Product Lines that supply other Product Lines: A Service-Oriented Approach

Salvador Trujillo
IKERLAN Research Centre
Mondragon, Spain
strujillo@ikerlan.es

Christian Kästner
University of Magdeburg
Magdeburg, Germany
ckaestne@ovgu.de

Sven Apel
University of Passau
Passau, Germany
apel@uni-passau.de

Abstract

A software product line *is a family of software products that share a set of core assets with the goal of reuse. In this paper, we focus on a scenario in which different products from different product lines are combined together in a third product line to yield more elaborate products, i.e., a product line consumes products from third product line suppliers. The issue is not how different products can be produced separately, but how they can be combined together. We propose a* service-oriented architecture*, in which product lines are regarded as services, yielding a* service-oriented product line*. This paper illustrates the approach with an example for a service-oriented architecture of a web portal product line supplied by portlet product lines.*

1. Introduction

The goal of a *software product line* is to produce a set of distinct but similar products. Typically, this is achieved by reusing a common product line infrastructure, which consists not only of traditional reusable software (e.g., code, models, documentation, etc), but contains product line specific assets as well (e.g., feature model, production plan, product line architecture, etc).

Currently, product lines in software development are targeted primarily at producing software products that are used in isolation. They can depend on third-party software (e.g., operating system, embedded system, or web container), but this third-party software is usually regarded as fixed because it is considered to be part of the execution environment. So, they do not depend on other software developed by third-party product lines.

Service-oriented architecture (SOA) is a novel paradigm that may change this scenario. Typically, an SOA application comprises a set of third-party services, which may be distributed. Each of such services supplies some specific functionality, and all together complete the distributed application functionality (i.e., the web services with fine-grained functionality are combined together to serve an application with coarse-grained functionality). SOA promotes services to be easily consumed by diverse applications because the discovery and consumption of services are standardized. The usefulness of SOA rests on existing standardization efforts and tooling [16].

Reusing services can be further ameliorated by creating a product line that satisfies diverse variability requirements from different customer applications (e.g., a product line of customized portlets for customer portals where existing techniques are used [10, 18]). This way, not only the application interface is customized by using standards to consume supplied services, but also the application functionality is customized by using product lines of supplied services.

However, the entire SOA application itself could require its customization (e.g., not only is each portlet customized from a product line, but the portal as well). When the SOA application itself turns into a product line, a new scenario emerges. This scenario requires that a product line consumes products that are supplied from third-party product lines. We call such scenario a *service-oriented product line* (SOPL).

Such situations are well-known in real industrial assembly lines. Consider a carmaker with an assembly line (e.g., from the chassis to the end-product) where third-party supplied components provided by other product lines are assembled together. These non-trivial components are the engine, the gear, the front-end, etc., which are also customized products of other product lines. In this case, there is a product line of cars that is supplied by other product lines of components.

Although this context seems futuristic for traditional software at first, it occurs for example when developing software for consumer electronics (e.g., several components like a TV receiver with different options are built into a TV product line) [19]. Here, *product populations* offer an architecture-centric approach to combine multiple product lines where human intervention is required [19]. Our work strives to homogenize the combination of products from product lines using the SOA paradigm. This reduces human intervention during product line discovery and minimizes human intervention from consumption. This way, the challenge is how to enable the automatic consumption of products from a third-party product line, which we address in this paper.

2. Service-Oriented Product Lines

There is nothing new on how multiple, distributed and heterogeneous product lines are developed in isolation, i.e. existing techniques can be used to create an individual SOPL. It is even possible for a product line to manually supply a product to a product line (e.g., when two products from two product lines are manually combined together). We envisage (semi-)automatic combination of multiple product lines.

The issue on how a product is plugged into a more complex end-product is addressed by *product populations*, which describe an architecture-centric approach to attain this coupling [19] (see Section 5). This approach requires human intervention.

We envisage the composition of software products supplied from different product lines with only little human intervention. To achieve this, several issues must be addressed. We have to *(i)* describe a supplier product line, *(ii)* sketch how to consume products supplied by other product lines, *(iii)* establish the product line operation, where performance, production schedule, bill of materials and other elements should be considered beforehand, and *(iv)* provide adequate tool support.

