Software Engineering Institute

Proceedings of the First Workshop on Service-Oriented Architectures and Software Product Lines

Sholom Cohen
Robert Krut

May 2008

SPECIAL REPORT WITH UNLIMITED DISTRIBUTION
CMU/SEI-2008-SR-006

Product Line Practice Initiative
Unlimited distribution subject to the copyright.

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2008 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

This report contains the proceedings of the First Workshop on Service-Oriented Architectures and Product Lines (SOAPL) 2007 that was held on September 10th, 2007 in Kyoto, Japan as part of the 2007 Software Product Line Conference (SPLC 2007). This report includes an overview of the workshop, four invited presentations, details of the workshop's outcomes, and the workshop position papers.

1 Introduction

Service-oriented architecture (SOA) and software product line (SPL) approaches to software development share a common goal. They both encourage an organization to reuse existing assets and capabilities rather than repeatedly redeveloping them for new systems. These approaches enable organizations to capitalize on reuse to achieve desired benefits such as productivity gains, decreased development costs, improved time to market, higher reliability, and competitive advantage. Their distinct goals may be stated as

- SOA: "enable assembly, orchestration and maintenance of enterprise solutions to quickly react to changing business requirements" [1]
- SPL: systematically capture and exploit commonality among a set of related systems while managing variations for specific customers or market segments

The First Workshop on Service-Oriented Architectures and Product Lines (SOAPL) 2007 explored the connections from two perspectives:

1. Can services support product lines using a service-oriented architecture?
2. How can use of product line practices support services and service-oriented architectures?

1.1 ABOUT THIS REPORT

This report captures the information presented and discussed during SOAPL 2007. Section 2 outlines the workshop organization and format, Section 3 summarizes the presentations, Section 4 presents additional discussion topics, and Section 5 presents workshop outcomes. Appendices A through E contain the accepted workshop papers, which appear as they did upon acceptance except for minor editorial and formatting changes.

[1] Wienands, Christoph. "Studying the Common Problems with Service-Oriented Architecture and Software Product Lines." *Service-Oriented Architecture (SOA) & Web Services Conference*. Atlanta, GA, October 2006.

2 Workshop Organization and Format

2.1 WORKSHOP ORGANIZATION

Sections 2.1.1 through 2.1.3 below list the people who organized, facilitated, and participated in SOAPL 2007.

2.1.1 Organizers

- Sholom Cohen, Carnegie Mellon® Software Engineering Institute (SEI), USA, sgc@sei.cmu.edu
- Paul Clements, SEI, USA, clements@sei.cmu.edu
- Andreas Helferich, Universität Stuttgart, Germany, helferich@wi.uni-stuttgart.de
- Robert Krut, SEI, USA, rk@sei.cmu.edu
- Grace Lewis, SEI, USA, glewis@sei.cmu.edu
- Dennis Smith, SEI, USA, dbs@sei.cmu.edu
- Christoph Wienands, Siemens Corporate Research, USA, christoph.wienands@siemens.com

2.1.2 Facilitator

- Robert Krut, Software Engineering Institute, USA, rk@sei.cmu.edu

2.1.3 Participants

- David Benavides, University of Seville, Spain, benavides@tdg.lsi.us.es
- Masayoshi Hagiwara, Microsoft, Japan, masayh@microsoft.com
- Andreas Helferich, Universität Stuttgart, Germany, helferich@wi.uni-stuttgart.de
- Jean-Narc Jezequel, University of Rennes, INRIA, France, Jean-Narc.Jezequel@inria.fr
- Larry Jones, Software Engineering Institute, USA, lgj@sei.cmu.edu
- Christian Kästner, University of Magdeburg, Germany, ckaestne@uni-magdeburg.de
- Dan Lee, ICU, Korea, danlee@icu.ac.kr
- Jaejoon Lee, Lancaster University, U.K., j.lee@comp.lancs.ac.uk (Fraunhofer Institute for Experimental Software Engineering (IESE) in Frankfurt at the time of his participation)
- Tomi Männistö, Helsinki University of Technology, Finland, tomi.mannisto@tkk.fi
- Shuhei Nojiri, Hitachi, Japan, shuhei.nojri.dd@hitachi.com
- Mikko Raatikainen, Helsinki University of Technology, Finland, mikko.raatikainen@tkk.fi
- Ktar Sato, DENSO, Japan, ktar@bof.jp

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

2.2 WORKSHOP FORMAT

The workshop format was highly interactive and focused on making tangible progress towards answering the two questions relating to the connections between SOA and product lines. The accepted papers provided the key issues relating to the workshop theme: “Service-Oriented Architectures and Product Lines - What Is the Connection?” The breakdown of the papers into topical areas helped us set up topics for discussion at the workshop. The paper topics broke down into three areas:

1. methods for SOA and product line development
2. managing service features and variability
3. examples of applications

The morning session featured presentations based on position papers (Section 3). At least one paper was presented for each topic area. Presentations were limited to 15 minutes followed by discussion.

The afternoon session provided an opportunity for the group to continue discussing the identified topic areas or to identify new topics based on the dynamics and interests of the group. The group identified six topics to discuss for the afternoon session (see Section 4).

3 Workshop Papers and Presentations

3.1 PAPERS

The workshop organizers accepted the following five papers, each of which appears in an appendix of this report.

3. Appendix A: Software Product Lines and Service-Oriented Architecture: A Systematic Comparison of Two Concepts
4. Appendix B: A Taxonomy of Variability in Web Service Flows
5. Appendix C: Comparison of Service and Software Product Family Modeling
6. Appendix D: Identifying and Specifying Reusable Services of Service Centric Systems Through Product Line Technology
7. Appendix E: Product Lines that Supply Other Product Lines: A Service-Oriented Approach

In addition to the papers, the organizers accepted one website as a contribution, which addresses the relationship between SOA and SPL: *A Framework for Software Product Line Practice, Version 5.0, FAQ* (http://www.sei.cmu.edu/productlines/frame_report/FAQ.htm#other_approaches) [SEI 2007a].

3.2 PRESENTATIONS

Four papers were presented during the workshop morning session and are described below in Sections 3.2.1 through 3.2.3. Each presentation is listed by the topic area identified by the workshop organizers. A brief overview of the presentation is included as well as questions submitted by the workshop organizers prior to the presentations.

The complete presentations are provided on the SOAPL 2007 website (http://www.sei.cmu.edu/productlines/SOAPL/) [SEI 2007b].

3.2.1 Methods for SOA and Product Line Development

Mikko Raatikainen made the presentation for the first topic area. Mikko is from the Helsinki University of Technology, Finland**,** and co-authored the paper he presented: *Comparison of Service and Software Product Family Modeling*.

The presentation began with a brief discussion of the similarities and differences between software product families[2] (SPFs) and service-oriented computing. They are similar in that they both involve developing applications from existing software and a reliance on modeling. They differ in that service-oriented computing involves dynamic computational elements whereas SPFs typically comprise static elements (i.e., dynamic binding versus static).

[2] *Software product families* were equated to *software product lines* as defined by Clements [Clements 2001].

The main body of the presentation examined and compared the modeling methods of SPFs and service-oriented computing. SPF family modeling focuses on domain models which include variability models and product models. SPF modeling employs many approaches such as Feature-Oriented Domain Analysis and extensions to existing approaches such as UML. Service-oriented computing modeling focuses on modeling approaches for web services, since web services are currently the dominant implementation of service-oriented computing. Modeling of web services is typically driven by standards such as Web Service Definition Language (WSDL) and Business Process Execution Language (BPEL). The notation is usually Extensible Markup Language (XML).

