’, in *Wissensbasiertes Konfigurieren*, ed., A. Günter, Infix, (1995).
21. S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick, ‘A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems’, in *Proceedings of In-Vehicle Software 2001 (SP-1587)*, pp. 43–55, Detroit, Michigan, USA, (March, 5-8 2001).