2.1. Supplier

First, we need to analyze which information a supplier product line should publish in order to enable its automatic consumption afterwards. A supplier is characterized by *(i)* descriptive information, *(ii)* product information, and *(iii)* a production interface.

- Descriptive information contains a unique identifier, a name, and a brief description of the product line. It is used later during the discovery and registration of the supplier product line.
- Product information describes how products are distinguished in a product line setting. A product is characterized usually by its features. This is the basic specification we need to build a product. Further information about core assets should be offered as well for descriptive purposes.
- A production interface provides information such as the production time, delivery time, average product cost, average product LOC, average product size, etc. An important piece of information is related to the interface for consumption (e.g., which URL should be invoked in the case of a web service and which parameters used). This information is useful when choosing among similar product lines.

Starting from this information provided by a supplier, a consumer might consume such supplier product line.

2.2. Consumer

A consumer product line demands products from third-party product lines. This demand is specified in terms of the suppliers' characteristics (e.g. descriptive information, etc). The purpose of a consumer product line is to effectively enable the access to a supplier. Each consumer product line is realized by a consumer *stub*, which links with its corresponding product line supplier. In SOA terms, a supplier is supplying services, and the consumer aggregates services to offer an application.

Nonetheless, our aim is not only to consume a single supplier, but to consume multiple supplier product lines. This can be achieved by combining a set of consumer product lines together. So, a set of consumer stubs can be used simultaneously.

This combination of consumers exposes an entire SOPL architecture representing all the product line suppliers involved. We envisage SOPLs for automating the operation of the entire product line.

2.3. Operation

We define a sequence of operations between the consumer and their suppliers in order to enable their communication. This is roughly divided into registration and consumption (see Figure 1).

Registration. The registration requires the discovery of each product line supplier (i.e., human intervention is required)[1]. Figure 1 shows how a consumer can register to an individual supplier product line where *PL_A* registers to *PL_1*. The sequence of operations involves first a *getServiceDescription()* call. Then, a *register()* operation establishes a relationship for future consumptions in which the supplier provides average production time, delivery time, etc. The general case would encompass registrations with several suppliers.

[1]The *UDDI* standard (for web-services) can be used in this context (http://www.uddi.org/).

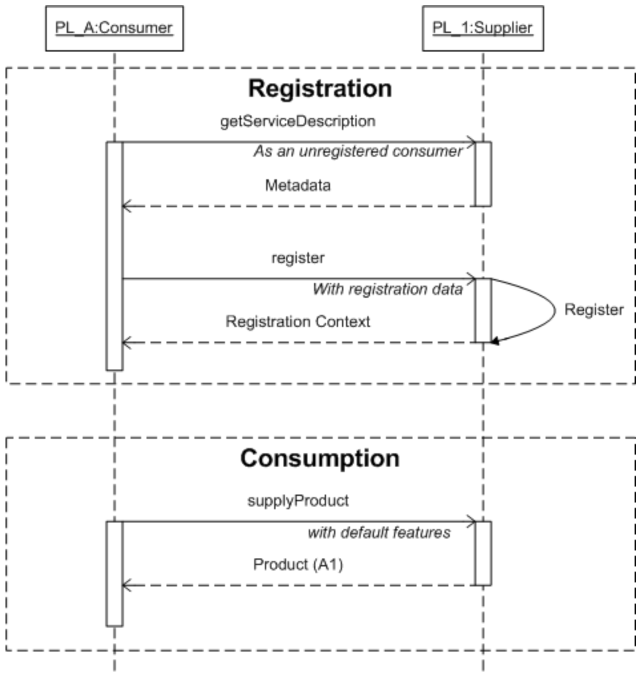

Figure 1. Operation - Sequence Diagram

Consumption. The consumption refers to the production and delivery of a product. In general, when the product line production or derivation process is automated, we can invoke such product line specifying desired features and receive a product [2, 6].

Figure 1 shows the sequence of operations where a *supplyProduct()* calls for the production and delivery of a specific *Product* (e.g., *A1* from *PL_1* in Figure 3). The supplied product is considered as a reusable asset by the product line consumer (*PL_A*). Nonetheless, tool support is needed to automate such consumption.

2.4. Tool Support

The ideas presented here benefit from SOA ideas. More to the point, existing SOA standardization efforts and tool support may readily enable the creation of such infrastructure.