The noted comparisons between the modeling methods are

- Services involve no domain or variability modeling while SPFs do.
- Services tend to be compositional while SPFs tend to be top down.
- Both focus on architectural entities. However, SPFs typically focus on static entities whereas service-oriented computing models typically focus on dynamic entities.
- SPFs are much broader in focus at the architectural level while modeling in service-oriented computing tends to focus on the behavior of the system.
- Service-oriented computing models employ an XML notation, while SPF modeling typically uses a graphical notation.

The presentation concluded with suggestions of future directions for combining the two modeling approaches:

- The feasibility of variability modeling for service-oriented computing should be studied.
- Variability modeling in SPFs should be extended to include lessons learned from behavior modeling and analysis of services and business processes.
- The necessary approach for modeling of services and SPFs should be studied more thoroughly.

The workshop organizers submitted the following questions prior to the presentation.

Questions: "Could criteria from the SEI Service Migration and Reuse Technique (SMART) serve as an approach for the migration of legacy components for product lines? What specific criteria would apply here? Are there detailed examples or a comparison of models (e.g, feature models versus SDL/BPEL/Business Process modeling notation (BPMN)?"

Response: The authors were not familiar with specific examples of SMART's application to legacy systems. They were also not aware of any detailed examples or a comparison of models.

The presenter pointed out that SPFs were intra-organizational whereas the use of services is external. The presenter reiterated that there was a relative tendency for a static focus in SPFs versus a dynamic focus for service-oriented computing. His team had tried to apply its SPF modeling tools (KumbangTools) to service composition with some success [KumbangTools 2008]. They were not suitable for complex behavior.

The presenter felt that future efforts should focus on

- the creation of standards for SPLs similar to those being worked on with services
- the working implementation of SPF modeling tools in service-oriented computing
- more interplay between research and practice

3.2.2 Managing Service Features and Variability

David Benavides made the presentation for the second topic area. David is from the University of Seville, Seville, Spain**,** and co-authored the paper he presented: *A Taxonomy of Variability in Web Service Flows.*

This presentation provided a brief discussion of how SPL practices can be used to support service-oriented applications. Since the most common implementation of service-oriented applications is web services, this presentation primarily focused on how to manage variability in web services, in the context of SPL and SOA, by defining a Web Service Flow (WS-flow) and identifying variability points in WS-flows. This research provides a starting point for a base of knowledge about variability in WS-flows. It can be used further for evaluating the different mechanisms for implementing variability in WS-flows and identifying factors that affect the selection of such variability mechanisms.

A WS-flow is a composite web service that is implemented through use of a process-based approach. A WS-flow specifies a set of tasks that are executed by the participants of a process and defines the execution order of tasks, the data exchange among the participants, and the business rules. The language used to define WS-flows is BPEL.

In this body of work, identification of variability points in WS-flows was limited to service invocation and the process workflow structure. The presenter defined a *service invocation* as "an activity in which workflow invokes another service and exchanges messages with it returning control back to the workflow." The process workflow structure "determines all the aspects related to the way in which the process is executed: the execution order, the data exchanged between participants, the business rules, the errors treatment, etc."

The presentation provided a feature model summarizing the variability points in the invocation of services, as shown in Figure 1.

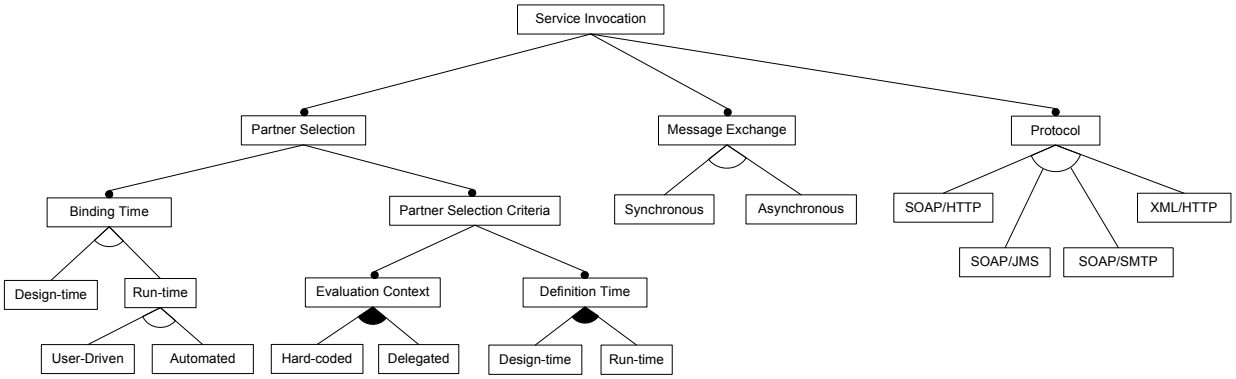

Figure 1: Variability Points in Service Invocation

The four main variability points identified were: 1) binding time, 2) partner selection criteria, 3) message exchange, and 4) protocols. Binding time offers the selection of services to be invoked at design time or runtime where runtime is further divided into user driven and automated. Partner selection criteria helps to determine which of the available services offering the same functionality will be selected for invocation. Evaluation context enables the selection criteria to be hard coded or delegated. Definition time enables the selection criteria to be modified at design time or runtime. Messages exchanged between service workflows and services may be synchronous or asynchronous. Four different protocols may be used for service interactions over the network.

The two main variability points in process workflow structure are control flow and data flow. *Control flow* is the workflow structure that determines the tasks to be executed and the execution order. *Data flow* covers the exchange of data between services.

The presentation concluded with the reiteration that there is a need for a classification of variability points in WS-flow to serve as a starting point for handling variability through services in the context of SPLs and SOA. Future work will look at implementation technologies—paying particular attention to the ways in which they support the variability points presented—leading to a service-based development of business-driven SPLs.

The workshop organizers submitted the following questions prior to the presentation.

Questions: "Where an application in an SOA-based product line is built using services from external core asset sources, how would product development manage variability and selection of variation of features within those assets? Could entire services be substituted? Are there variations within a service? Is there any implementation of the taxonomy?"

Response: In their work, the authors don't have an implementation yet, so they are not sure how product development would manage variability and selection of variation of features within those assets or how to automate the feature model. Their research currently examines the relationship between SPLs and SOA.

3.2.3 Application Examples

Two presentations were made for the third topic area. Jaejoon Lee made the first presentation. Jaejoon worked at the Fraunhofer Institute for Experimental Software Engineering (IESE) at the

time of this presentation and now can be contacted at Lancaster University in the U.K. He co-authored the paper he presented: *Identifying and Specifying Reusable Services of Service Centric Systems Through Product Line Technology*.

This presentation provided a brief discussion on the challenges of dynamically managing services in an SOA-based system and how product line engineering concepts were used to identify and specify reusable services based on features. The approach to identify or specify reusable services of an SOA-based system is presented in Figure 2.

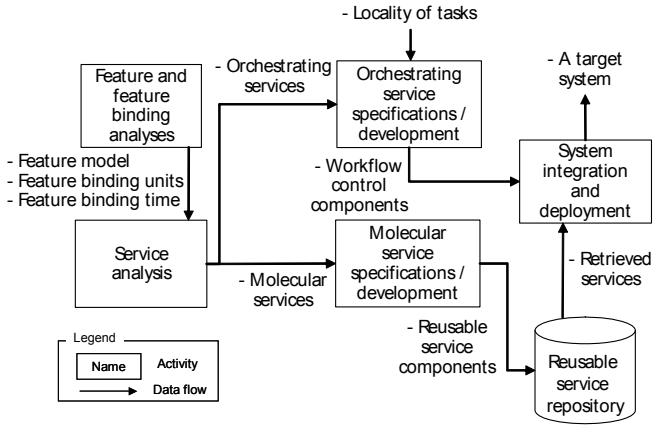

Figure 2: Activities for Managing Services in an SOA-Based System

The feature and feature binding analysis organize the system features into a product line features model that includes identified binding units (representing major functionality of a system) and relative binding times.[3] The service analysis examines the feature model and feature binding information to identify molecular services (computational-oriented services that represent a predefined task) and orchestrating services (behavior-oriented services that define a sequence of tasks). The molecular services are the basic building blocks, reused as-is by the orchestrating services. Molecular services are self-contained and stateless, have pre-/post-conditions, and represent domain-specific services.[4,5] The orchestrating service represents a workflow for dependable orchestration of molecular services. Each workflow is based on a service behavior specification with pre-/post-conditions and invariants.

