

Chapter 9

Conclusion

In the introduction we discussed a number of trends in the field of embedded software systems. The development of software for these systems has become more and more challenging. This is due primarily to the increase in the size of the software. Furthermore, there has been a rapid rise in the amount of variation that the systems have to support; instead of delivering one type of product, different products are supplied that support different features. In addition to these trends, high demands are also placed on the quality of the products. The quality affects issues relating to the business, e.g. time-to-market, the product development, e.g. reusability, and the user perceived quality, e.g. performance.

9.1 Research Questions

In this section we will discuss how the research questions formulated in section 1.3 have been addressed in the previous chapters. We will first consider the research questions for the three research areas and then the overall research question. We will address these research questions in the context of large embedded systems for the professional market, as we performed the work presented in this thesis in the medical imaging system domain and the telecommunication switching system domain. This does however not mean that the results are only useful for systems of this type. For example, the design aspects can be applied to more software systems as additional views. As discussed in section 1.5, we used several techniques for our research: case studies, experiments and surveys.

9.1.1 Variation Mechanisms in Product Family Architectures

In addition to the main research question for this area, there are four sub questions. Let us consider these first:

- **RQ1.1:** What variation mechanism can be used to support variation in various kinds of services within a product family and what are its characteristics?

In chapter 3 we have introduced service component frameworks. A service component framework provides services to its customers that are again provided by one or more plug-in components. In a medical imaging product family architecture decomposed into just over 30 units, about 10 units contain one or more of these service component frameworks. This architecture has already been used to deliver several commercially successful family members to the market, using the component frameworks to achieve variation. This product family contains service components frameworks for different kinds of services, e.g. movement of the geometry, acquisition procedures, and field service functionality.

Despite these different types of services, the service component frameworks share a number of characteristics.

- A service component framework defines one or more roles related to the types of services that must be fulfilled by services from the plug-ins.
- The component framework provides an availability interface, via which the clients can query which services are currently available.
- The clients obtain access to a service via the component framework, not directly via the plug-in component. We must define for each role whether there should be at least one service provided for it and whether it is permitted for there to be more than one service for a role.
- Since different plug-ins may provide different implementations for the same service, a list of services must be provided. Each plug-in implements a subset of this list.

More of these characteristics can be found in section 3.3.

It should be noted that the service component framework is but one of the mechanisms that could be used to support variation in services. Another option, for example, would be to make one large component containing all services and using configuration parameters to activate the right services. Which variation mechanism is most appropriate will depend on the context in which it is applied (see also section 8.5).

- **RQ1.2:** How can service component frameworks be used within a product family architecture?

The use of service component frameworks in a product family architecture was discussed in section 3.5. We refer to two important topics:

- When using component frameworks, two views on the architecture are important. The top-down (decomposition) view is important for the architect to partition the system into smaller parts in order to master the complexity and to understand the system. On the basis of the required variability within a unit of the system, the unit designer defines one or more component frameworks. These component frameworks form the basis or skeleton of the system together with the ‘fixed components’. The bottom-up (composition) view is important to determine which (plug-in) components should be used for a specific product.
- The architect can use service component frameworks for various parts of the architecture. We have identified three types of component frameworks¹⁵. The application framework type deals with variation in the application domain. The technical frameworks deal with variation in the realization technology, e.g. a different geometry in a medical imaging system. Relationships between these two component framework types can exist to realize a feature. For example, for a specific end-user feature a plug-in with a certain movement may be needed in the technical geometry framework and a plug-in with an image acquisition procedure may be needed in the acquisition component framework. The infrastructural frameworks are the third type. An example is the field-service component

¹⁵ Fayad also defines three types of frameworks: system infrastructure frameworks, middleware integration frameworks, and enterprise application frameworks [26]. System infrastructure frameworks and middleware integration frameworks focus on internal software development concerns. This relates to our infrastructural frameworks. In the context of embedded systems, we have identified technical and application frameworks, which both realize product functionality (instead of the single category enterprise application frameworks). The technical frameworks are used to handle the variation in the underlying hardware; the application frameworks handle the variation provided to the end-users.

framework. These component frameworks form a stable part of the architecture.

