Structuring Product-lines: A Layered Architectural Style

Tommi Myllymäki, Kai Koskimies, and Tommi Mikkonen

Institute of Software Systems, Tampere University of Technology
Box 553, FIN-33101 Tampere, Finland
{tommikm, kk, tjm}@cs.tut.fi

Abstract. A product-line is a set of software systems sharing the same architecture and implementation platform. Based on an analysis of several industrial product-lines, a general layered model is proposed as the main structuring principle of a product-line. It is demonstrated that the model can be used to explain and clarify several existing product-line architectures.

1 Introduction

The current focus in industrial software development is to find techniques that optimize the software development process within the domain of an enterprise. This has led to the proliferation of *software product-lines* ([1], [2]). A (software) product-line is a set of systems that share a common architecture and a set of reusable components, which jointly constitute a software platform supporting application development for a particular domain.

Although the idea of software product-lines has been widely adopted, the design of product-line architectures has proved to be hard. Typically, product-lines make use of a variety of existing techniques like architectural styles [3], components [4], frameworks [5], design patterns [6], and generative programming approaches [7]. The overwhelming options of existing technologies make it difficult for a software architect to come up with a well-structured high-level solution for a product-line architecture. In many cases, the platform is seen as an unstructured set of assets, some of which are then selected to be part of a product build. Then, it is not clearly understood how the platform should be structured, what kind of variability is supported by the different parts of the platform, what kind of specialization or usage interfaces are offered by each part, and how the different parts are versioned. Another indication of the lack of understanding of product-lines is that the terminology is often confusing, not clearly separating the constituent concepts of product-lines.

In this paper, our aim is to study the structuring principles of the software platforms enabling product-lines, propose a general model based on a layered architectural style for such platforms, and clarify the product-line terminology on the basis of that model. The work builds on case studies carried out in a research project investigating industrial product-line practices. The model is not new in the sense that it has been actually used in a number of product-lines, but we argue that it has not

been previously clearly recognized and analyzed. We hope that this model and the terminology it implies will help software architects to understand product-line organization and to communicate about product-lines.

We proceed as follows. In Section 2 we give an account of the proposed layered model, followed by real-life examples sketched in Section 3. Finally, some discussion and concluding remarks are presented in Section 4.

2 Layered platform: A product-line architectural style

A software asset (say, a component) may depend on architectural aspects at different levels of abstraction and generality. Firstly, a software asset may depend only on the form and semantics of some general resources common to a wide spectrum of products. Secondly, a software asset may depend on a particular, general architectural style. Thirdly, a software asset may depend on the design decisions related to a particular application domain. Finally, a software asset may depend on the design decisions required for a particular product. On the basis of these dependencies, we divide the assets of a software system into four categories, each constituting a layer. The layers are called the *resource platform*, the *architecture platform*, the *product platform*, and the *product* layer, respectively. The resulting layered architecture is depicted in Fig. 1.

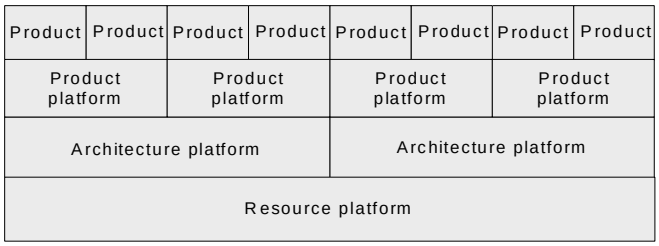

Fig. 1. Layered platform architecture

The layers are naturally ordered on the basis of generality: the categories were listed above in the order of decreasing level of generality. If a software asset depends only on the way general resources are provided but not on any particular architectural style, it belongs to the resource platform layer (i.e., the lowest layer in the figure). If a software asset depends on a chosen architectural style, but not on a particular domain, it belongs to the architecture platform layer. If a software asset depends on an application domain but not on a particular product, it belongs to the product platform layer. Finally, if a software asset depends on a particular product, it belongs to the product layer (uppermost layer). All the software artifacts belonging to any of these layers are collectively called a product-line.

