A Component Design Process based on Feature Model Transformations

Matthias Riebisch, Periklis Sochos, Technical University Ilmenau

Abstract— **For the architectural design of component-based systems, reusability, flexibility and several other non-functional properties constitute important goals. The component design determines if the requirements to a system can be fulfilled, if the components can be composed easily and if they can be used in a flexible way as it is crucial especially for mobile and distributed applications. Due to the complexity of the design task, architectural design methods, which solve the following four steps, are necessary. First, a transition between problem specification and solution has to be performed. Second, the non-functional requirements have to be implemented by functional solutions. Third, the components have to be defined in such a way that they enable a high flexibility, evolvability and reusability. Fourth, the design has to be developed in conformance to the established component technologies. Available architectural design methods solve some or parts of these tasks, but methods and a process enabling an all-embracing design methodology are missing. This paper presents the Feature-Architecture Mapping Method (FArM) as a means for the architectural design of well-separated components conforming to plug-in frameworks; and it places FArM in a process of component design especially for the needs of mobile and distributed applications. By performing the architectural design activities by operations on features instead on components, the principle of separations of concerns can be applied as well as the goals of variability and flexibility can be achieved. The method has been evaluated through its application in a number of domains including the mobile domain, the Integrated Development Environment domain and in a neural network trainer product line.**

Index Terms—**Architecture, Design methodology, Modeling, Software maintenance, Software reusability,**

I. INTRODUCTION

In many domains, the software systems have to fulfill frequently changing requirements, combined with a demand for customization and for a high efficiency of the development. Component-based software systems are able to satisfy this demand. However, because the systems are frequently of a high complexity, the design of the components and of the software architecture is a challenging task. Component-based techniques have been on the focus of the software engineering research during the last three decades, because of the advantages of components for mastering the complexity and as elements of reusability [13]. By hiding the implementation details of the component behind their interfaces, the dependencies between components can be minimized; and there is a higher likelihood that a changed requirement leads to only a local change and does not affect wide parts of a software system. However, the latter will only occur, if the component interfaces are not changed. Therefore, software architectural methods and principles have been developed to design robust structures of components with stable interfaces.

The software architecture plays a very critical role for different aspects that are important for a successful development of software systems. As a very basic aspect, the architecture assigns the responsibilities to the parts of the realization, e.g. to the components. To mention a very important aspect, an architecture has to guarantee that the non-functional requirements can be fulfilled by the later implementation built on it, e.g. efficiency and response time, scalability, reliability and safety. Bosch has described a very fundamental principle for solving the last aspect: for non-functional requirements, functional solutions are elaborated, which can be implemented easily [3]. Since non-functional requirements are often vague, incomplete and competing, methods for architectural decision-making have to be applied, e.g. [16].

For customizability and an ease of change, a system has to be variable and flexible. Components—and the architecture they are based—on can be prepared for a later adaptation, as well as for reusability and evolvability. In software product lines, measures for the preparation of future product variants are taken, to enable later changes with a low effort. To obtain information about possible future changes of requirements, domain analysis methods have been developed, e.g. [8]. Information about common and variable features of applications within a domain is represented by feature models (FM). UML models are not able to express variants, but through traceability links to FMs, variable aspects in different views can be described [12]. A feature model represents the properties of all systems in a domain structured by refinement relations in a tree-like structure, with additional relations between features in different subtrees. There is a distinction between features which are part of all systems in the domain—the so-called mandatory features—and ones which are only covered by some of the systems—the so-called

Manuscript received September 25th, 2008.

M. Riebisch is with the Technical University of Ilmenau, Germany (phone: +49-3677-691459; fax: +49-3677-691220; e-mail: matthias.riebisch@tu-ilmenau.de).

P. Sochos., was with the Technical University of Ilmenau, Germany. He is now with a telecommunication company in Switzerland.

variable features. Fig. 3 shows an example.

For software product lines and highly evolvable systems it is very important to apply the principle Separation of Concerns during the architectural design for an extended alignment of software elements according to features. This is essential because features are usually requested completely, and in this case a particular element is just added or removed without the need for further changes.

In this paper, the Feature-Architecture Mapping Method (FArM) is presented as a design method for developing independent components according to the principle Separation of Concerns. This method uses the FMs resulting from a domain analysis. The FMs are transformed similarly to the FAD method mentioned above until independent features have been developed which are ready for an implementation. The FArM method is briefly described in section II.

