SERVCE-ORIENTED AND SOFTWARE PRODUCT LINES
ARCHITECTURE: A SYSTEMATIC COMPARISON OF TWO CONCEPTS

M. Chandra Pal. , A.Vinay Babu , A.Sadanandam

KeyWords
SOA, Component, Service, Application Engineering, Domain Engineering.

ABSTRACT

Service-Oriented Architectures (SOA) and Software Product Lines are two concepts that currently get a lot of attention in research and practice. Both promise to make possible the development of flexible, cost effective software systems and to support high levels of reuse. But at the same time they are quite different from one another: while Software Product Lines focus on one producer alone developing a set of systems based on a common platform (often in the embedded systems-domain), most proponents of SOA propose systems consisting of loosely coupled services or company-wide infrastructures including a variety of systems that are loosely coupled using services.

In any case, the services are usually developed by various companies. The focus of this paper is the systematic comparison of these concepts and an outlook on how Enterprise Component Platforms could be created by combining SOA and Software Product Lines.
A.1 INTRODUCTION

The focus of this paper is the systematic comparison of Software Product Lines and SOA. Specifically, the goal is to analyse both concepts with two questions in mind:
1) Can web services support product lines using a service-oriented architecture?
2) How can use of product line practices support web services and service-oriented architectures?

Therefore, we briefly describe Software Product Lines and SOA in Section A.2 before comparing them using defined criteria in Section A.3. Our conclusion in Section A.4 recapitulates the findings, linking them with the concept of Enterprise Component Platforms. Also, an outlook on further research that is necessary is given.

A.2 BRIEF PRESENTATION OF THE CONCEPTS

A.2.1 SOA

"SOA is conceptual business architecture where business functionality, or application logic, is made available to SOA users, or consumers, as shared, reusable services or an IT network. ‘Services’ in an SOA are modules of business or application functionality with exposed interfaces, and are invoked by messages" [1].

Service-oriented development essentially integrates disparate heterogeneous software services from arrange of providers [2]. Thus, an SOA is a means of designing software systems to provide services to either end user applications or other services through published and discoverable interfaces. There are several guiding principles that define the ground rules for development, maintenance, and usage of the SOA. The guiding principles cover
- Reuse, granularity, modularity, composability, componentization, and interoperability,
- Compliance to standards (both common and industry-specific),
- Service identification and categorization, provisioning and delivery, and monitoring and tracking. The following specific architectural principles for design and service definition focus on specific themes that influence the intrinsic behaviour of a system and the style of its design. They are derived from the guiding principles and cover [3]:
  • Service encapsulation - Accessing functionality through some well-defined interface, the application being seen as a black box to the user.
  • Service loose coupling - Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.
  • Service contract - Services adhere to a communications agreement, as defined collectively by one or more service description documents.
  • Service abstraction - Beyond what is described in the service contract, services hide logic from the outsideworld.
  • Service reusability - Logic is divided into services with the intention of promoting reuse.
  • Service composability - Collections of services can be coordinated and assembled to form composite services.
  • Service autonomy – Services have control over the logic they encapsulate.
  • Service statelessness – Services minimize retaining information specific to an activity.
  • Service discoverability –Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

While many early publications promote SOA as some kind of silver bullet for building flexible applications and for integrating different applications, newer publications point out the problems resulting from this architectural paradigm and Web Services as the most prominent way of implementing an SOA (e.g., [5], Chapter 4).

A.2.2 Software Product Lines

Exploiting commonalities between different systems is at the heart of Software Product Line Engineering. Therefore, different products of one domain (also referred to as problem space or application range, e.g., operating systems for mobile telephones or software support of the sales department) are viewed as a family and not as single products.

The main elements of a Software Product Line are the product line architecture and the individual products which are part of the product line. The product line architecture describes the individual products, their common components and the differences between the products of the family. These commonalities and differences are described using the core concept in Software Product Line Engineering: variability. Variability describes the variations in (functional as well as non-functional) features along the product line: features are either a commonality or a variation [8]. Different process models exist for the development process of product lines, e.g., those described in [9], [10] or [11]. Common to them is that the product line development process is modeled along the structure of a product line. Just as the product line consists of product line architecture and product line members, the development
process also consists of the process of the development of the product line architecture and the development process of product line members. The development of the product line architecture is called domain engineering and the development of the product line members is called application engineering. Preceding both is the activity called scoping, that is the process during which it is determined what to develop, i.e., which products will be part of the product line and what the commonalities and variabilities will be. Since both domain engineering and application engineering encompass analysis, design, implementation and testing, the resulting model is also called the two life-cycle model.

A.3 COMPARISON OF THE CONCEPTS

Having presented Software Product Lines and Service-Oriented Architecture, we will now compare these concepts and investigate the commonalities and differences between the concepts. To facilitate the comparison, we use the following criteria:

• Goal: What exactly is the concept trying to achieve?
• Defining features: What are the characteristics of the concept that are at its heart?
• Technical methods and elements: Which Software Engineering methods and elements are used to develop systems in this concept?
• Organizational methods and elements: How is software development organized according to this concept and which are the key steps in the development process?
• Field of application: In what kinds of software is this concept primarily applied?
• Reuse methods and entities: All three concepts have reuse in one way or another as their goal, but the methods and entities that are reused differ substantially.
• Level of Abstraction: Which is the primary unit of analysis for reuse? Not only methods and entities, even the level of abstraction differs significantly.

