Chapter 4
Components, Interfaces and Information Models within a Platform Architecture

Abstract

In this paper we describe our experiences with the development of a platform in the medical imaging domain. Three important ingredients of this platform are components, interfaces and information models. We will explain the requirements for the platform, why these three ingredients have been chosen, and our experiences when using this approach.

4.1 Introduction

Products in the medical imaging market are becoming more complex and more diverse, must support easy extension with new features (feature propagation across products), should have similar appearance to its users, and have to be of high quality. A short time-to-market and limited development costs are also important factors. These requirements should be met by a product family that covers a large part of the market. The development of a product family and its individual members (i.e. single products) can be supported by a shared family architecture.

Similarities can even exist between various product families. These similarities can be exploited in a similar way as they would be within a single product family. Such a group of related product families is sometimes referred to as a

product population [72]. A shared architecture can also be defined for such a product population. Based on this architecture, we can define and provide assets that can be reused across the product families belonging to the product population.

In this paper we describe our approach for components, interfaces and information models within a platform for a product population. This approach is applied to a project, which aims at the delivery of common components across a number of different product families. These product families share the characteristic of acquiring and processing digital images of the inside of a human body. The platform can also be directly used for products; the 'indirection' via a product family is not required. Section 4.2 introduces the requirements and set-up of the platform. In section 4.3, we elaborate on the ingredients of the platform, i.e. components, interfaces and information models. Section 4.4 describes the approach of using these ingredients to build a platform. Finally, section 4.5 presents result of practical experiences with the approach, followed by concluding remarks in section 4.6.

4.2 Product Population Platform

In the previous section, we introduced the arguments for a platform approach to support the development of products in the medical imaging domain. The platform has to support a number of properties. The most important ones are:

- **Open systems.** The platform is intended for various systems within the hospital. These systems must be interconnectable and must be able to incorporate new functionality.
- **Forward/backward compatible.** It is desirable that older and newer products made with the platform can work together. Also within a product, it must be possible to combine functions of different versions.
- **Independent life-cycles.** The platform contains different groups of coherent functionality. Each of these groups should have an independent life-cycle, so that it can be updated without affecting the rest of the platform.
- **Configurability, extendibility.** The composability/configurability of individual specific products from specific parts of the platform is very important. It should also be easy to extend products with new functionality.
- **Complexity management.** When defining a platform that must be applicable for several product families and will evolve over time, it is important that the complexity is manageable.

- **Outsourcing and third-party software.** In the light of the growing size and complexity of the software, we want to be able to buy-in certain parts or to outsource well-defined functionality.
- **Portability (technology independence).** The platform is used for different products that may have different operating systems or component technology. The platform must be usable in these contexts.

To support these properties, the product population platform will consist of a number of components. These components have clearly defined interfaces. Information models are defined for some of these interfaces, describing the semantics of the data that is exchanged over these interfaces (components, interfaces and information models are explained in section 4.3). Specific products can reuse the components, taking the interfaces and information models into account.

The term 'platform' has different meanings, depending on the context in which it is used. For example, when looking at operating systems or middleware such as COM, the platform is a piece of infrastructure functionality on top of which you can build your own product. Another way in which the term platform is used is for a collection of frameworks to which specific functionality can be added via plug-ins (see [113]). In the context of this paper, we define a platform to be a set of generally reusable components. The users of this platform are free to decide which components to use. This is illustrated in Figure 4-1. Here, two product families and one product are depicted based on the platform. Each of them is composed from product-specific components and components that are selected from the platform.

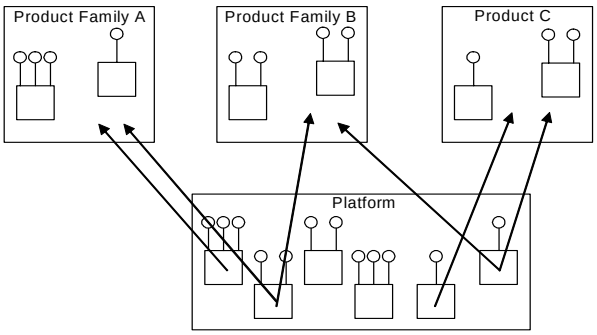

Figure 4-1 – Platform Set-up

4.3 Components, Interfaces and Information Models

The three main construction elements for the platform are components, interfaces and information models. They will be discussed in this section.