Furthermore, the paper presents a design process for components fulfilling the requirements mentioned above. Within the FArM method, other methods are integrated e.g. for the resolution of feature interactions and for the determination of the communication relations between the later components. Later in that process, a plug-in component is developed according to each feature. The design by contract principle [9] can be applied as a later design step in the process. This process is contained in section IV of the paper. The FArM method has been evaluated through its application in a number of domains including the mobile domain, the Integrated Development Environment domain and in a larger neural network trainer product line [15]. Some of the illustrating examples are taken from the MobilePL project which was described in an earlier publication, together with more information about the evaluation [14].

The contribution of this paper is seen in the presentation of a new design process based on the FArM method, and in the explanation of the correspondences between the different principles and methods during the architectural design. As a position paper, its intention is to inspire the discussion about the architectural design process and about the constraints to this process in terms of target implementation platform, design methodology, domain, and project complexity and size. It addresses several workshop topics, e.g. development processes, relating architectural methods to components, design for variability and extensibility.

II. Related Works

There are some related design methods which are based on feature modeling and which are considering reusability. The two most important examples are FeaturRSEB [5] and KobrA [1]. Both have been developed for the development of software product lines, and both exploit use cases to develop architectural components. In order to establish a mapping between features and architectural components, FeatuRSEB introduces traces while KobrA uses a so-called decision model in a very similar way. However, through the derivation of components from use cases the effect of feature scattering and tangling occurs. Since a feature usually affects more than one use case, several components are related to it. Furthermore, the implementation of one component is frequently contributing to several features. Both effects lead to a reduced flexibility and a higher impact of feature changes to other components. Nevertheless, the use case-driven approaches for component design result in a high encapsulation and a low coupling, and they enable an application in domains with medium complexity. Since a growing number of features lead to an exponential increase in the number of traces and decision models, the maintainability becomes a critical factor for complex systems. In our approach we relate the components to features instead of use cases to overcome this problem.

Generative programming techniques [10] aim at a composition of systems from elements, which are built according to the principle separation of concerns. If features are applied as the criteria of the separation, one element contains only those code parts that belong to one feature. If applying usual programming languages, these elements represent rather fragments than complete components. For a composition, the needed elements are weaved together using a generator. However, the resulting evolvability is limited because of hidden dependencies between the elements. The development of a generator requires a very high effort; it is only feasible for stable, mature domains. Furthermore, these technologies are not available on every programming platform, which is of special significance for mobile and distributed applications. Compared to the weaving of Generative programming approaches, plug-in components offer advantages in terms of the composition.

Among the several architectural design approaches the method Functionality-based Architectural Design (FAD) [3] which is part of the Quality-oriented Software Architecture Design method (QASAR) is mentioned here as a strongly related one. FAD uses core abstractions of functional concepts – the so-called archetypes – to derive architectural components. During the implementation, architectural styles and patterns are applied. For the implementation of non-functional properties, archetypes with functional solutions are introduced, which we consider as a very powerful principle. For the important step of the archetypes' identification FAD provides several hints, however, FAD considers it as a mainly intuitive, creative step. Hence there is no mapping mechanism the missing guidance is a challenge for the developers while developing complex systems. We embed the elaboration of functional elements for non-functional requirements into our method, and introduce a strong mapping between components and features instead of use cases.

III. Component Design by Feature Model Transformations in the Method FArM

The Feature-Architecture Mapping Method (FArM) aims at a high flexibility regarding changing requirements and a low

impact of such changes to the software architecture. Because many changes of requirements concern complete features, the method increases the mapping between features and the software architecture. The method supports the design of components, which ideally depend on only one feature. It therefore emphasizes the principle separation of concerns. If a feature is introduced or changed, the only change to the implementation consists in an insertion or replacement of one or a few components related to this feature. The method FArM is developed for component-based systems using a component model with plug-in interfaces, which are frequently used for example in mobile and distributed applications. FArM's advantages in comparison to generative programming approaches consist in an improved evolvability and in availability for a broad variety of programming languages and platforms.

FArM is performed as an iterative process with four phases as shown in Fig. 1. It starts with an initial FM (see Fig. 3 below) which is produced as a result of a usual domain analysis method, e.g. FODA. Each feature in the FM represents one or a set of requirements which are usually specified by other UML models, for example in form of use cases, behavioural model or domain model. The relations between the features and the elements of the requirements models are represented by traceability links. Since features are related to each other by refinement relations, the FM represents a functional decomposition of the requirements from a customer's point of view.