• Examples: To illustrate the concepts, some examples for real-world application of each concept are presented here.

The primary goal of Software Product Lines is to promote reuse and thereby realize gains in productivity, software quality and time to market. More specifically, exploiting the commonalities between related products is the actual goal. To achieve this, rather extensive analyzing and planning processes for the whole set of systems to be developed are performed. After that, the common architecture and the so-called core assets are developed in a generic way (domain engineering), before the systems belonging to the product line are developed (application engineering). Neither no architecture or core assets are planned to be reused outside the Software Product Line. SOA seeks to change this by developing rather small services (potentially totally independent from each other). These are published in a registry (e.g., using the Standards WSDL and UDDI) and can then be used by anyone within a company or even worldwide (the so-called service consumer). Software Product Lines are mainly focused on internal reuse of components in another product, while the focus of Service-Oriented Architecture is the reuse of component-based software on a larger scale. The creation of SOA-compliant component-based software (e.g., Modules or Components in Enterprise Resource Planning Software like SAP) seems to become a popular business model for companies, e.g., sub suppliers to SAP’s ERP system, that mainly focus on the creation of reusable component-based software but also for bigger companies, enabling them to sell SOA compliant component-based software that was developed in-house. Since this will probably lead to customers combining services from different suppliers, one could also argue that reuse will actually become less common: instead of a few large companies developing ERP systems and customers buying the whole package, many other companies can offer specialized services replacing the service included in the package. This does increase the choice for the customers, but not the level of software reuse.

The defining feature of the concept of Software Product Lines is variability (and vice versa commonality) as defined by the common and application-specific parts of the systems that are part of the Software Product Line; this includes defining a common architecture. This common architecture is lacking SOA; one could even say that the lack of a common is one of the defining features together with the services being encapsulated and loosely coupled. On the other hand, some of the aspects usually included in an architecture still have to be specified for services in order for them to be able to work together, e.g., messaging (cf. [5]).

The technical methods and elements that are typical for the concepts are additional criteria we used: for Software Product Lines, variation points and variation mechanisms and the distinction between scoping, domain engineering and application engineering are the defining technical methods and elements. While variation points and variation mechanisms provide the opportunity to efficiently handle the differences between the members of a product line, scoping, domain engineering and application engineering are distinct phases in the development process where special methods for Software Product Line Engineering are used (see for example [6] for details). Since SOA is a concept that is rather independent of the development platform/language to be used, the reliance on the architectural principles mentioned in Section A.2.1 need to be mentioned here. Additionally, standards such as UDDI and WSDL are important and absolutely necessary elements of SOA.

IJOAR® 2013
http://www.ijoar.org
Organizational methods and elements:

Unlike the technical methods and elements, the organizational methods and elements define the way software development is organized. For Software Product Lines, the key question here is how domain engineering and application engineering are organized: basically, they are separate development cycles with application engineering depending on the results of domain engineering. This could, for example, lead to separate teams responsible for domain and application engineering. Another possibility would include a separate team for domain engineering, with a member of this team being part of each application engineering team. For SOA, it is more difficult to make any statements concerning the organization since every service could be developed independently of all other services. But this implies a decentralized organization with no centralized coordinating unit, since there is no common architecture behind. For a company reorganizing their own infrastructure in an SOA-based way, there probably will be such a centralized unit, but they might very well use services that have been provided by third parties that were not coordinated by this unit. The reliance on additional services such as a service registry and services for identification or authentication implies separate centralized organizational units providing these services to all other services. The reuse methods and entities differ quite substantially: in a Software Product Line, all kinds of assets are reused, not only code, but also specifications, models (e.g., in UML), test cases and (end user) documentation, but only within the Software Product Line. In an SOA, the services are the main reuse entity, and interestingly, the services are physically and not only logically reused. Thereby, logical reuse is present, if a component is replicated and delivered by the manufacturer to the application developer. By physical reuse however, the service is invoked by remote call on demand [12]. In this case the service, e.g., a single-sign-on Web Service, is hosted by the manufacturer of the software. Closely related to the reuse entity is the level of abstraction: all considerations for a Software Product Line are based on the product line as a unit of analysis, all decisions on another level (product, component or even function) are derived from the utility on the product line level. As the name Service-Oriented Architecture already implies, single services are the main unit of analysis in this concept, since a service can theoretically stand alone.

A.4 CONCLUSION

The goal of this paper was the systematic comparison of Software Product Lines and Service-Oriented Architectures. The comparison shows that the two concepts share a number of characteristics, but differ significantly in other characteristics. And where they differ, they sometimes actually complement each other. Designing Software Product Lines based on a Service-Oriented Architecture with the possibility of replacing or extending existing functionality by services offered by third-party providers opens a path towards Enterprise Component Platforms that we find very promising. This leads to new research questions, e.g., on pricing of services and the platform, security and safety of the resulting systems, but also on business models for Enterprise Component Platforms.

REFERENCES