To illustrate this approach, the presenter introduced the application domain "Virtual Office of the Future." The virtual office provides workers with tools, technology, and skills to perform tasks at any time, from any location. The presenter walked through diagrams for molecular service identification, workflow specification, and identification of tasks from a workflow specification for the virtual office.

[3] Grouping of features into binding units of the same binding times is a key driver for identifying reusable services.

[4] *Domain-specific* is a key property in identifying the correct level of granularity of a service.

[5] *Quality of service* is defined in the features of the molecular service.

The workshop organizers submitted the following question prior to the presentation.

Question: How would identified services be used in applications? Might we see hybrid service-/component- oriented applications? What evidence is there of an actual "right" scale of granularity? Do case study artifacts beyond the limited figures in the paper actually exist?

Response: The authors are planning to prototype the virtual office. They are not sure about a hybrid service/component-oriented application. They need to study the molecular service as having the right level of granularity.

Christian Kästner made the second presentation in the third topic area. Christian is from the University of Magdeburg, Magdeburg, Germany, and co-authored the paper he presented: *Product Lines that Supply Other Product Lines: A Service-Oriented Approach.*

This presentation provided a service-oriented approach to combining different products from different product lines into a third product line, yielding more elaborate products. The approach uses an SOA in which product lines are regarded as services that are consumed by service-oriented product lines (SOPLs).

The concept of a SOPL is illustrated through a "web portals of portlets" example. A *portal* is defined as an "application that provides centralized access to a variety of services." *Portlets* are components (services) offered by a third party. The scenario requires that a product line consumes products that are supplied from third-party product lines.

In this example, there exists a two-fold connection between product lines and SOA. First, there exists a product line of portals that enables customer portals to be developed from customized portlets. The application functionality is customized by using product lines of supplied services, and the application interface is customized by using SOA standards to consume supplied services. Second, the portals may be customized creating a product line of portals. Therefore, not only is the portlet customized from a product line, but the portal is as well.

How can a software product line automatically request and consume a product from another product line? The vision for the SOPL is the integration of products supplied from different product lines with minimal "human intervention." Currently, *manual integration* is the means of combining different products from different product lines. By using SOA, product developers can "homogenize" the products from product lines. Therefore, the SOPL can be used to automate the operation of a software product line by automatically requesting and consuming products from another product line.

The SOPL relies on a supplier/consumer relationship and operations. A *supplier* is defined as a product line that supplies products to other product lines. It is characterized by descriptive information, product information (including feature and core asset information), and product interface. A *consumer* is a product line that consumes products from supplier product lines. Operations involve registration (i.e., the discovery of each product line supplier) and consumption (i.e., production and delivery of a product) based on the existing SOA standardization efforts and tool support.

The web portals of portlets example illustrated the idea of SOPL. However, more work must be done to create the infrastructure to make this a viable approach with models, tools, and so forth.

Since this presentation did not have a question submitted by the workshop organizers prior to the presentations, the authors elected to answer the workshop theme: “Service-Oriented Architectures and Product Lines - What Is the Connection?”

Question: Service-Oriented architectures and product lines - what is the connection?

Response: The authors believe that SOA techniques can be used as an infrastructure on which to build increasingly complex software product line systems. Their vision is to facilitate the emergence of a concurrent market where atomic products from supplier product lines can be automatically integrated into a larger product line.

4 Additional Discussion Topics

When the presenters were finished, the group discussed topics that arose in response to their presentations. The discussions followed the dynamics and interests of the group by identifying the following five topics:

1. What are the possible SOA-PL connections?
2. What are the issues surrounding dynamic aspects of both SOA and PLs?
3. What is a reusable service?
4. What are the architectural aspects of SPLs versus SOA?
5. What is the scope of a system in the context of services?

4.1 WHAT ARE THE POSSIBLE SOA-PL CONNECTIONS?

This discussion focused on two topics:

1. including services within a product line architecture
2. developing a service as a product line

To include services in a product line, developers could include a variation point in the architecture implemented as a component or as a service. A specific configuration could select the component or the service, depending on the specific functional or quality features needed by the application and satisfied by each alternate. Services in this context could address possible selection features such as

- a need for dynamic variation
- exploitation of the availability of existing services where appropriate
- use of Universal Description Discovery and Integration (UDDI) to transfer information during execution for service selection
- rapid construction of product line systems

A second connection could be designing services as a product line. In this context, services themselves would be configurable according to architecture variations or specific features. Possibly a service product line could be offered in a marketplace, where an organization acquires the service outright for in-house tailoring or commissions the SOA product line developer to tailor the product line for the organization's use. For example, the SOA product line may be a mortgage service product line. A bank or other lending institution could acquire access to a specific instance, defining the specializations it needs to the SOA product line developer. Alternatively, the entire product line capability could be acquired, and the bank or lending institution could tailor the service in multiple ways dependent on customer categories, local banking regulations, or other variations.

Many of the organizational issues encountered in introducing SOA or SPLs are similar. While some involve technical aspects—architecture, testing, integration—the highest risk areas tend to be organizational. The need to justify investment, train developers, and operate a product

line/SOA development organization involves many of the same practice areas. A sharing of case studies based on real-world examples could support integrating product line solutions and SOA solutions. For example, both SOA and product lines currently suffer from limitations on reuse outside the immediate development organization. Investigation of successful uses of a product line or SOA across an enterprise and even between enterprises could support the SOA and product line connection.

4.2 DYNAMIC ASPECTS—WHAT ARE THE ISSUES?

Much of this discussion focused on the advantages of SOA in supporting dynamic execution. The position of many in the discussion is that SOA executes dynamically, by definition, while component technology is static. However, a product line architecture may also support a dynamic variation mechanism via plug-ins or some other plug-and-play architecture. Dynamic class loading in Java, for example, allows selection of classes when needed, based on product and user context at the time of class selection. Dynamic link libraries and reflection offer runtime selection for variation. The SOA and product line connection can benefit from the sharing of experience results in this area.

The group also proposed that a performance penalty comes with dynamic selection and that development is more difficult. However, dynamics can also reduce complexity. The group discussed printing services as an example. An application may support dynamic determination of a printer, based on printing needs and existing conditions such as queue length or printer condition. If I have a long file to print, the application may determine the efficiency of waiting for a faster printer with a longer queue than immediate printing where there is no wait. The printer example may be overly simplistic, since the application involved is by definition stateless—the application "doesn't care" what print services have been previously executed. Other services may perform differently based on prior execution, where caching or other runtime service states may affect quality of service.

In a pure SOA or mixed product line/SOA, other dynamic issues emerge. These include detection of available or unavailable services and responses to these conditions. Is the protocol to retry or to immediately fall back to an alternative service? What if no alternate is available or identified? Also, can an existing application dynamically integrate services with new, unforeseen functionality?

Testing and reliability in a dynamic environment also affect validation. A tested service operates within some known bounds, but dynamic selection may pose a context outside the tested bounds. Does the service continue to perform within its "guaranteed performance parameters?" How does a potential service user confirm or at least measure this situation? Third-party services in general lead to uncertainty for the user. A service should publish its assumed pre- and post-conditions for validation of services, so the user can determine, dynamically, if its current context satisfies these conditions. If the current context does not, the potential service user looks elsewhere.