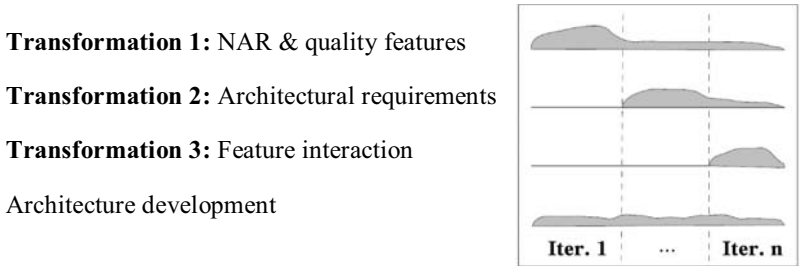

Fig. 1. FArM Phases

The first FArM transformation deals with two aspects, the so-called Non-Architecture-Related (NAR) and the Quality features. Examples for NAR features in a mobile handheld project are *Weight* as a physical feature or *Competitive Market Price*[1] as a business feature (see Fig. 3). Physical features are implemented directly by hardware solutions, and business features by managerial solutions. After resolution the NAR features are not present in the resulting transformed FM.

Quality features represent quality requirements which are important for the project's stakeholders. Examples for quality features are *Efficiency* and *Security*. Quality features have usually a broad impact on a software system. Since the method aims at a strong mapping between features and components, the goal of this first transformation is the production of a FM with only functional features whose responsibilities can be expressed as some sort of function and are thus implementable. For many quality features, functional solutions can be identified or created, in a very similar way as described by the FAD method [3] mentioned in section II. These quality features are not present in the resulting transformed FM. As an example, *Security* is satisfied as set of subfeatures including the feature *Firewall*, which can be implemented later by a functional solution. All remaining quality features are resolved by establishing profiles and then assigning quantitatively defined responsibilities to other features in a very similar way to the QASAR method [3] mentioned in section II. According to that method, the *Efficiency* feature from our example is refined by a quantitatively specified profile for time behavior and memory usage. Based on this profile, quantitative responsibilities are defined for other functional features. The resulting FM after this first transformation contains only functional features, which are later implementable to components.

The second transformation handles Architectural requirements. These requirements are not visible from the customer perspective, however they are important for a conformity to the architectural style or for architectural quality, e.g. for a robust and maintainable architecture. The goal of this transformation consists in the addition of new functional features or the extension of existing ones to satisfy these requirements. An examples for illustration is the introduction of a feature *HTTP Authentication* that can be integrated into the pre-existing feature *Web Browser*. This transformation therefore introduces aspects important to the system architects. They are added to the customer-related aspects stemming which are already contained in the initial FM. This second transformation contributes to a balanced mix between both aspects.

These first two transformations deal with the identification of most of the features needed to implement an architecture that tightly maps to the functional features. For each of these features a respective architectural component can be developed, matching the feature's specification. The next

[1] Both examples and all following ones are taken from the MobilePL project, which was performed as a product line case study in the domain of mobile phones. The product line is based on an API [14] and on Symbian OS. Data are exchanged between a handheld and a MobilePL enterprise server with a *Push* feature as a key element.

transformation further prepares the implementation of these components by defining and optimizing their interfaces.

The third transformation identifies and resolves feature interactions. The identification of feature interactions is based both on the domain specific feature communication needs, as well as on the hierarchy relations between the features in the FM. The identified feature interactions are then resolved and optimized. In this transformation, the various feature interaction resolution techniques are adapted [4]. This transformation effectively contributes to the decoupling of the respective architectural components and the enhancement of the system maintainability. The optimization of the feature interactions has also a positive impact on the communication between the respective architectural components, because the interfaces are derived from the feature interactions. It contributes to the encapsulation of components and the enhancement of the system's variability.

After the three transformations the architecture development phase follows next. The system components have been derived together with their interfaces. If not previously done, the developers have to decide for a specific reference architecture, a component model or an architectural style as the architectural context. Examples are the *Layers*, *Microkernel* or *Blackboard* architectural style. This step will extend some of the components regarding architecture related interfaces, too, for example if a *Microkernel* architectural style is chosen, the components' interfaces are extended by methods for dynamic loading and termination.

IV. Design Process for Plug-in Components

Component-based composition techniques using plug-in mechanisms contribute to a high flexibility, evolvability and variability of the resulting systems. Since they are well-suitable for mobile and distributed applications they are chosen as the target composition technique for the process presented here. However, the presented method can easily be adapted to other composition technologies as well.

Generally, the design process is very similar to those of many product line approaches since there is the common goal of a planned reusability (see Fig. 2). It consists of two main parts, one for the design of the reusable assets and the reference architecture, and the other for the development of a concrete product by composition and adaptation. Focusing on component design, we care about the first part, shown in the upper row of Fig. 2.