In general, we envisage two kinds of consumptions. First, when the consumer and all supplier product lines are in the same workspace (same vendors), which is named *internal* consumption. Second, when the product lines are in distinct workspaces (distinct vendors), which is named *external* consumption. So far, we created initial tool support for the internal consumption (not detailed), and are planning to work on external.

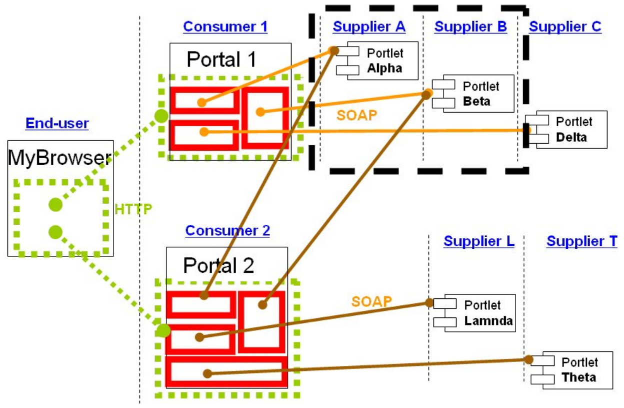

Figure 2. Portal / Portlet Architecture

3. An Example

Portals and Portlets. We choose *portals of portlets* to illustrate the idea of SOPL [10]. A *portal* is a web page that provides centralized access to a variety of services [8]. An increasing number of these services are not offered by the portal itself, but by a third-party component called a portlet, which is a presentation oriented web service [12, 15].

Figure 2 depicts a 3-tier architecture for portlets, where *MyBrowser* accesses the *Portal_1* page through *HTTP*. *Portal_1* is hosted by *Consumer1* and consists of a layout aggregating the *Alpha*, *Beta*, and *Delta* portlets that are hosted by different producers (a.k.a. suppliers).

When a family of similar portals (e.g., research group sites) is required, a customized portal can be the outcome of a product line that consumes portlets that are supplied by third-party product lines. Figure 2 shows this where *Portal_2* consists of a version of *Alpha* different from that used by *Portal_1* (same holds for *Beta*), and other portlets (e.g., *Lamnda*, *Theta*). This setting is commonplace in SOA.

Scenario. As a specific example, consider an SOPL on a product line of enterprise web portals, in which different services are offered. Each company requires a similar, but different version of the portal. So, there is a family of products. The services in each instance of the portal are offered by portlets (e.g., meeting room reservation, calendar, hotel reservation, flight reservation, etc), which as well vary and hence come from a product-line.

Figure 3 sketches our motivating scenario for a set of product lines of portlets, which supply to a product line of portals. *PL_A* is the product line of enterprise portals. This product line uses several portlets (from *A1* to *A6*). Note that some of them (*A1* and *A4*) are directly supplied by third-party product lines. *A1* is a meeting-room reservation portlet supplied from *PL_1* product line while *A4* comes from

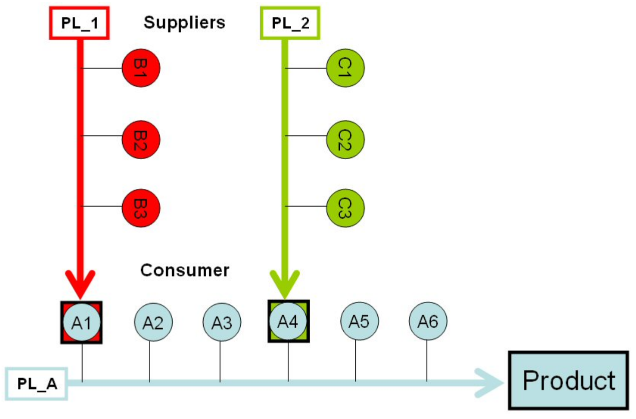

Figure 3. SOPL Scenario

PL_2 product line, which offers flight reservation functionality. *A1* and *A4* are actually portlets that are integrated into the entire portal[2].

A mechanism is needed for each product line supplier to receive the product configuration as input (e.g., selection of product features [2, 7]), and manufacture as output the final product[3].

The challenge of SOPL is to consume products that are supplied by *PL_1* and *PL_2* as composable artifacts in the *PL_A* product line (i.e., invoking third-party product lines and obtaining the product as a reusable asset for another product line). We believe that existing SOA tool support provides an adequate foundation for SOPL.

