The Fourth International Workshop on Product Family Engineering PFE-4, Bilbao, Spain, October 3-5, 2001.

Architectural evolution of legacy product families

Alessandro Maccari[1] and Claudio Riva[1]

[1] Nokia Research Center, P.0. Box 407, FIN-00045, NOKIA GROUP
{alessandro.maccari, claudio.riva}@nokia.com

Abstract. Recent research has focused on the concept of product family architecture. We address the more specific case of legacy product families, whose life spans across several years and product generations.
We illustrate the method we use to describe legacy product family architecture and manage its evolution. To describe of the family architecture we use two separate documents. The reference architecture, which describes the abstract architecture that is instantiated in every product, and contains architecturally significant rules for adding new components to the system. And the configuration architecture, which maps the product family features into the various products, thus, allowing to model commonality and variability.
The concept of a family is an abstraction that automatically generates a new layer in every product. This layer includes all the software that is common to other products in the family, and is, naturally, less prone to change than the layer constituted by software which is specific to the product.
In certain domains like mobile telecommunications, when new products are added to the family, they tend to share most of the stable features that belong to legacy products. This phenomenon abstracts the issues of architectural evolution from the single products to the entire family scope.
We also sketch the process we follow to maintain the documents that model the product family architecture. Our approach combines reverse and forward architecting activities, and is currently applied in Nokia Mobile Phones. Research on the issues of architectural modelling is still insufficient: we propose some hints for future work.

1 Introduction

A product family is a set of products that share common features, chunks of functionality or architectural concepts. Products that belong to a family usually have common requirements and, consequently, share code, often in the form of components.

Product families are considered important because treating them as a separate issue may allow identifying and abstracting common problems and permitting to avoid duplication of software development and maintenance activities. An amount of research has been performed on the field of product families, and, recently, discussion concerning architectural evolution of product families has begun to take place. Three workshop series regularly provide a forum for discussion. Two of them are co-located respectively with ECOOP (International Workshop on Architectural Evolution) and

ICSE (Workshop on the transition from requirements to architecture, or STRAW), while the third one is the Product Family Engineering (PFE) workshop.

During the last edition of this workshop (Third International Workshop on Software Architecture for Product Families, IW-SAPF, Las Palmas de Gran Canaria, March 2000 [3]), an interesting point was brought up (by Alexander Ran and Jan Bosch): is product family architectural evolution an opportunity or a phenomenon to avoid?

As usually, the answer probably lies in the middle. We believe that family architecture stability is a positive fact, since it allows industries and software houses to develop products starting from a common, well-known framework. On the other hand, some product domains (e.g. electronics, operating systems, telecommunications) evolve at such a high pace that architectures may become unable to sustain market requirements in a relatively short time. [2] describes an interesting Nokia case study of product family architectural evolution.

The term "product family architecture" has different flavours in the literature. In our experience, when describing a family architecture two concerns should be separated: the architectural constraints (style) and the mapping of features into products.

Therefore, the architecture of a product family should be described by means of (at least) two different documents, representing two different architectural views.

The *reference architecture* (Section 2) describes the abstract architecture that is then instantiated in every product which is a member of the family; the document also describes the communication infrastructure.

The *configuration architecture* (Section 3) describes the mapping of common features into the various products of the family.

The evolution of a product architecture is necessarily tied to the evolution of the family architecture it is derivated from (Section 4). The reference architecture rules the evolution by constraining the types of components that may be added and imposing syntax and semantics on the messaging infrastructure. The configuration architecture describes the evolution by showing the change patterns of all product features.

Our experience has shown that the separation of these two concerns allows to minimize interaction between two different groups of people: the product stakeholders (who are mainly interested in configuration architecture) and the product architects and chief developers (whose work is mostly related to the reference architecture). The person or team who are responsible for the product family architecture should enable communication between the two groups and ensure consistency between the two documents.

2 Reference Architecture

A product family includes products that are built upon the same architectural style. Usually, every product that belongs to a certain family is built according to similar architectural rules. The style includes all such rules that are relevant at the architecture level. Examples of such rules may be "components must be either clients or servers", and "the communication between clients and servers should happen by means of asynchronous messages that conform to a certain format".