This discussion assumes a number of architecture concepts like layering and decomposition into units. No domain-specific issues play a role here. We therefore believe that these concepts are more broadly applicable to embedded systems.

- **RQ1.3:** What are the benefits of using information models within the definition of component interfaces?

In chapter 4 we discussed a medical imaging platform that applies information models in the definition of interfaces. This technique does not capture all stable concepts in the syntax of the interface; certain concepts are captured in an information model describing the data that can be passed via interfaces. Positive experiences prompted wider use of information models in the platform. The number of medical imaging products containing components from this platform is increasing.

We discussed benefits of applying information models in section 4.3. Some of the most prominent ones are:

- Applying information models supports a declarative way of working; the data describes the work that needs to be done, not the individual steps. This leads to more stable interfaces.
- The information models can be adapted without affecting the syntax of the interface.
- Only the components that use that data need to know the semantics; intermediate components can simply pass the data without considering the content.
- The use of information models also supports forward and backward compatibility; for example, a receiving component can skip attributes it does not know.

Besides these benefits, there are also some drawbacks. For example, as the interfaces become more generic, they also become more difficult to use.

- **RQ1.4:** What are useful concepts for composable components in a platform to support variation?

In section 4.4 we discussed three concepts that support the handling of variation in the context of a platform with composable components.

These concepts are configuration data per component, required interface concept, and information model concept:

- Configuration data makes it possible to tailor a component to suit a specific situation.
- The required interface concept is important to make the dependencies clear; these dependencies are based on interfaces and not on components. This allows the exchange of components with different implementations, but with the same interfaces.
- The concept of information models supports stable interfaces. Furthermore, the declarative style enabled by information models allows for completely different implementations of the same interface.

As discussed in section 4.5, the experiences gained with these concepts in the medical imaging platform are positive. In addition to these concepts, there are also other concepts for supporting the handling of variation, e.g. the use of design aspects.

Having considered the sub questions, we will now look at the question for this research area. This question is: ‘Which variation mechanisms can be used and what are their properties?’. The sub questions addressed various mechanisms in detail: component frameworks, composable components and the configuration of a component. In addition to these variation mechanisms we have also identified abstract components and additional components, and we have grouped them into two levels: ‘below architectural level’ and ‘at architectural level’ (see chapter 8). For each variation mechanism we have discussed the properties. We have also discussed these variation mechanisms in the context of the platform coverage. For example, component frameworks can be used to partly cover a subdomain.

9.1.2 Multi-view Architecting, Quality Attributes and Design Aspects

In addition to the main research question for this area, there are three sub questions that we will consider first:

- **RQ2.1:** What are design aspects and how do they help to define an architecture?

A design aspect of a system is a coherent part of the functionality realized by the artifacts within the system that crosscuts the

decomposition of the system into artifacts. A design aspect plays a role in the design of a system.

The IEEE standard on Architectural Description [35] indicates the importance of having several views on the architecture. In chapters 5 and 6 we discussed design aspects which introduce views upon the system. Using design aspects was found to support the definition of the architecture of a medical imaging system in several ways, including:

- Explicitly dealing with design aspects within an architecture enables uniform handling of them throughout the system.
- Using design aspects introduces separate views which can be documented separately, helping to structure the architecture documentation. The description of a design aspect can contain a reference to the supported quality attribute(s), the general policy, and how it should be realized in the artifacts, e.g. using certain design patterns.
- Since design aspects deal with generic functionality, such as error handling, checklists (e.g. based on Figure 6-5) can be used when identifying relevant concerns within an architecture. Several examples have been given of ways of realizing quality attributes with design aspects. For example, reliability can be supported by implementing self-tests, graceful degradation functionality and error handling functionality in the artifacts of the system.