The products built on a product-line can be divided into a system hierarchy based on the level of shared layers. Products that share the same product platform are called a *product family*. These products have a significant amount of common functionality,

and they share the same architecture. Products that share the same architecture platform are called a *product tribe*. These products have a common high-level architecture but not necessarily common functionalities. Products that share the same resource platform are called a *product culture*. Such products have neither common architecture nor common functionality, but they have a common set of resource abstractions.

2.1 Resource platform layer

The lowest layer, the resource platform, provides an API for accessing the basic services and resources of the environment. Depending on the system, these services might be related, for example, to communication (distributed systems), process scheduling (embedded systems), or persistence (business systems). The provided services do not depend on a predefined architectural style or domain of the products. The services are called through a simple API: the caller becomes dependent on the form and semantics of the service, but not on the internal structure of the resource platform. Sometimes the resource platform may possess a state, in which case the order of the service calls may be restricted. In that case the resource platform should inform the caller about incorrect usage (e.g. with exceptions), allowing the caller to respond accordingly.

The task of the resource platform is to attach the software system to its external resource environment (hardware, graphics, networking etc.), and to facilitate the changing of that environment. For the latter purpose, the layer should map the underlying resources and services of the environment to an abstract, environment-independent interface. The abstract API may be an industrial standard, and the platform itself may be available for a particular environment as a commercial product.

The design of the resource platform is based on identifying relevant horizontal domains like file management, database access, network protocols, threading facilities and inter-process communication mechanisms.

2.2 Architecture platform layer

The architecture platform defines the architectural style of the product-line. It provides specific component roles as implied by the style, to be played by various components of the product platforms, and points where these components can be attached. It also provides a set of conventions or design patterns that must be followed by those components. Within the object-oriented paradigm, a role can be represented by a base class (interface) in the architecture platform, to be subclassed (implemented) in the product platform. Hence an object-oriented architecture platform can be implemented as a framework. Such a framework can be called *architecture framework*, to distinguish it from the more conventional *application framework*. Note that the extension points of the architecture framework are domain-independent, pertaining only to the architectural style.

The interface provided by the architecture platform may also include API-like service interfaces. However, an essential difference with the resource platform is that

each caller is in some role with respect to the architecture, and that role may imply certain interaction patterns that must be followed, whereas the callers of the resource platform are in a uniform position with respect to the platform.

The design of the architecture platform is based on the quality requirements of the products and on the functional requirements related to the chosen architecture. For example, in a platform based on the implicit invocation architecture [3] the functional requirements specify the communication mechanisms of the components. The architecture platform may be developed by the same company that builds the products, or it may be a third-party component, possibly based on a standard interface.

2.3 Product platform layer

The product platform builds on the general architectural style provided by the architecture platform, and provides a skeleton for products in a particular product category (or domain). The skeleton may embody refined architectural solutions for this domain, within the boundaries of the underlying architectural style. The product platform provides a specialization interface that allows for variance in the functional requirements of the products. The design of the product platform is affected both by the common, domain-specific functional requirements and by the quality requirements of the products.

A possible form for the product platform is a framework together with supplementary components, each component being used in at least two different products. Such a framework is called an *application framework*.

A product platform is usually developed by the same company that develops the products: from the business viewpoint, a product platform allows the company to store its knowhow on building software for a particular domain, so that this knowhow can be easily reused for new products.

2.4 Product layer

The product layer implements product-specific functional requirements. The product layer is written against the specialization interface provided by the product platform. This interface may hide the underlying architectural style. It is often possible to generate the product-specific parts, or a large portion of them, automatically on the basis of a description of the desired functional properties of the product. The product layer is typically company-specific.

3 Examples

In this section, we will show how the discussed principles fit various systems we have studied as part of this research. The selection of systems introduced here covers different domains, which demonstrates the genericity of the approach.