The reference architecture document describes the architectural rules that hold for every product that is part of a certain family. It can be thought of as the basis for the architectures of the products. The architecture of every single product of that family is an instantiation (or a specialisation) of the family reference architecture.

The reference architecture document should contain at least the parts described in the following sections.

2.1 Architecturally significant requirements

Architecturally significant requirements [5] include general requirements for the whole family, plus what we call lifetime requirements. These are requirements that must be satisfied at different stages of the software development, and concern different instances of the software. For example, at design time the software is made of logical components, at write time it is made of modules and at runtime it is made of tasks and threads. The requirements for all these phases that concern all the products that are part of a certain family should be described in the document.

We use mostly unstructured English text for this, although we are experimenting formal notations for specific issues. Examples are Petri Nets for describing feature interaction (see also 2.4) [4] and structured tabular descriptions for writing use cases using the guidelines provided by Alistair Cockburn [1].

Usually, architecturally significant requirements are pretty stable. We keep them with the reference architecture, although in theory they do not belong together. The reason is that changes in architecturally significant requirements usually map into changes in the reference architecture, and having them in the same document eases up maintenance (e.g. by making hyperlinks easier to type).

2.2 Architectural rules

This section of the document contains the system-level rules that all products in the family must conform to. Rules may describe what types of components (modules, entities and subsystems) may exist and what kinds of relationships are allowed between the different types of components. When implementing a new feature, developers should create components that conform to the architectural rules specified in this section.

An example of our reference architecture can be found in [2]. Currently, we use UML to describe architectural rules, although we find that this notation has some shortcomings for this purpose [7].

2.3 Communication infrastructure

The communication infrastructure is the means used by the various software components (modules, entities, subsystems) to communicate at runtime. The description of the infrastructure is crucial, as it fixes the structuring of interfaces and the communication paradigm. The effort spent on accurate description of the interfaces seems to pay back during the integration testing phase.

We will not treat communication infrastructure in this short paper.

2.4 Runtime architecture

Runtime architecture concerns the following issues:

- The allocation of processes to threads;
- The division of tasks at the operating system level;
- The interaction of different features at system level; an example is the keyguard feature that allows to lock the keypad to prevent accidental keystroke: when keyguard is activated, most other functions are disabled; this and other, more complex types of interaction need to be modelled [4].

We will not treat runtime architecture in this short paper.

3 Configuration architecture

The configuration architecture is used to organise the features of the product family and facilitate the derivation of the product architectures. The features are divided between the generic ones for the product family and the specific ones for the products. The features are also categorised in feature sets according to their domain).

The reference architecture imposes the conceptual rules for building the product architectures. The configuration architecture fixes the rules how to configure the architectures. This means how to organise the components that will implement the features, what interfaces will exist among the components, etc.

The product family features are used to configure the product family architecture, the product features configure the product architectures.

In the configuration architecture the description of the commonality and variability is done at feature (i.e. system-level, user-visible requirement) level.

4 Concrete product architecture

The concrete product architecture reflects the actual implementation of the product. The implementation is often not consistent with its design, therefore with reverse architecting [6] we extract it from the code. The reverse architecting process is driven by the reference architecture where the building boxes of the family architecture are

clearly defined. Clear and updated reference architecture is essential for extracting a significant description of the concrete architecture.

The analysis of the concrete product architecture allows us to argue about:

- violations of the architectural rules stated in the reference architecture
- points of divergence from the product design
- inconsistency with the product family

5 Family architectural evolution

As soon as a product becomes part of a family, it is often the case that the most stable features are also those that are shared with other products.

Thus, there is no more product architecture, but merely a (change-prone) structure when a product belongs to a family.

Instead, all the components that constitute the more stable part of a product tend to be shared with other products in the family. This evolution pattern is natural, since families tend to be structured around a few key requirements.