4.3 WHAT IS A REUSABLE SERVICE?

The paper and presentation by Jaejoon Lee, *Identifying and Specifying Reusable Services of Service Centric Systems Through Product Line Technology*, makes a distinction between molecular, or fine-grained, components and behavior or orchestrating services that manage the workflow of

molecular tasks. This structure provides a two-tier scope—a lower tier of molecular, task-oriented services that are intended for widespread as-is reuse and orchestrating services that must be tailored for reuse. Orchestrating services satisfy a defined scope much as a product line restricts scope. Scoping of service applications addresses some of the design risk of unbounded reuse. Inherent in product lines is restriction of atomic services.

Klaus Turowski of the University of Augsburg, along with others in the German information systems community, has identified seven levels for specifying components within an information systems application [Fellner 2000]. These components range in complexity from blocks of code, modules, classes, objects, macro/templates, abstracts, data types, to component. The framework distinguishes among these by reuse (e.g., platform dependency or inter-component dependency), interface standards, interoperability, extent of deployment, marketability, and other factors. This work has been extended to cover real-time service selection in component-based architectures [Skroch 2007]. Adding services to the classification framework, and possibly components and services of different granularities, could also support a service-to-component core asset comparison.

The group discussed the perceived differences between reusable services and components in a product line. Components generally operate within a context defined by the architecture. The component interface defines that context, and any component user must satisfy the terms of use. Services, especially those that Jaejoon Lee's paper refers to as molecular, make no assumptions about context of use. While reusable services are intended for use in different contexts, a component could similarly be built without any assumptions regarding context. A research area could be established in order to determine the additional context information or assumptions that must be stored and/or communicated. One proposed solution is an information broker that makes services available through user registration with the broker. This approach could manage context between a service or component and its respective users.

4.4 WHAT ARE THE ARCHITECTURAL ASPECTS OF SPLs VERSUS SOA?

The discussion contrasted differences in architecture practices between those used with product lines and those used with SOA. While both share the need to define architecture context, structure, and compositional rules, many of the participants perceive a significant contrast between SOA and product line architecture practices as summarized in the following table.

Table 1: Differences Between Architecture Practices for SOA and Product Lines

SOA Architecture Practices	Product Line Architecture Practices
Architecture characterized as autonomous, decentralized	Architecture characterized as centralized, static
Business processes examined and modeled	Architectures concentrate on views and viewpoints for architecture descriptions
Rules easily changed	Compositional rules predefined
Variability only within services or possibly within processes	Variability within structure and components
Architecture defined by platform (e.g., enterprise service bus)	Architecture defines platform
Role of SOA unclear with respect to quality attributes	Architecture guarantees quality attributes

A final aspect of the discussion contrasted the perceived "simplicity" of SOA systems. Integrating independent services, the SOA protocols (web-based or others) and underlying platform may address many architectural issues that are open design problems for a product line architecture. The SOA developer in this view is the service developer/provider of interfaces with concerns separate from those of the integration environment using services through published interfaces.

4.5 WHAT IS THE SCOPE OF A SYSTEM IN THE CONTEXT OF SERVICES?

This part of the discussion focused on the meaning of a service product line. Suppose services are offered as static services for others. The offered services describe the scope of a product line, defined by the functions offered, and vary according to nonfunctional, quality attributes such as security, memory/processor performance, or availability. Selection among services may occur at runtime based on quality attributes offered by services performing the same function.

Many organizations control large numbers of services to support internal processes. Services may be shared across groups within the organization, with designated partners, or on the open market. The granularity of use within a product line of services may be at the level of just a single service inside one product line—basically one feature where SOA is not a factor—or entire applications may be fashioned by utilizing services from across the service product line. Service orientation in the latter context becomes a variability mechanism.

The service product line could itself be used across multiple product lines. Services might not even be bound within a domain of a particular product line. Microsoft offers the Workflow Foundation to rapidly build activity-based applications. These are generally service oriented and may in turn use BizTalk services to perform a variety of identity and connection management operations. Other examples might exist and enrich the understanding of product line and scope of service applicability.

5 Workshop Outcomes

The First Workshop on Service-Oriented Architectures and Product Lines (SOAPL) 2007 made progress towards answering the two questions relating to the connections between SOA and product lines:

1. Can services support product lines using an SOA?
2. How can use of product line practices support services and SOAs?

The accepted papers, located in Appendices A-E, provided a basis for identifying key issues relating to the workshop theme "Service-Oriented Architectures and Product Lines - What Is the Connection?" Along with the workshop presentations described in Section 3, the papers helped establish topics for additional discussion at the workshop, as described in Section 4.

A look at the comparison of software product line and service-oriented modeling methods identified key issues in

- the perceived static focus of software product lines versus the dynamic focus of service-oriented computing
- variability modeling in services
- the creation of standards for software product lines similar to those being developed for services

Participants addressed features and variability issues by examining how software product line practices might support the management of variability in service-oriented applications. For example, could feature modeling be used to identify the variability points in the invocation of composite web services? Could this initial knowledge base be used to evaluate the different mechanisms for implementing variability in composite web services and to identify factors that affect the selection of such variability mechanisms?

Application examples addressed the challenges of dynamically managing services in an SOA-based system:

- how product line engineering concepts can be used to identify and specify reusable services—based on features
- how a service-oriented approach can be used to combine different products from different product lines into a third product line

Additional discussions covered the use of services within a product line architecture, developing a service as a product line, the dynamic aspects of SOA versus product lines, reusable services, the perceived differences between reusable services and components in a product line, architectural aspects of software product lines versus service-oriented computing, and the meaning of a service product line.

The participants felt the goals of the workshop were addressed. The workshop offered an early glimpse at how the SPL community looks at SOA. It examined tools and techniques currently in progress and generated a list of open questions for future research directions. Most importantly, the workshop provided the ability to network with others working on the same issues.

Several participants discussed follow-on work that should be monitored. Mikko Raatikainen plans to work on "how to build configurable services" (i.e., understanding issues, the modeling of behaviors, tool support, and dynamic aspects). Jaejoon Lee will continue implementation of the model and complete the current work described in the paper he presented. He will also start exploring platform issues (such as .net). Christian Kästner plans to implement the web portal of portlets example and look at how to dynamically consume configured products from the product line.

The workshop participants felt that this workshop should be followed up with a second SOAPL workshop at the 12th International Software Product Line Conference (SPLC 2008), Limerick, Ireland. Suggested changes for this workshop are

- Include SOA representations. The participants at this workshop primarily represented software product lines. The second workshop should include experts in SOA to balance the discussions.
- Include a keynote speaker to open up the workshop.
- Invite and include more experience reports. The workshop papers should focus on experience reports rather than research or simply the connection between service-oriented architectures and product lines.
- Invite a product line architect to address dynamic versus static issues.

References

URLs are valid as of the publication date of this document.

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

[Fellner 2000]
Fellner, Klement J. & Turowski, Klaus. "Classification Framework for Business Components." *Proceedings of the 33rd Hawaii International Conference on System Sciences.* Waikoloa, HI, January, 2000. IEEE, 2000. http://ieeexplore.ieee.org/iel5/6709/20043/00927009.pdf

[KumbangTools 2008]
KumbangTools. http://www.soberit.hut.fi/KumbangTools (2008).

[SEI 2007a]
Software Engineering Institute. *A Framework for Software Product Line Practice, Version 5.0,* 2007. http://www.sei.cmu.edu/productlines/framework.html
Glossary. http://www.sei.cmu.edu/productlines/frame_report/glossary.htm
Training. http://www.sei.cmu.edu/productlines/frame_report/training.htm
Using Software Product Lines with Other Approaches. http://www.sei.cmu.edu/productlines/frame_report/FAQ.htm#other_approaches

[SEI 2007b]
Software Engineering Institute. *11th International Software Product Line Conference.* http://www.sei.cmu.edu/productlines/SOAPL (2007).