In our platform, components are units containing re-usable functionality with explicit interfaces that offer:

- a quick delivery of existing functionality through stable interfaces;
- composability of products from components by product groups;
- independent life-cycles of components and products, allowing incremental updates;
- easy distribution/installation of updated/new functionality;
- an opportunity to (re-)use third-party and legacy software;
- improved control over outsourcing;
- integration of heterogeneous technologies (operating system, programming language).

In addition to the components, interfaces are essential for our approach. Important characteristics of these interfaces are:

- access points to clearly defined functionality for use by other components;
- contracts between component creators and component users;
- components must implement interfaces in their entirety (no optional methods);
- components usually implement a set of logically-related interfaces;
- the same interface may be implemented by multiple components;
- the interfaces should be stable, the implementation can be flexible.

Interfaces are closely related to components. This relation is illustrated in Figure 4-2. A component has two types of interfaces, namely:

- *provided* interface; the component guarantees that it will implement the functionality associated with the interface
- *required* interface; the component accesses functionality through this interface and relies on the functionality to be implemented *outside* the component

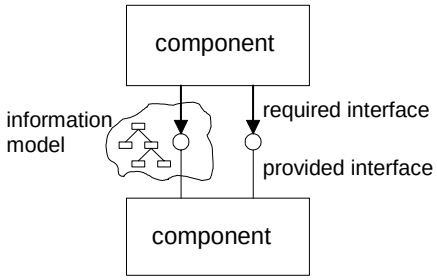

Figure 4-2 – Components, Interfaces and Information Models

The interfaces and components are smaller parts that are defined in the context of the overall platform architecture. In addition to these interfaces and components, there are concepts that are relevant at several places in the architecture. An example of such a concept in the medical domain is a medical image. The concept of an image is relevant when displaying an image on the screen, when printing an image on film, or when storing an image on a CD. The image concept is a complex structure, and includes for example the pixel matrix and attributes that refer to the acquisition of the image. Concepts like this are included in a so-called 'information model'. An information model captures relevant concepts from the domain, and is independent of the underlying technology. A number of information models have been defined. It is possible for one information model to build upon another (extending it).

As an example, Figure 4-3 shows part of the imaging information model, which is based on the DICOM (Digital Imaging and Communications in Medicine) standard. Each of the objects in the structure has a number of attributes. There may be attributes that are specific for specific product families. It can also be the case that specific objects are added for individual product families.

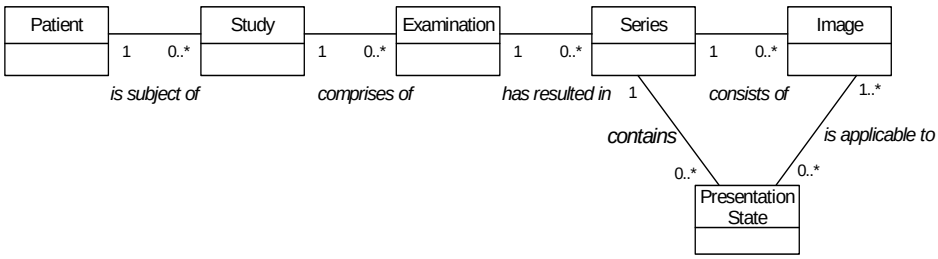

Figure 4-3 – Information Model Example

The concepts in an information model play a role at several places in the architecture, between various components. For example, an application component makes a request to a print service to print an image. One way of realising this functionality is to let the print service provide a number of

methods, each with a number of parameters, so that the application component can pass all the individual attributes that belong to the image and the attributes that control the printing of the image. However, instead of this possibility, the request is passed via one method call as a structure of data objects as defined in the information model. As a consequence, there are fewer and simpler interface methods, and the semantics are moved to the information model.

Figure 4-2 illustrates how an information model is related to an interface. An information model defines the structure and semantics of the data objects that are exchanged between components via a certain type of interface. Such an interface enables data objects that adhere to the related information model to be passed. Two interacting components both must know the information model in order to deal with the data that is exchanged.