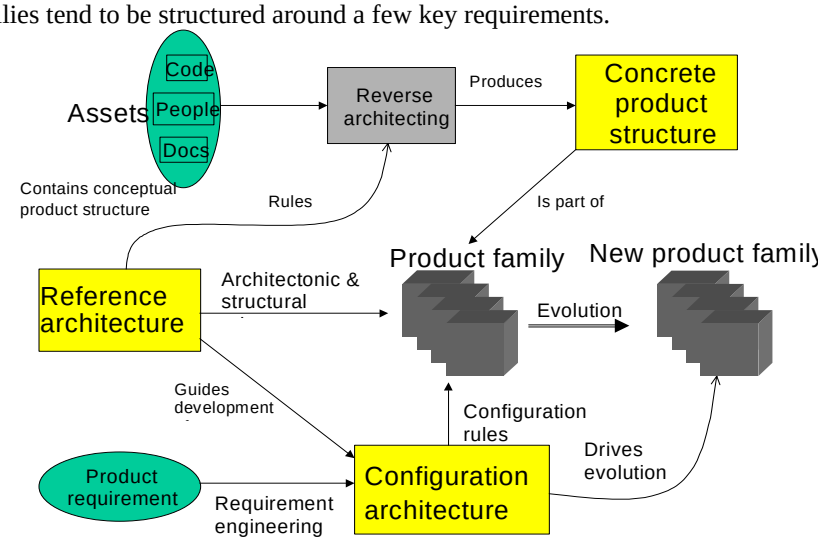

Figure 1. Family architectural evolution.

The process of evolution of our family architecture (and maintenance of the corresponding documents) is sketched in Figure 1. In short, the information on the product structure is contained in what we call assets; assets are code components, documents and people. In our experience, people are probably the most valuable assets, since their knowledge can hardly be transferred into documents or code comments. (This may be the source of a claim against the applicability of software engineering automation methods, but that's material for another paper). Reverse architecting activities (see section 4) allow us to extract the relevant architectural information from legacy assets on the basis of the reference architecture.

A possible output of reverse architecting is a set of diagrams showing the product structure. This, in turn, feeds into the product family architecture. At this stage, it is possible to identify violations of the reference architecture rules, and thus detect inconsistencies between the reference architecture model and the product structure. This might sound like a consequence of flawed architecting process, but has instead proved to be a frequent case when dealing with complex systems. It has proved to be surprisingly common to create change requests for the reference architecture document as a consequence of the introduction of a new product.

6 Conclusions

The approach we have shown is intended to be a report on current family architecture modelling and evolution practise in a large, distributed software development organization. We hope to generate discussion by illustrating how things are done with us, and comparing to how other organizations and academia approach the same problem. In the last few years, we feel we have improved the way we document software architecture for our mobile phone product family.

However, the practise of architectural modelling is still rather young in our company, and we still have room for improvement. In particular, we feel there is need for more research on how to combine the concepts (and the corresponding documents) of reference architecture, configuration architecture and product architecture. Also, similar approaches to architectural modelling should be compared to ours, and experimental validation should be provided.

References

1. Cockburn A., Writing effective use cases, Addison-Wesley, 2000.
2. Kuusela J., Architectural evolution, in Software Architecture, P. Donohoe (editor), Kluwer Academic Publishers, 1999.
3. Van Der Linden F. (ed.), Software Architectures for Product Families, Springer LNCS 1951.
4. Lorentsen L., Tuovinen A.-P., Xu J., Modelling feature and feature interaction of mobile phone software with Coloured Petri Nets, accepted for presentation at the Workshop on Feature Interaction in Composed Systems, ECOOP 2001, Eötvös Lorand University, Budapest, Hungary, June 2001.
5. Ran A., "ARES Conceptual Framework for Software Architecture" in M. Jazayeri, A. Ran, F. van der Linden (eds.), *Software Architecture for Product Families Principles and Practice*, Addison Wesley, 2000.
6. Riva C., Reverse Architecting: an Industrial Experience Report, *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE2000)*, Brisbane, Australia, 23-25 November, 2000.
7. Riva C., Xu J., Maccari A., Architecting and reverse architecting in UML, presented at the First International Workshop on Describing Software Architectures with UML, ICSE 2001, Toronto, Canada, May 2001.