[Skroch 2007]
Skroch, Oliver & Turowski, Klaus. "Improving Service Selection in Component-Based Architectures with Optimal Stopping," 39-46. *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007).* Lubeck, Germany, August, 2007. IEEE Computer Society, 2007.

Appendix A: Software Product Lines and Service-Oriented Architecture: A Systematic Comparison of Two Concepts

Andreas Helferich, Georg Herzwurm and Stefan Jesse
Universität Stuttgart, Chair of Information Systems (Business Software),
Breitscheidstr. 2c, 70174 Stuttgart, Germany
{helferich,herzwurm,jesse}@wi.uni-stuttgart.de

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 analyze 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 a conceptual business architecture where business functionality, or application logic, is made available to SOA users, or consumers, as shared, reusable services on 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 a range 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 [3]:

- 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 behavior 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 outside world.
- 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. According to the SEI at Carnegie Mellon University, Software Product Lines are defined as “set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” (cf. [6], p. 5). 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 (cf. [7]). 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.

Table A-1 provides an overview of the comparison using these criteria, whereas the in-depth comparison follows in the remainder of this section.

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 architecture nor core assets are planned to be reused outside the Software Product Line. The primary goal behind SOA is to promote flexibility in information systems/corporate information systems landscapes: today large enterprise application packages and tight coupling between different packages and legacy systems are prevalent, leading to problems whenever a new system is introduced or business needs require changes in existing systems and/or their interfaces. SOA seeks to change this by developing rather small services

Table A-1: Comparison of the Concepts

Criteria	Software Product Lines	Service-Oriented Architecture
Goal	Planned exploitation of commonalities within related systems -> reuse	Use of services of fine granularity within (enterprise) system landscapes -> flexibility
Defining features	Variability; Family of related systems based on common architecture	No common architecture, services are encapsulated and loosely coupled
Technical methods and elements	Variation points and mechanisms, scoping, application engineering, domain engineering	Reliance on generally accepted standards, additional service registration and authentication services
Organizational methods and elements	Two life-cycle models: first domain engineering to develop the assets to be reused, then application engineering to derive the actual systems	Development as well as hosting of the services can be distributed, only the light-weight interface and some additional services (registry, authentication…) are provided
Reuse methods and entities	Logical reuse of all kinds of assets (components, test cases, analysis & design models), but only within the product line	Services are physically reused, potentially by anyone, and can be combined with other services into more complex services
Level of abstraction	Primarily family of systems and secondarily systems within the family	Single services (atomic or composed of services)
Examples	Nokia cell phones, Cummins diesel engines	Telecommunications provider

(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 world-wide (the so-called service consumer). As Dietzsch [12] points out, this kind of reuse is physical rather than logical: the same entity provides the service, not a copy of the entity (a reused component is a copy of the original component used in another piece of software, the service is reused by sending a request to the very same service over the network/Internet). Such a service can be part of a system, stand alone or be a connector between two independent systems. Additionally, a service can be atomic or combine several services (composition of services).

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 architecture (since the service could be used by anyone as part of his/her system with its specific architecture) 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.

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 an in-depth discussion of possible ways to organize Software Product Line Engineering see [13], but basically all possibilities have their own advantages and disadvantages and their suitability depends on the organization of the company as a whole. 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.

Taking organizational methods and elements on the one hand and the reuse methods and elements on the other hand, one gets the matrix shown in Table A-2.

Table A-2: Organizational Level of Reuse

Phase within the two-lifecycle model	Software Product Lines	Service-Oriented Architecture
Development for reuse	within organization	within organization / outside the organization
Development with reuse	within organization	outside the organization

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.

Cummins diesel engines and Nokia cell phones are just two **examples for the application** taken from the Software Product Line Hall of Fame [14]. One example of using SOA in order to streamline business processes and to integrate various applications is presented in [15], where a "large telecommunication wholesaler, supplying its services to more than 150 different service retailers, enhanced the process integration capabilities of its core order management system through wide-spread use of SOA, business process choreography and Web services concepts" [15].

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, e.g., while Software Product Lines do not focus on components being marketable or developed in different organizations, this is not explicitly excluded. At the same time, many proponents of SOA argue that SOA will lead to companies not purchasing licenses for large application packages but instead using services and paying per use of the services, thereby combining best-of-breed services from multiple providers. 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. The large business software companies, i.e., SAP, Oracle, IBM and Microsoft have already invested large amounts of money and effort into the transition of their application packages into services, while trying to maintain control over the resulting platform and trying to create a network of partners supporting the platform. SAP for example uses the term business ecosystem (cf. for example [16] and [17] for SAP's

strategy or [18] and [19] for a more theoretical viewpoint).

REFERENCES

[1] Marks, A. and Bell, M. *Service-Oriented Architecture*: *A Planning and Implementation Guide for Business and Technology*, John Wiley & Sons, New Jersey (2006).

[2] Cerami, E. *Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*, O'Reilly, Beijing et al., 2002.

[3] Balzer, Y. *Improve Your SOA Project Plans*, IBM, 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-improvesoa/

[4] Erl, T. *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, Upper Saddle River, New Jersey, 2005.

[5] Kaye, D. *Loosely Coupled: The Missing Pieces of Web Services*, RDS Press, Marin County, CA, 2003.

[6] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, Boston, MA, 2002.

[7] Bosch, J. *Design and Use of Software Architectures*, Addison-Wesley, Harlow, 2000.

[8] Kang, K., Cohen, S., Hess, J., Novak, W., & Peterson, S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* (CMU/SEI-90-TR-021). Software Engineering Institute, Pittsburgh, PA (1990).

[9] Bayer, J. et al. "PuLSE: A Methodology to Develop Software Product Lines." *Proceedings of the 5th Symposium on Software Reusability*, 1999, pp. 122-131.

[10] Weiss, D. M., and Lai, C. T. R. *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, Reading, MA, 1999.

[11] Muthig, D. *A Light-Weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*. PhD Thesis, Fraunhofer-IRB Verlag, Stuttgart, 2002.

[12] Dietzsch, A. *Systematische Wiederverwendung in der Software-Entwicklung*, PhD thesis, Deutscher Universitäts-Verlag, Wiesbaden, 2002.

[13] Pohl, K., Böckle, G., & van der Linden, F. *Software Product Line Engineering*, Springer, Heidelberg, 2005.

[14] Software Engineering Institute, Carnegie Mellon University: "Product Line Hall of Fame." http://www.sei.cmu.edu/productlines/plp_hof.html, June 11, 2007.

[15] Zimmermann, O., Doubrovski, V., Grundler, J., Hogg, K. "Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned." OOPSLA'05, October 16–20, 2005, San Diego, CA, pp. 301-312.

[16] Oswald, G. *SAP Service and Support*, Galileo Press, New York, Bonn, 3rd edition, 2006.

[17] Karch, S. and Heilig, L. *SAP NetWeaver Roadmap*, Galileo Press, New York, NY, 2005.

[18] Gawer, A. & Cusumano, M. A. *Platform Leadership: How Intel, Microsoft and Cisco Drive Industry Innovation*. Harvard Business School Pr., Boston, MA, 2002.

[19] Baldwin, C. Y. and Clark, K. B. *Design Rules - The Power of Modularity.* MIT Press, Cambridge, MA, 2000.