4. Discussion

Consistency. Products to be reused within a consumer product line need to fit precisely. Hence, the consistency is crucial to assure that the product, e.g., the configured portlet, fits as an artifact of a larger product, e.g., the portal. This consistency issue appears when features from a consumer require propagation to a supplier (e.g., the features from supplier should be consistent with the product features where it is to be aggregated[4]). It is not trivial how to do so as different names could designate same functionality and vice versa. Similarly, when dealing with heterogeneous product lines (e.g., products implemented in different platforms) consistency issues may appear as well.

Production. Production does not only depend on product line artifacts, but also depends on third-party artifacts. If these artifacts are not available within schedule, the product would not be produced. Hence, production schedule and timing should be carefully planned. Otherwise, undesirable production bottlenecks would appear in the performance.

Orchestration. Consistency and timing issues are symptomatic of a more general issue, which is orchestration (i.e., how different product lines are smoothly orchestrated together). This way, consistency, timing and production issues could be catered for. To attain this, experience from "real-world" manufacturing seems beneficial for production experiences. *Business Process Execution Language* (BPEL) is a case in point[5].

Service-Oriented Refactoring. The idea of SOPL to yield a product is backed by a non-trivial SOA scenario. However, the use of multiple product lines is not restricted to this case. Consider an individual product line, which has grown in time into a large product line. When this occurs, both technical and organizational management of the product line becomes highly intricate (e.g., core assets management, production planning, etc). There is an ancient principle to face this: "*divide and conquer*" (a.k.a. *separation of concerns* in software engineering). Applying such a principle leads to divide an original product line into a set of product lines. This refactoring of an original product line into a set of product lines would eventually enable to ease product line management (as they are smaller). This refactoring is also motivated when the newly created product line is to supply products to new customers that demand only restricted functionality (i.e., fewer than original product line functionality). Therefore, we envisage that several situations would demand service-oriented refactoring.

5. Related Work

As industrialization of the automobile manufacturing process led to increased productivity and higher quality at lower costs, industrialization of the software development process is leading to the same advantages [11]. A software factory[6] is defined as "*a facility that assembles (not codes) software applications to conform to a specification following a strict methodology*". In general, to set-up a factory is to create a production capability. An important piece of

[2]This does not preclude that the portlet is physically deployed on the same machine as the portal, but can be deployed externally, and reused solely by this specific portal.

[3]Such manufacturing (e.g., portlet product lines *PL_1* and *PL_2*) involves to *(i)* compose target product code, *(ii)* compile the resulting composition, *(iii)* create a Portlet bundle, and *(iv)* deploy it to a given location.

[4]This refers to a feature model, whose terminal features are replaced with an entire feature model [1, 7].

[5]BPEL is a business process language that grew out of WSFL and XLANG. It is serialized in XML and aims to enable programming in the large. The concepts of programming in the large and programming in the small distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes (from http://en.wikipedia.org/wiki/BPEL).

[6]http://en.wikipedia.org/wiki/Software_factory

work is how such factories connect with third-party factories.

In a product line setting, a factory uses a production plan, which is "*a description of how core assets are to be used to develop a product in a product line*" [4]. A production plan describes how a product is developed [3, 4, 14]. Lee et al. describe an approach for production planning based on features [13]. Recently, Wang et al. describe production "*on the fly*" where dynamic reconfiguration was used to support privacy in web applications [21].

Consider a typical production plan that implies the selection of product desired features in order to compose such selected features [9]. Then, when the raw compound product is obtained, it is necessary to create a binary (e.g., an executable, a JAR or a WAR). To this end, the raw is compiled, packaged, and deployed. Optionally, it may be measured, tested, versioned or even have documentation created. In general, it describes how the factory operates the reusable artifacts [17]. Such production techniques reuse not only the artifacts, but even the process that are present in the product line infrastructure. However, they do not enable invocation of a third-party product line and reuse the third-party product.

The notion of product populations is not far from SOPL. The difference stems from the fact that product populations focus on how a product is integrated into a product line (i.e., the architectural interfaces that glue them together) [19, 20]. This work focuses on the automation of this combination rather than on how products are glued together.