Changes can occur within an information model, as the concepts within the domain may evolve. This means that several versions of an information model can exist. A certain component therefore provides a combination of an interface together with a particular version of the related information model. Since these two elements together actually determine the functionality that is provided by the component, they form the complete interface provided by the component.

Each of the product families supports the basic set of concepts that are defined for a particular information model. However, there may also be extensions that are specific for a product family. This means that when data is transferred from one system to another, the receiving system will at least understand the basic set. The receiving system may or may not ignore additional elements or not, depending on the information models used by the sending and receiving systems.

The main benefits of using information models within the definitions of component interfaces are as follows:

- **Reuse within several interfaces.** An information model captures concepts that play a system-wide role in the interaction between components. For example, an image can be printed, stored, or displayed, requiring interaction with different components. These concepts can easily be reused in different contexts by defining them as data objects that can be passed via interfaces. Interfaces might also be reusable for several information models. This is likely when one information model is an extension of another information model.
- **Stable methods in interfaces.** Since the syntactic part (the methods and their attributes) of the interfaces become smaller, they will remain more stable than traditional interfaces. This means that fewer new interfaces will be introduced during evolution of the system than when

applying the 'COM-rule', that is, when an interface needs to be modified, a new interface has to be introduced.

- **Handling concepts as separate entities.** As in object-orientation, we intuitively deal with concepts in the real world as objects in an information model. These objects can be exchanged as entities over interfaces, and can be passed from component to component. This is different to the situation where an object's attributes have to be put in the parameters of a method call, one by one. This would lead to complex method descriptions, particularly when concerning object structures consisting of several hierarchical levels. It is even possible that when a data object is passed through a number of components, the intermediate components do not know the complete semantics of the data object, but simply pass the data object to the next component. This allows the introduction of generic services that can handle data objects with slightly different structures and semantics, e.g. a service that queues and handles print requests.
- **Data objects can be stored (persistent).** Since the data that is passed between components is in the form of data objects, it can easily be stored. For example, the request to print an image with all its data can be handled by a print service, and can later on be passed to a printer that is free. The same object structure can also be used to store these objects. Also for the exchange of information between interconnected system, the data objects are used.
- **Support for forward and backward compatibility.** Since the methods of the interfaces remain stable, each component can receive data objects belonging to an older or newer version of the information model. If the version of the information model used by the requesting component is older, the receiving component knows how to deal with the data. If the version is newer, some new attributes in the data may have to be skipped, as the receiving component will not understand what they mean. This mechanism should of course be used with care, since although the methods of the provided and required interfaces are compatible, the combination of different versions of an information model may lead to undesired behaviour.
- **Enabling declarative way-of-working.** The components in a product population platform have to be used together with components from specific products. This means that a stable level of interaction must be chosen for these component interfaces. By applying data objects, a declarative way-of-working is enabled in which the functionality is requested in terms of end-results (more *what*) and not in a sequence of steps (less *how*). So, the request to perform some action is captured in a

data structure and is passed to the performer of that action. This kind of interaction is more stable than interaction based on interfaces that have methods for each individual step.

- **Data driven tools.** The use of information models increases the value of tooling that is based on data structures. As mentioned earlier, the data objects can easily be stored in a database. Another example is the generation of test data, based on an information model that defines the allowed structures and values for the data objects.

This information models approach follows the trend of separating syntax and semantics, which can also be seen in XML. The information models are not applied for all interfaces. Usually those interfaces that deal with relevant system-wide concepts have related information models. There are specific interfaces for local interaction between components, since the advantages mentioned above mostly apply to the interfaces related to relevant system-wide concepts.

In addition to the advantages of the information model approach, there are also some drawbacks. One of these drawbacks is that it becomes more difficult to see what an interface is used for, since the methods of the interface are more generic. It also requires a different way of working than in the situation in which ‘normal’ interfaces are used.

4.4 Approach

In this section, we give a general description of our platform approach (section 4.4.1), the main architectural styles (section 4.4.2), the way diversity is supported by the platform (section 4.4.3), and look in more detail at how the components, interfaces and information models are defined (section 4.4.4).