In the first phase of the process, the requirements to the components and the architecture are elaborated. State of the Art domain analysis techniques can be applied in this phase, e.g. FODA [8]. A FM is developed as a result, expressing a stepwise refinement of the features with their variability, for the applications in the domain. Fig. 3 shows an example from the MobilePL project. A proper consideration of non-functional properties has to be especially emphasized. UML diagrams for requirements modeling are related to the features to provide more detailed information about structure and behavior.

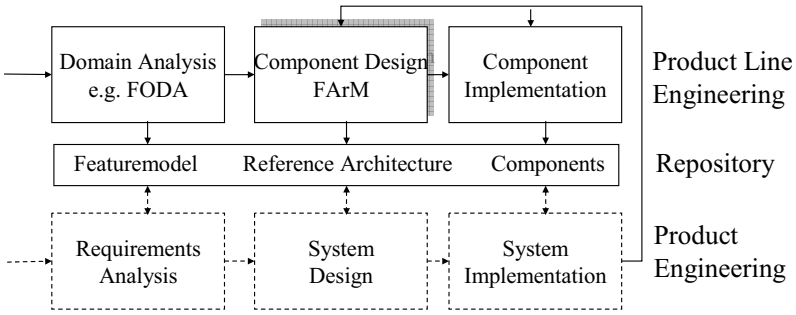

Fig. 2. Process Overview

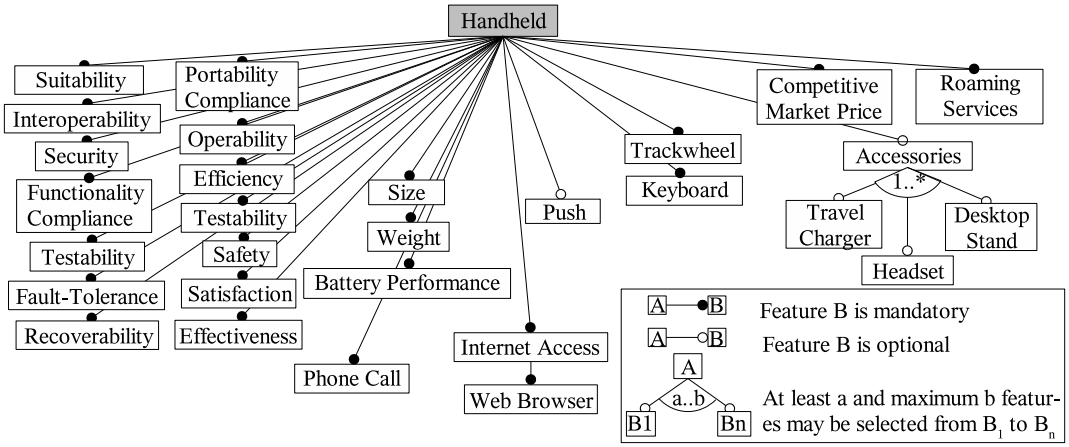

Fig. 3. Initial Featuremodel of MobilePL (partly)

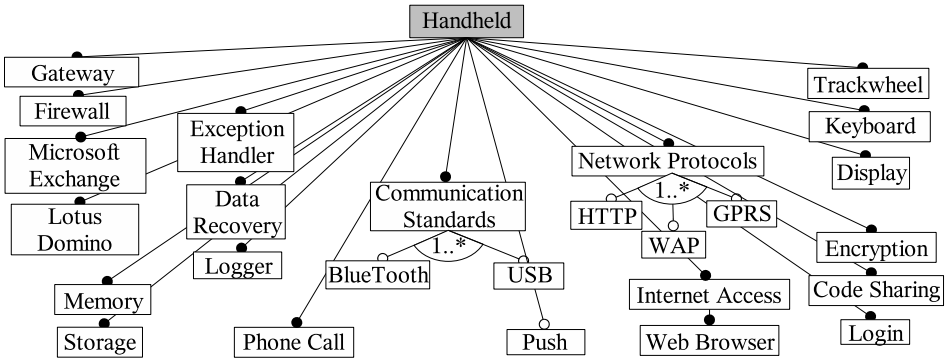

Fig. 4. Featuremodel of MobilePL after the FArM Transformations (partly)

In the next phase the three FArM transformations are performed as described in the previous section. New features are introduced; others are just copied or not taken over into the resulting FM of each step. In Fig. 4 parts of the resulting FM are shown. For details of the transformations we refer to other works [14][15] due to the space limitations.

From a more abstract point of view, the performed architectural design steps are strongly corresponding to those of other architectural design methods, e.g.

- the elaboration of functional solutions for non-functional requirements – similar to FAD,
- the resolution of component interdependencies – here performed by feature interaction resolution methods, and
- the definition of the components' interfaces – here by analyzing the communication relations between the features.