Product line products are usually produced reusing a common infrastructure. This infrastructure is usually internal to the product line. Even though there is experience with COTS components [5], they are not part of a product line. Hence, to the best of our knowledge, we are unaware of tooling to enable the automated consumption of a product line supplier.

6. Conclusions

This paper presented our ongoing work on the vision of SOPL, which consumes products from supplier product lines. We motivated our idea with an example for a product line of portals consuming supplier product lines of portlets. We introduced preliminary representations for consumer and supplier product lines, described the basic operation of registration and consumption, and sketched the initial tool support required.

SOPLs rely on SOA for product line production. To answer the workshop question, existing SOA techniques are used to build more complex SPL systems. Our longstanding aim is to facilitate the emergence of a concurrent market where atomic products from supplier product lines can be automatically integrated into a product line.

Acknowledgments. We thank Maider Azanza, Don Batory, Rafael Capilla, Oscar Diaz and Jon Iturrioz for their helpful comments on earlier drafts of this paper.

References

[1] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated Analysis of Feature Models: Challenges Ahead. *Comm. of the ACM - Special Issue on Software Product Lines*, Dec 2006.

[2] D. Batory, J.Neal Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, June 2004.

[3] G. Chastek, P. Donohoe, and J.D. McGregor. Product Line Production Planning for the Home Integration System Example. Technical report, CMU/SEI, September 2002. CMU/SEI-2002-TN-029.

[4] G. Chastek and J.D. McGregor. Guidelines for Developing a Product Line Production Plan. Technical report, CMU/SEI, June 2002. CMU/SEI-2002-TR-06.

[5] P. Clements and L.M. Northrop. *Software Product Lines - Practices and Patterns*. Addison-Wesley, 2001.

[6] K. Czarnecki and U. Eisenecker. *Generative Programming*. Addison-Wesley, 2000.

[7] K. Czarnecki and K. Pietroszek. Verifying Feature-Based Model Templates Against Well-Formed OCL Constraints. In *5th International Conference on Generative Programming and Component Engineering (GPCE 2006), Portland, Oregon, USA, Oct 24-27*, 2006.

[8] O. Díaz and J.J. Rodríguez. Portlets as Web Components: an Introduction. *J. UCS*, 10(4):454–472, 2004.

[9] O. Diaz, S. Trujillo, and F. I. Anfurrutia. Supporting Production Strategies as Refinements of the Production Process. In *Software Product Lines, 9th International Conference (SPLC), Rennes, France, September 26-29*, pages 210–221, 2005.

[10] O. Díaz, S. Trujillo, and S. Perez. Turning Portlets into Services: Introducing the Organization Profile. In *16th International World Wide Web Conference (WWW), Banff, Canada, May 8-12*, November 2007.

[11] J. Greenfield et Al. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.

[12] JCP. JSR 168 Portlet Specification Version 1.0, September 2003. at http://www.jcp.org/en/jsr/detail?id=168.

[13] J. Lee, K. Kang, and S. Kim. A feature-based approach to product line production planning. In *SPLC*, 2004.

[14] J.D. McGregor. Product Production. *Journal Object Technology*, 3(10):89–98, November/December 2004.

[15] OASIS. Web Service for Remote Portals (WSRP) Version 1.0, 2003. http://www.oasis-open.org/commitees/wsrp/.

[16] T. Erl. *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services*. Prentice Hall, 2004.

[17] S. Trujillo, M. Azanza, and O. Diaz. Generative Metaprogramming. In *6th International Conference on Generative Programming and Component Engineering (GPCE), Salzburg, Austria*, 2007.

[18] S. Trujillo, D. Batory, and O. Díaz. Feature Oriented Model Driven Development: A Case Study for Portlets. In *29th International Conference on Software Engineering (ICSE), Minneapolis, Minnesota, USA, May 20-26*, 2007.

[19] R. van Ommering. Building Product Populations with Software Components. In *24th International Conference on Software Engineering (ICSE), Orlando, Florida (USA)*, 2002.

[20] R. C. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala Component Model for Consumer Electronics Software. *IEEE Computer*, 33(3):78–85, 2000.

[21] Y. Wang, A. Kobsa, A. van der Hoek, and J. White. PLA-based Runtime Dynamism in Support of Privacy-Enhanced Web Personalization. In *SPLC*, 2006.