Appendix B: A Taxonomy of Variability in Web Service Flows[1]

Sergio Segura, David Benavides, Antonio Ruiz-Cortés and Pablo Trinidad
Department of Computer Languages and Systems, University of Seville
email: {segura, benavides, aruiz, trinidad}@tdg.lsi.us.es

ABSTRACT

The combination of Software Product Lines (SPLs) and Service-Oriented Architectures (SOAs) development practices is expected to become a new development paradigm maximizing reuse and business integration. However, multiple issues must be still addressed in order to clarify the connections between both fields. One of the key questions to answer is how SPL practices can be used to support service-oriented applications. In this context, identifying and managing the points of variability in composite Web services emerges as an inevitable step for making possible such integration. In this position paper we give a first step toward such direction by introducing a comprehensible overview of the main variability points in Web service flows.

B.1 INTRODUCTION

Software Product Lines (SPLs) [8] and *Service-Oriented Architectures (SOAs)* [18] approaches to software development pursue different goals from a common perspective: software reuse. On the one hand, SPLs focus on managing commonalities and variabilities among a set of related software systems. On the other hand, SOAs enable assembly, orchestration and maintenance of service-based solutions implementing business processes.

Contributions about the connections between both development approaches, SPLs and SOAs, are starting to emerge in the SPL community [22]. However, multiple issues must be still addressed for studying how SPL practices could support the development of service-oriented systems. In this context, a relevant issue to be analyzed is managing variations for specific customers or market segments in SOA.

Service-oriented applications are not tied to a specific technology. However, the most common implementations of SOA-based systems use Web services as a suitable integration technology. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network using Web standards protocols [2]. The main goal is to achieve interoperability among applications in a language and platform independent manner. However, the real strength of Web services is obtained when combining them and orchestrating them in order to deliver value-added services. In this context, Web Service Flows (WS-flows) are a common way of implementing composite Web services in SOA. WS-flows are composite Web services implemented using a process-based approach. Roughly speaking, a WS-flow process defines an executable business process in which participants are Web services.

[1] This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web–Factories (TIN2006-00472).

Research in the field of variability in conventional Web services [12, 16, 19] and process workflow [7, 10, 11, 15, 20] is merely addressed in the literature. In [13] a high level classification of approaches to WS-flow adaptability is presented. A more technological classification of WS-flow variability points in service invocation is introduced by IBM staff in [9]. However, an explicit classification of the main variability points in WS-flow is still missed.

In this paper we give a first step toward a proposal for managing variability in WS-flow in the context of SPLs and SOAs. In particular, we first introduce WS-flow and BPEL. Secondly, we describe and classify the main variability points in WS-flow. The goal is to provide the starting point for a base of knowledge about variability in WS-flows that can be later used for both: 1) evaluating the different mechanisms for implementing variability in WS-flow and 2) identifying factors that affect the selection of such variability mechanisms.

The remainder of this paper is organized as follows: In Section B.2 WS-flows and BPEL are introduced. The main variability points identified in WS-flows are described in Section B.3. Finally, we summarize our main conclusions and describe our future work in Section B.4.

B.2 WEB SERVICE FLOWS

A *Web Service Flow (WS-flow)* is a composite Web service implemented using a process-based approach [13]. Similar to conventional process workflow, WS-flows specify a set of tasks which are executed by the participants of a process. Additionally, a WS-flow defines the execution order of tasks, the data exchange among the participants and the business rules. In contrast with traditional workflows, the main characteristic of a WS-flow is that it works mainly with a single type of participant: Web services. Figure B-1 depicts an example of a WS-flow of a travel agency for travel arrangement. The WS-flow invokes the Web services of different airlines, car rental companies, and hotels offering to the customer a value-added service for travel reservation.

There exist multiple proposed languages for defining WS-flows such as WSCI [21], BPML [4] or BPEL [14]. However, the *Business Process Execution Language (BPEL)* is recognized as *de facto* standard in this area. BPEL introduces basic and structured activities, control structures such as loops and conditional branches, synchronous and asynchronous communication, etc. Although BPEL processes are defined in XML format, most development IDEs provide a graphical notation for it. Once a BPEL process is defined it can be executed in any BPEL-compliant execution engine such as active-BPEL [1]. The execution engine orchestrates the invocations to the participant's Web services according to the process definition.

B.3 VARIABILITY IN WS-FLOWS

In this section we explore the main variability points in WS-flow. In particular, we focus on the variability in the invocation of services and the workflow structure. Variability in other advanced aspects of services such as security is out of the scope of this paper because of space constraints.

B.3.1 Service invocation

A service invocation is an activity in which the workflow invokes another service and exchange messages with it returning control back to the workflow. Figure B-2 summarizes the main variability points identified in the invocation of services using a feature model. In particular, we have identified

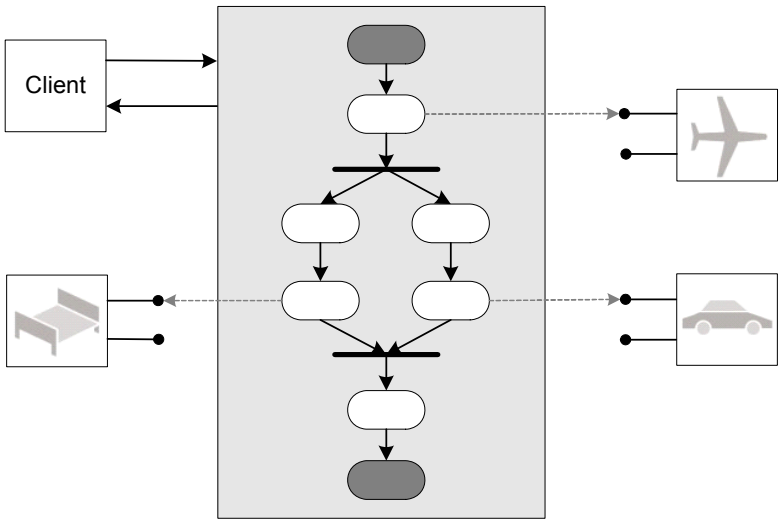

Figure B-1: A Possible WS-Flow for Travel Arrangement

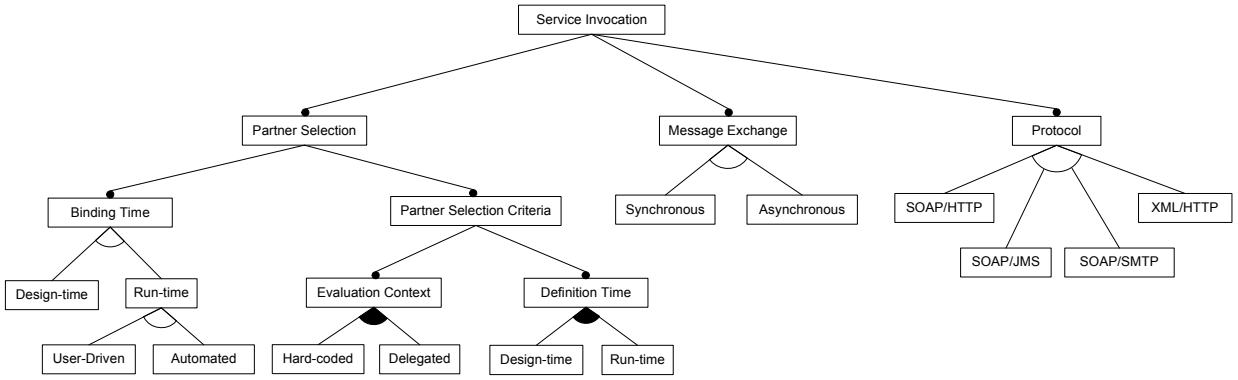

Figure B-2: Variability Points in Service Invocation

four main variability points:

- **Binding Time**. The selection of the service to invoke can be performed either during the development or the execution of the workflow. In the first case, the service reference is defined in design-time forcing to redeploy the workflow if changes in the participants need to be done. On the other hand, most flexible approaches propose selecting participants in run-time making the application adaptable to changes in the execution environment. Additionally, partner selection during run-time can be performed either by the user or automatically according to some selection policies. Figure B-3 shows a possible implementation of run-time automated partner selection using a so-called service registry [3]. First, the information of the services (e.g., different airline Web services) is registered in a service registry. Then, the workflow sends a query to the registry to determine a matching service according to a set of parameters (e.g., a service with time of response ≤ 10s) and the prede-

fined selection policies. Finally, the service reference obtained as a result of the query is used to invoke the matching service.