Since all these steps are performed by manipulating features despite of concrete components, the components can be designed and implemented afterwards considering the constraints of an actual component model, an existing system architecture and other sources. These design and implementation steps follow the principle of design by contract.

The design decisions are stored as traceability links relating the input and the result of each development activity. These links can provide the connection between UML models for design activities and the feature models. For the MobilePL, a prototype tool for FArM was built based on the DOmain Modeling Environment tool DOME. In other projects within an industrial environment we have successfully used a commercial requirements engineering tool, e.g. IBM's RequisitePro, for managing the linkage between features and design models.

V. CONCLUSION

In this paper an architectural design method and a design process have been presented which facilitate the goals of component-based systems: variability, flexibility and evolvability, furthermore the satisfaction of the non-functional requirements, the design of robust components with stable interfaces. These results are achieved by developing components for plug-in platforms which are highly independent and mostly corresponding to a single feature. The novelty of the approach consists in performing many of the architectural design activities by FM transformations. In this way, FMs become an integrated part of the architectural model. The foregoing activities for example the domain analysis fit seamlessly into the design process, and the same holds for the implementation and composition activities. In comparison to well-established object-oriented design methodologies – e.g. the Unified Process [7] – which are characterized as use case-centered, the FArM design method and the introduced process are feature-centered. The latter aspect leads to an applicability of the method for projects which focus on variability, reusability and planned pre-fabrication of software, which are typical for component-based development. The applicability has been proven by an application in different domains [15].

The presented work is part of a continuous research on the evolvability of software systems. Future works will deal with the utilization of traceability concepts for verification and validation purposes, as well as with effort for enhanced tool support and automation.

REFERENCES

[1] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jurgen Wust, Jorg Zettel: Component-based Product Line Engineering with UML. Addison-Wesley, 2002.

[2] Barry W. Boehm; John R. Brown; Hans Kaspar; Myron Lipow; Gordon J. MacLeod; Michael J. Merritt: Characteristics of Software Quality. New York, NY: North-Holland Publishing Company, 1978.

[3] Jan Bosch: Design & Use of Software Architectures-Adopting and Evolving a Product Line Approach. Addison-Wesley, 2000.

[4] Muffy Calder, Mario Kolberg; Evan H. Magill, Stephan Reiff-Marganiec: Feature Interaction - A Critical Review and Considered Forecast. In: Computer Networks, Elsevier, v. 41 n.1, 2003.. pp. 115 – 141.

[5] Martin Griss, John Favaro and Massimo d'Alessandro: Integrating Feature Modeling with the RSEB. Proc. ICSR98, Victoria, BC, IEEE, June 1998, pp. 36-45.

[6] Standard Software engineering -- Product quality -- Part 1: Quality model (ISO 9126-1). ISO, 2001.

[7] Ivar Jacobson, Grady Booch, James Rumbaugh: The Unified Software Development Process. Addison-Wesley, 1999.

[8] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University, 1990.

[9] Bertrand Meyer: Applying "Design by Contract". In: IEEE Computer, vol. 25, no. 10, October 1992, pp. 40-51.

[10] H. Ossher and P. Tarr. Software Architectures and Component Technology, chapter Multi-Dimensional Separation of Concerns and the Hyperspace Approach. Kluwer, 2001.

[11] David L. Parnas, On the criteria to be used in decomposing systems into modules, Communications of the ACM, v.15 n.12, 1972. pp.1053-1058.

[12] Matthias Riebisch, Detlef Streitferdt, Ilian Pashov: Modeling Variability for Object-Oriented Product Lines. In: Buschmann, Frank; Buchmann, Alejandro P.; Cilia, Mariano (Ed.): Object-Oriented Technology. ECOOP 2003 Workshop Reader. Springer, Lecture Notes in Computer Science , Vol. 3013, 2004, pp. 165 - 178.

[13] Johannes Sametinger: Software Engineering with Reusable Components. Springer, 2001.

[14] Periklis Sochos, Matthias Riebisch, Ilka Philippow: The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines. In: Proceedings 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS06), Potsdam, Germany, March 2006, pp. 308-316.

[15] Periklis Sochos: The Feature-Architecture Mapping Method for Feature-Oriented Development of Software Product Lines. Doctoral Dissertation, Technical University Ilmenau, 2007.

[16] Sven Wohlfarth, Matthias Riebisch: Evaluating Alternatives for Architecture-Oriented Refactoring. In: Proceedings 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS06), Potsdam, Germany, March 2006, pp. 73-79.