3.1 Symbian OS

Symbian OS is a software platform targeted for mobile systems [8], in which the layers discussed above have the following contents. The resource platform of the Symbian architecture is constituted by the kernel. In the simplest form, it can be understood as a context-switching machine. In addition, as the platform is targeted to mobile systems, special measures are required against memory garbaging. The architecture platform layer in the Symbian environment provides an infrastructure that can be easily adapted to different kinds of mobile devices. The most important decision in the architectural sense is that hardware (and also many software) resources are predominantly protected with servers, and therefore, client-server sessions are built in the system. Moreover, the architecture platform defines the basic application structure. The programmer must comply with the defined practices, which is often enforced in a framework-like fashion. The product platform is populated by the core application logic of the applications built in the environment. Device specific user interface adaptations also reside in this layer. Applications in the Symbian environment are coded by attaching application specific code into a predefined MVC-like framework defined by the lower layers.

3.2 Insurance product-line

EJB (Enterprise JavaBeans) is a component architecture for distributed business applications provided by Sun Microsystems [9]. Currently several commercial and non-commercial products implement the EJB specification.

An EJB-based insurance system product-line supports the development of (distributed) insurance applications for various sub-domains of insurance business. In this case the resource platform consists of the services of the underlying technological infrastructure, most notably relational database and communication protocols (ORB, Object Request Broker). The architecture platform is represented by an EJB implementation, providing general services for the EJB component model. An application platform constitutes an application framework for life insurance systems, based on the EJB component model. Finally, a product is a particular life insurance application. Note that other insurance application frameworks can be built on top of the same architecture platform.

3.2 3G network elements

IP multimedia services [10] in 3G telecommunications network is supported by several inter-connected database like network elements. Each element performs different functions and handles different data, but essential quality goals like high availability and high performance remain the same. Development of these elements in Nokia Networks is organized roughly according to the presented layered model. In their case, the resource platform layer offers network element independent services like inter-process communication facilities and database access. These services are

not limited to supporting IP multimedia only, but can be used in the development of other kinds of network elements too. The architecture platform layer provides a very flexible architecture framework implementing design solutions ensuring high availability and high performance. The framework also implements key concepts that can be specialized significantly in order to enable differences in functionality. The product platform layer consists of complete implementation of each network element. The elements are then fine tuned for customer specific needs according to the configuration information contained in the product layer.

4 Discussion

We have witnessed the same structure in several industrial systems; therefore, we consider it an emerging pattern. The proposed layered model based on this structure is expected to facilitate organizing product-lines in several ways. Foremost, it can be used as a base to organize development of software assets shared between individual products and product families in an enterprise. It also alleviates estimating the amount of work required to introduce a new product into an existing product family or to build an entirely new product family. The model localizes the effects of changes in the requirements, allowing the reuse of large portions of the system even when the domain changes significantly. The introduction of the model also helps in communicating about product-lines, because a terminology is available for addressing assets at different levels of generality and different sets of products.

Ongoing application of the model in the industry will provide further confirmation of these expectations. Hopefully it also provides other interesting insights and reveals potential downsides of the model.

References

1. Bosch J., Design & Use of Software Architectures: Adopting and evolving a product-line approach. Addison.Wesley, 2000.
2. Clements P., Northrop L., Software Product Lines: Practices and Patterns. Addison.Wesley, 2002.
3. Shaw M., Garlan D., Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
4. Szyperski C., Component Software: Beyond Object-Oriented Programming. Addison-Wesley, 1998.
5. Fayad M., Schmidt D., Johnson R., (eds.), Building Application Frameworks — Object-Oriented Foundations of Framework Design. Wiley, 1999.
6. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
7. Czarnecki K., Eisenecker U., Generative Programming, Methods, Tools, and Applications. Addison.Wesley, 2000.
8. Tasker M., Allen J., Dixon J., Shackman M., Richardson T., Forrest J., Professional Symbian Programming: Mobile Solutions on the EPOC Platform. Wrox Press Inc, 2000.
9. Sun Microsystems: http://developer.java.sun.com/developer/products/j2ee/, 2002.
10. IP Multimedia (IM) Subsystem - Stage 2, 3GPP TS 23.228, http://www.3gpp.org, 2002