- **Partner Selection Criteria**. Selection criteria help to determine which of the available services offering the same functionality will be selected for its invocation [17]. In this context, two main variability points are identified:

  Evaluation Context. Selection criteria can be either hard-coded in the workflow or delegated to an external entity. The first option is very limited since workflow and selection criteria are highly coupled. On the other hand, defining the selection criteria in an independent manner is a preferred approach since it allows managing changes more efficiently. Figure B-4 depicts an example in which the selection criteria are defined out of the scope of the workflow. Notice that changes in the selection criteria would be welcome since they would not affect the workflow.

  Definition Time. Selection criteria can be modified either in design-time or run-time. Similar to the partner selection, the first option forces the workflow to redeploy to respond to changes. Meanwhile, the second alternative is much more flexible since it allows adapting the process workflow dynamically.

- **Messages Exchanged**. Messages exchanged between executable service workflows and other services are typically performed using two different communication patterns: synchronous or asynchronous. Synchronous request/response message exchange consists of sending a request message to the service and waiting for it to respond. Although this is the most common and natural approach, it is clearly not feasible if the services require significant time to respond since it blocks the workflow processing. Hence, when the participants' services can take a long time to respond and such response is not needed for workflow processing, an asynchronous pattern is typically used.

  In the asynchronous model the communication is performed between two workflows, the so-called service provider and service requestor or client. In this situation, the client need not block the call. Instead, the client implements a callback interface, and once the results are available, the service provider simply makes a callback invocation on the client. Figure B-5 illustrates an example of asynchronous message exchange.

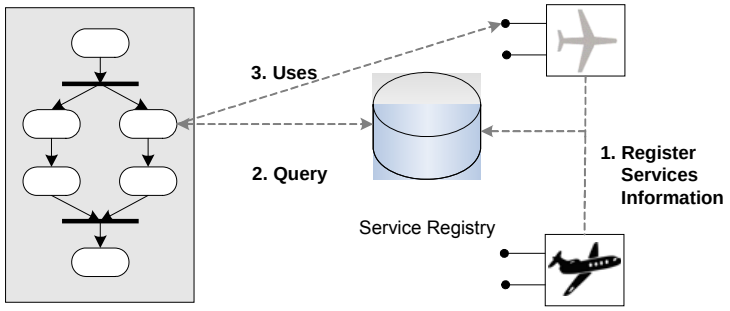

Figure B-3: Service Registry

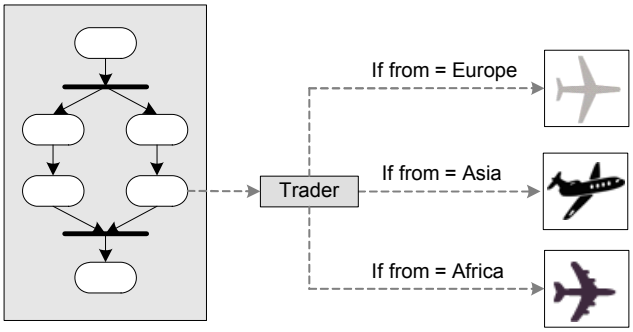

Figure B-4: Workflow-Independent Selection Criteria

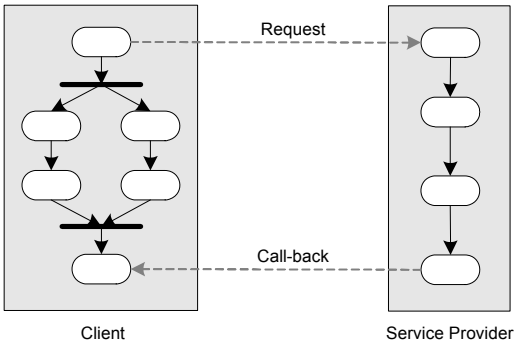

Figure B-5: Asynchronous Model

- **Protocols**. Multiple protocols can be used for service interactions over a network, i.e., SOAP/HTTP, SOAP/JMS, XML/HTTP, etc. Thus, the selection of a suitable set of protocols for the communication with services is a key variability point.

B.3.2 Process Workflow Structure

The process workflow structure determines all the aspects related to the way in which the process is executed: the execution order, the data exchange between participants, the business rules, the errors treatment, etc. Hence, two main variability points are identified in this context:

- **Control Flow**. The workflow structure determines the tasks to be executed, the execution order, and even the participant in the process. Therefore, the control flow will commonly have locations likely to change in response to changes in the business process. Hence, for instance, suppose the travel agency decides to change the order in which flight fares are consulted for certain customers, e.g., prioritizing low-cost airlines for young people.
- **Data Flow**. During the execution of a WS-flow, participants exchange different kinds of data in XML format. Similar to the control flow, data is likely to change as a consequence of implement-

ing changes in the business process. As an example, suppose the travel agency is asked to provide additional security information in the cases in which passengers travel to a specific country.

B.4 CONCLUSIONS AND FUTURE WORK

In this paper we expose the need for an explicit classification of variability points in WS-flow as a starting point for handling variability through services in the context of SPLs and SOAs. In particular, we identify and classify the main variability points in the invocation of services and the workflow structure. In some cases the distinction between development-time and run-time is exposed explicitly because of its relevance. However, we emphasize that the time in which variability is resolved will depend mainly on the technology used.

Many challenges remain for our future work. Once the main variability points are identified, it will be necessary to consider the available technological approaches for implementation. Hence, we are already evaluating the different implementation proposals and are paying special attention to the way in which they support the variability points presented in this paper.

Finally, our main goal is to develop a prototype development tool for the generation of a SPL of composite Web services. Although our work is still immature, we plan to develop a framework for the automated or semi-automated generation of BPEL code from a given extended feature model [6]. The framework will implement a core business process in which variable parts will be generated automatically according to the feature selection. For such purposes, we will start by associating features and feature attributes to Web services and Quality-of-Service (QoS) parameters respectively [5].

REFERENCES

[1] ActiveBPEL. www.activebpel.org/.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[3] D. Ardagna and B. Pernici. “Adaptive Service Composition in Flexible Processes.” *IEEE Transactions on Software Engineering*, 33(6): 369–384, 2007.

[4] A. Arkin. “Business Process Modeling Language (BPML).” Version 1.0, 2002. http://www.bpmi.org/.

[5] D. Benavides, A. Durán, M. A. Serrano, and C. Montes-Oca. “Quality of Service Variability in System Families Based on Web Services.” In *Simposio de Informática y Telecomunicaciones SIT 2002*, pages 205–218, Sevilla, Spain, 2002.

[6] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520: 491–503, 2005.

[7] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. “Adaptive and Dynamic Service Composition in eFlow.” In *Conference on Advanced Information Systems Engineering*, volume 1789, pages 13–31. Springer Verlag, 2000.

[8] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.

[9] G. Goldszmidt and C. Osipov. “Make Composite Business Services Adaptable with Points of Variability. Choosing the Right Implementation.” IBM, April 2007. http://www.ibm.com/developerworks/library/ar-cbspov1/.

[10] Y. Han, A. Sheth, and Chr. Bussler. “A Taxonomy of Adaptive Workflow Management.” In *CSCW-98 Workshop, Towards Adaptive Workflow Systems*, 1998.