In general, design aspects help by enhancing the system with additional structure, thus managing complexity. Since design aspects deal with generic functionality, their use is not restricted to a specific application domain. This is also evident from related techniques such as aspect-oriented programming. This supports a broader applicability of design aspects.

RQ2.2: How can design aspects be used to support the design process?

In chapters 5 and 6 we illustrated how the design and testing phase can benefit from using design aspects:

- In the design phase, design aspects were found to be helpful by structuring the artifacts and the related documentation. For example, the documentation template contained individual sections for various design aspects. Furthermore, various UML diagrams, e.g. a class diagram and a sequence diagram, can be

specifically geared to a single design aspect, e.g. initialization. The design aspects have been applied at various levels of decomposition.

- The structure provided by design aspects has proved to be useful for checking whether all artifacts have dealt with the design aspects correctly. During review of the design documentation, attention is paid to the design aspects. Design aspects have been supported via automated tooling that checks whether the implementation is correct, e.g. whether the interfaces related to design aspects are present.

So, we have found in our study that design aspects are useful for enhancing the design of a complex system with additional structure. An issue that needs to be investigated further is how tooling can support this way of working (a verification tool has already been developed, but specifically for this medical imaging system).

- **RQ2.3:** How do the components of a platform benefit from design aspects?

We have seen various systems in which certain concerns were not handled in a cross-cutting way. One such a system, for example, contained a central error handler for all hardware (similar to the upper half of Figure 6-10). This meant that when a new component was added or an old one was removed, the error handler had to be modified, too.

In section 6.3.11 we argued that each component within a platform should deal with all relevant design aspects. When such a component contains all the required design aspects, other components do not have to be adapted when it is added or removed. This also holds for plug-in components of component frameworks, as discussed in section 3.4.2. If each replaceable component in a platform were not to handle all relevant aspects, it would be almost impossible to construct a family of medical imaging systems.

After considering the sub questions, we will now look at the question for this research area. This question is: ‘How can design aspects and quality attributes help to build complex systems, and product families in particular?’. The sub questions that have been addressed above illustrate the relevance of using design aspects as important views during architecting and design. Although design aspects and quality attributes are important for software development in general, they are especially relevant for product family development. The reason is that the availability of these additional views supports the careful analysis of important quality attributes like configurability and evolvability.

Furthermore, by making the artifact aspect-complete, they are easily reusable in different family members.

9.1.3 Product Family Development and Evolution

In addition to the main research question for this area, there are three questions that we will consider first:

- **RQ3.1:** How are the business, processes and organization affected by product family development?

Applying product family development will affect the business, architecture, process and organization. In chapter 7 we presented a range of factors that must be considered when applying product family development. Some of the most important ones are:

- The business must determine the goal and scope of the product family. This will determine for example the functionality provided by the platform and the product family architecture; things that are difficult to change later on.
- In the process and organization area, it must be determined how the reusable assets and the product are developed. If separate groups are used, it is important to establish good communication between them. For example, if the reusable asset development does not receive feedback on time, this may lead to 'floating' of the asset development, i.e. the development becomes too generic and deadlines are missed.
- In the initial stage, care must be taken concerning the 'weight' of the introduction. A full-blown introduction of a product family approach is very risky, e.g. due to factors such as the large number of changes that must be managed, the acceptance by the developers and the long lead-time and high costs involved in making the reusable assets.

These factors are based on a number of case studies. We have seen similar observations in other studies. The work presented here is not complete in the sense that it can be used as a kind of cookbook to set up product family development. It does however provide important factors that have to be considered when applying product family development.

- **RQ3.2:** How can platforms of product families be classified?

In chapter 8 we introduced two classifications for product family approaches: platform coverage and variation mechanisms. We chose these classifications because they deal with the properties of a product family platform, and a platform forms the basis for the successful development of a product family. By the coverage of a platform we mean the number of artifacts already provided by the platform in proportion to the number of artifacts that still have to be provided by the application engineering in order to derive a complete product. The types of variation mechanisms used within a product family determine the way specific products can be made from the platform.