4.4.1 Platform Definition

As mentioned earlier, the first aim of the shared approach is to arrive at an architecture that is shared across the various medical imaging systems. Such a shared architecture is essential to be able to define shared components, interfaces and information models. All parties involved, both the platform group and the product groups, contribute to an architecture group that defines the shared platform architecture. The next steps are as follows:

- To define shared interfaces and information models, enabling the exchange and sharing of medical (imaging) data;

- To develop and deliver a set of medical imaging software components, that adhere to the agreed interfaces and information models, and that can be used within the various products.

A prerequisite for this approach is that the information models and interfaces are explicitly defined and managed. To achieve this, there are two working groups, one for the information models and one for the interfaces, each containing members from the various platform and product groups. The focus is on the standardisation of functionality that already existed in multiple systems. As a rule of thumb, a component is part of the platform when it is required by at least two products.

The platform group defines and realises reusable components for the platform. These components can be built quite independently of each other. They will be released when they are finished, and will be integrated into the platform in a next step. The product groups take a version of the platform and build their products on top of it. Any product may use its own selection of reusable assets.

The product groups also use components that they have built for their own purpose. If these components comply with the interfaces of the shared architecture and are useful to other product groups, they may also be integrated into the shared platform. Note that this integration step will often need a redesign, both of the component itself and of those component interfaces that are not yet available. The reason for this is that the functionality that the component provides may have to be slightly adapted to make it reusable within other products.

4.4.2 Platform Architecture

The shared platform architecture defines a number of layers. The layers are (from bottom to top): base layer (containing infrastructure functionality), service layer (medical service components) and application layer (medical application components). The application components provide integrated functionality that forms an application. Applications are integrated to form products. The functionality of the application components is realised by using the functionality of the underlying service components. Each service component can thus be reused for several applications.

As shown in Figure 4-4 below, the information models play an important role in the interaction between application and service components. These information models form a stable factor, allowing exchange of services and applications. Furthermore, the data objects that are exchanged between application and services can be made persistent by storing them in repositories. Since each of

these repositories supports the same interfaces and information models, these repositories can be exchanged. When a product family wants to incorporate platform services and applications, it has to provide its own implemented functionality, e.g. a specific repository implementation, via the defined interfaces and information models so that platform applications and services can be reused. The use of data objects also supports the co-operation of services and applications on distributed systems.

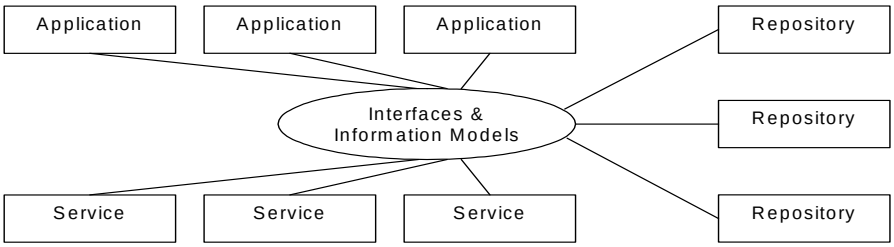

Figure 4-4 – Applications and Services

4.4.3 Dealing with Diversity

The product population platform is used in the context of different medical imaging products, each with its own architecture. Each of these products provides its own functionality and requires specific functionality from the platform. These products will each evolve over time, as will the platform. This means that the platform must have an excellent level of support for diversity and evolvability.

The architecture of the platform contains three main concepts that support the diversity needed across products and product families, as explained below:

- configuration data per component;
- required interface concept;
- information model concept.

Each component within the platform has a required interface for configuration data. During start-up configuration data is read from the configuration database and is used to initialise each of the components. We will not elaborate further on this mechanism.

The required interfaces form explicit points of variability in the platform architecture. They allow the exchange of components that have the same provided interfaces, but have been implemented differently due to specific requirements. A simple example is the Logging interface. This is a provided

interface of the Logging component within the platform, which is responsible for storing the logging data. The Logging interface is also a required interface for all other platform components. A specific product family is however not obliged to use the logging implementation provided by the platform. If there are specific logging implementation requirements, the product family can implement a specialised logging component that adheres to the Logging interface. This is illustrated in Figure 4-5. All platform components perform logging through the Logging required interface, so none of the other components need to be changed.