These classifications are based on four case studies. Although these case studies had various aspects in common (embedded system, professional market, several million LOC, etc.), there were also differences, such as the coverage and the employed variation mechanisms. Other classifications can be found in the literature, but we found these two classifications very useful in comparing different product family approaches.

- **RQ3.3:** What introduction scenarios exist for product family approaches?

We used the two-dimensional classification of platforms to answer this question. In section 8.6, we described two introduction scenarios:

- The first scenario starts with one product (or a few products) and adds variation over time. This means that, initially, all subdomains are covered completely.
- The second scenario is to start with a few well known components and expand the set over time. This means that, initially, a few subdomains are completely covered.

Both scenarios have a limited introduction ‘weight’. As discussed in section 7.5.3, a more lightweight introduction has a better chance of succeeding. We have encountered these scenarios in two of our case studies. Other introduction scenarios are possible, but the ones proposed here are favorable due to their low introduction barriers.

Having considered the sub question, we will now look at the question for this research area. This question is: ‘What (non-technical) factors are important for defining and evolving a product family approach?’. The various sub questions have addressed various parts of this question. First of all, you have to determine

how the product family supports the business strategy and what the scope is. This impacts the choice of the platform coverage and the type of variation mechanisms. The processes must be defined and show how the platform and the products will be made. A decision must be made as to how the development is to be organized, and how acceptance of a new way of working is to be promoted, e.g. by starting small.

9.1.4 Overall Research Question

The overall research question formulated in the introduction was:

How is a product family platform to be defined and what aspects must be considered when introducing and applying a product family approach?

It is unfortunately not possible to give an answer that completely covers this question. Instead of answering the complete question, we focused on a number of elements of this question. In the first part of this thesis we focused on variation mechanisms and how they can be used in the product family architecture. Such mechanisms are indispensable for supporting the variation in a product family. Since product families are complex systems, it is important to master the complexity. An important means in this respect is to use views on the architecture. In this context we have discussed the use of design aspects. Using design aspects helps to make components self-contained, increasing their reusability. In the third part we indicated that non-technical issues also play a role in product family development. We have identified several factors that must be considered for product family development. So as to be able to reason about product families and their platforms, we introduced two classifications: platform coverage and variation mechanisms. These classifications helped us to reason about introduction and evolution strategies.

9.2 Contributions

In this thesis we have covered three important topics in the context of product family approaches. In the previous section about the research questions, we have already listed the contribution made by this thesis by answering the research questions. In this section, we will summarize the contributions.

9.2.1 Variation Mechanisms in Product Family Architectures

In chapter 3 we have described a product family in which service component frameworks are applied for achieving the required variation across the family

members. We described what the characteristics of such a framework are, as already summarized in the answer of research question 1.1. We also described how such a component framework should be developed and what issues need to be taken into account. Furthermore, the different types of component framework regarding multiplicity are listed.

In addition to the component framework itself, it is also important to embed component frameworks into the architecture. We have illustrated how they can be used in the architecture. Component frameworks can be applied in different parts of the system – in the application part, in the technical part or in the infrastructure part. We have furthermore illustrated how several component frameworks can be involved in realizing a feature for an end user.

In chapter 4 we illustrated another product family approach. This approach is based on the composition of components that are provided by a platform together with specific components to form product family members. We have described how the components can be defined in such a way that they are reusable across different family members. Three concepts play an important role in achieving this reusability. First of all, the components can be configured via configuration data. The required interfaces are then described explicitly. This allows the reuse of a component in different contexts, as long as the required interfaces are present. The underlying implementation of these interfaces does not have to be the same. A third concept is that of the information model. The use of information models offers several benefits. One important benefit is that it supports a declarative way of working. This makes it easier to provide different implementations for the same interface.

9.2.2 Multi-view Architecting, Quality Attributes and Design Aspects

In chapter 5 we illustrated the importance of having multiple views. The quality attributes take into account the behavior that the product should exhibit. This behavior can be realized in different ways. One important way to realize quality attributes is to use design aspects, which form orthogonal views on the primary decomposition of the system. We have illustrated how design aspects can play a role at different levels of decomposition of the system. Furthermore, examples are given of how quality attributes can be ‘translated’ into design aspects. Especially in the context of product families, design aspects are relevant. If a product family platform contains aspect-complete components, these components are more reusable for different products.

In chapter 6, which also deals with quality attributes and design aspects, we have shown how design aspects can be used in the design process. Since aspects

are orthogonal to the primary decomposition, they are relevant to the artifacts of the primary decomposition. This is useful, for example, when defining a template for describing the design of a component. Such a template can already have sections that deal with each relevant design aspect. It also makes it possible to indicate which design diagrams are relevant for use. An example here is the use of sequence diagrams that describe the initialization aspect. Furthermore, aspect-related interfaces can be prescribed for each of the components in a system. By using standard interfaces for design aspects, it is possible to make architectural checks in an automated way, e.g. ‘Does each component provide the obligatory aspect-related interfaces’.

9.2.3 Product Family Development and Evolution

In chapter 7, we have sketched the importance of non-technical topics in the context of product family development. To achieve successful product family development, the architecture and other technical issues form only a part of the story. The technical decisions have to be aligned with the non-technical issues. We have given several examples of the non-technical issues and their relevance. The introduction of a product family approach to an organization is an important subject; if the introduction fails, this will be a major set-back for the company. We have also paid special attention to the business, architecture, process and organization when introducing a product family approach.

A product family approach should be selected on the basis of the context in which it will be applied. As a consequence different product family approaches exist. In chapter 8, we have therefore introduced a classification scheme. This scheme is based on the platform coverage and the variation mechanisms used. We have used it to explain the properties of the different approaches and to help select one. Since the requirements on the product family approach may change over time, it is important to have evolution scenarios. We have described two ways of starting a product family approach and sketched the subsequent evolution paths.

9.3 Open Issues and Future Work

Various issues discussed in this thesis can be the subject of further research. Below, we will discuss briefly some of the areas that require further research.

- **Case studies in other domains.** The work presented in this thesis has been based primarily on four case studies. The different characteristics of these case studies mean that they form a good basis for the validation of the ideas presented here. However, the conclusions presented here

would be strengthened by the use of a greater number of cases studies. The case studies used here are all embedded systems for the professional market. As indicated in section 1.4, these systems share a number of characteristics. Although we feel that the results are more widely applicable, validation of the result for other types of systems will help to underpin this.

- **More quantitative studies.** The work performed in this thesis is of a more qualitative than quantitative nature. It would be very desirable to have quantifiable data. For example, it would be relevant to know what the benefit is when design aspects are used in the development of a product family, e.g. in the form of savings made on development and maintenance costs. The performance of quantitative case studies requires additional work in defining which data should be collected, and the process of collecting the actual data can be labor intensive.
- **Embedding in a larger development process.** We have performed two studies in which we introduced the use of component frameworks and design aspects in the development of a large medical imaging system. These studies have been carried out in the context of a specific development process, using specific tools and realization technologies. Although we are convinced that these techniques are applicable in other contexts, further study would be useful to back this up. For example, the component frameworks were realized using COM. We can study how these frameworks can be realized with other technologies. UML diagrams have been used to apply design aspects. Further study could investigate how greater support could be given by the tooling for design aspects. Furthermore, the use of quality attributes and design aspects fits into a larger multi-view architecting approach. Further research is needed in this area.
- **Elaborate scheme for product family selection/evaluation.** We have introduced a scheme for the selection of a product family approach for a specific context and the evaluation of existing approaches. This scheme is based on some technical issues and some non-technical considerations. A lot of work still needs to be done to make this scheme more complete. This will require further research into additional case studies.