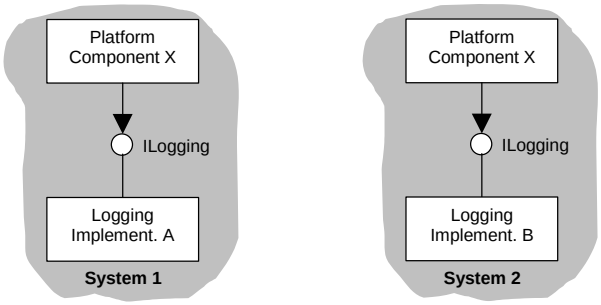

Figure 4-5 – Different Implementations for One Required Interface

Platform components should make as few assumptions as possible about their (execution) environment. As a design rule required interfaces should be defined for each function that may require variability in one or more products.

The third variability mechanism is the use of abstract, general interfaces in combination with information models. In order to keep the interfaces simple and stable, most of the semantics are passed on as structured data objects that adhere to a specific information model, resulting in a declarative style. This allows a single implementation of the generic component that can support requirements from different products (see Figure 4-6). If the differences in requirements cannot be met by one single implementation, however, the declarative style facilitates a completely different implementation, since the individual steps to arrive at the result are not defined in the interface.

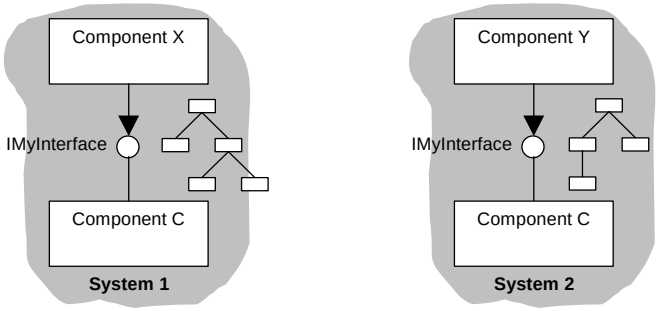

Figure 4-6 – One Implementation Supporting Different Requests

4.4.4 Defining Components, Interfaces and Information Models

The interfaces and their related information models play an important role in the development of the platform. We could say that instead of a component-centric architecture description we have an interface-centric architecture description. In a component-centric approach, the focus lies on describing the various components and not on how these components interact with each other. In an interface-centric approach, the focus lies on the interaction patterns between the components, thus putting the focus on interfaces. The following steps can be identified when defining interfaces and components for the platform:

- Determine which functionality must be provided, which kinds of components play a role in the solution, and which roles they play in collaborations. This is based on the domain knowledge.
- Based on the interactions between these kinds of components, define the interfaces and the information models for the information exchanged. These interfaces determine which roles components can play. The interfaces have to enable a declarative way of working.
- Using the interface specification, and taking the functional requirements into account, define the various components. Such a component definition heavily relies on these interface specifications, i.e. they can be seen as compositions of interface specifications. The functionality is then realised inside these components.

4.5 Experiences

In the previous sections, we described the approach for using components, interfaces and information models. Some of the experiences with this approach have taught us that:

- clear separation of platform internal and external interfaces (and components) is important;
- when moving from a component-oriented approach to a more interface-oriented approach, do not forget good definitions of components;
- prototyping of interfaces and components is an important means to validate the design choices at an early stage;
- starting on a small scale gives the opportunity to fine-tune the process and to gain experience for next developments;
- it is important to involve all parties (platform and product groups) in the relevant decisions about the interfaces, information models and components;
- the declarative approach supported by the data objects makes it easier to build systems from components that are developed by different teams;
- the design should not become too generic; this leads to unnecessary implementation complexity and increased integration effort;
- the chosen interfaces and information models support the required diversity in the specific components.

4.6 Concluding Remarks

In this paper we have described our approach and experiences with components, interfaces and information models in the development of a product population platform for medical imaging products. Interfaces, information models and components support the required platform variability in the following ways:

- By paying explicit attention to interfaces, they will be usually more stable. The explicit handling of required interfaces means that each interface can be implemented by different components, each with their own behaviour. This enables diversity.
- The interfaces are expected to remain stable. The variation, both at a particular moment in time and over time, is enabled by the information models. An information model can have a generic part that applies to all products, in addition to product-family-specific extensions.
- By having separate components in the platform, it is possible to select the relevant components from the platform, add your own components, and build a product. This would not be possible if the platform was not componentised.