[11] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. “A Comprehensive Approach to Flexibility in Workflow Management Systems.” *SIGSOFT Soft. Eng. Notes*, 24(2): 79–88, 1999.

[12] A. Ruokonen, J. Jiang, & T. Systa. “Pattern-Based Variability Management in Web Service Development.” IEEE, November 2005.

[13] D. Karastoyanova and A. Buchmann. “Extending Web Service Flow Models to Provide for Adaptability.” *Proceedings of the OOPSLA ’04 Workshop on Best Practices and Methodologies in Service-oriented Architectures: Paving the Way to Web Services Success*, Vancouver, Canada, October 2004.

[14] OASIS. “Web Services Business Process Execution Language Version 2.0,” May 2006. http://www.oasis open.org/.

[15] M. Reichert and P. Dadam. “ADEPT Flex-Supporting Dynamic Changes of Workflows Without Losing Control.” *Journal of Intelligent Information Systems*, 10(2): 93–129, 1998.

[16] S. Robak and B. Franczyk. “Modeling Web Services Variability with Feature Diagrams.” In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, pages 120–128, London, UK, 2003. Springer-Verlag.

[17] A. Ruiz-Cortés, O. Martín-Díaz, A. Durán-Toro, & M. Toro. “Improving the Automatic Procurement of Web Services Using Constraint Programming.” *Int. J. Cooperative Inf. Syst*, 14(4): 439–468, 2005.

[18] E. Thomas. *Service-Oriented Architecture: A Field Guide to Integrating Xml and Web Services*. Prentice Hall, 2004.

[19] N. Yasemin Topaloglu and R. Capilla. “Modeling the Variability of Web Services from a Pattern Point of View.” In *ECOWS*, pages 128–138, 2004.

[20] van der Aalst, W. M. P. and Jablonski, S. “Dealing with Workflow Change: Identification of Issues and Solutions.” *International Journal of Computer Systems Science and Engineering*, 15(5): 267–276, September 2000.

[21] W3C. “Web Service Choreography Interface 1.0,” August 2002. http://www.w3.org/TR/wsci/.

[22] C. Wienands. “Synergies Between Service-Oriented Architecture and Software Product Lines,” 2006. Siemens Corporate Research. Princeton, NJ.

Appendix C: Comparison of Service and Software Product Family Modeling

Mikko Raatikainen, Varvana Myllärniemi, Tomi Männistö
Helsinki University of Technology
Software Business and Engineering Institute (SoberIT)
P.O. Box 9210, 02015 TKK, Finland
{mikko.raatikainen, varvana.myllarniemi, tomi.mannisto}@tkk.fi

ABSTRACT

Service-oriented computing develops applications by composing services. In software product families, applications are developed by reusing existing assets. Hence, the approaches seem to have several similarities, although there are also differences. In this position paper, we discuss modeling methods in these two approaches. We conclude with directions for future studies for combining modeling in software product families and service-oriented computing that include variability modeling in service-oriented computing, behavior modeling and analysis in software product families, correct modeling concepts, unification modeling concepts in software product families, and reuse and a combination of methods between approaches.

C.1 INTRODUCTION

Service-oriented computing is a computing paradigm that utilizes services as fundamental elements for developing applications [24]. The vision of such a service is to be in place, and the service must have readily available functionality and be a platform-agnostic, self-describing, and location transparent computational element that supports rapid, low-cost composition of distributed applications. Typically, a service represents a business process. Service-oriented architecture (SOA) refers to a loosely coupled architectural style for services [24, 20]. The applications in service-oriented computing are developed by combining multiple services into one application [19].

A software product family, in turn, refers to a set of software products that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of assets in a prescribed way [8]. We consider "software product line" to be a synonym for "software product family." Software product family architecture and assets are developed in a special domain engineering phase. The products of a software product family are derived by reusing assets and potentially developing additional software. A key facilitator for efficient reuse in a software product family is managing variability within the assets. Variability is an asset's ability to be extended, changed, customized, or configured efficiently for use in a particular context [30]. Domain engineering aims at introducing needed variability into the assets. Variability is bound when assets are reused in product derivation.

These two approaches seem to have a great deal in common. For example, both aim at efficiently developing applications from existing pieces of software. However, there are also differences. For example, typically,

services are dynamic computational elements composed into applications, whereas the products in a software product family are usually derived by reusing and resolving variability in static elements, often referred to as components.

Specifically, in both approaches, different kinds of modeling have received a great deal of interest. Within software product families, several variability modeling approaches have emerged; services rely on descriptions of services and modeling their compositions in order to develop applications. In addition, WS-* standards [10, 37] essentially define different languages for expressing different aspects of services as models. Since the approaches share commonalities, it seems that the modeling methods of one approach could take advantage of modeling methods in the other approach.

In this position paper, we discuss the similarities and differences in service-oriented computing modeling and software product family modeling. We begin by briefly describing modeling in software product families and service-oriented computing in Sections C.2 and C.3, respectively. Section C.4 compares the similarities and differences of the modeling methods in the two approaches. In Section C.5, we discuss the approaches in terms of how they could benefit from modeling methods used in the other approach. Finally, Section C.6 draws conclusions for future directions in combining modeling in service-oriented computing and software product families.

C.2 SOFTWARE PRODUCT FAMILY MODELING

For a software product family model, it is important to be able to express what kind of product variants can be derived from the assets at hand. Therefore, many modeling approaches for software product families concentrate on introducing variability. In this section, we outline different kinds of variability modeling approaches.

Typically, there is a differentiation between a software product family model, which contains variability, and a product model, in which variability is bound. Thus, a product family model expresses the rules and relationships of how model elements can be combined within the product model, whereas a product model is an instantiation of the family model. This differentiation adheres to the separation of domain engineering and product derivation in a software product family.

Variability in a software product family encompasses all software artifacts from requirements to code (cf. e.g., [8, 26]). Thus, there are numerous modeling methods that aim at modeling variability within different artifacts and at different levels of abstraction.

One of the first approaches to concentrate on modeling variability is FODA feature modeling [17]. A feature can be defined as a user visible characteristic of a system. A feature model is typically a tree in which selections are made as to whether to include features in certain branches or leaves of a product. Several extensions to the original feature modeling have been proposed [9, 18, 1]. In addition, work has been carried out to study or formalize different feature modeling methods [14, 1]. Besides features, requirements-level artifacts have also been proposed to be modeled utilizing use cases [12, 11].

At the architectural level, Koala [36] is one of the first modeling methods that explicitly supports architectural variability. Others

include [32, 34, 2, 15]. Thiel and Hein [32] present an approach that adheres to and extends the IEEE standard for documenting software architecture using viewpoints [16]. In addition, approaches have been introduced that provide integrated feature modeling and architectural modeling, which means that relations between features and architectural elements can be modeled explicitly [2, 15].

In addition to the above methods, which include constructs for modeling software artifacts and variability in the same model, different modeling approaches that augment software artifact models with variability specific models have been developed. The artifact models can be UML or other generic software engineering modeling approaches. For example, orthogonal variability modeling (OMV) [26] describes only variability and constraints within variability in a separate model from software artifacts. This variability model is then used to refer to, e.g., component or process models to express the variability in such a model. Covamof [28] is another approach that has a similar variability model, but constraints are expressed in yet another model.

General-purpose modeling methods, such as UML, lack specific concepts and constructs for modeling the variability of a software product family, but certain UML constructs can be used to do so. Hence, primarily, they are not meant to be used for modeling variability, although they can be used to model the products of a software product family. Further, extensions to UML have been proposed to model variability [11]. In addition, since software product family development can be considered a special case of software engineering, the commonalities, i.e., the parts that do not contain variability, are feasible to model using existing software engineering methods, such as UML.

C.3 SERVICE-ORIENTED COMPUTING MODELING

In the following, we outline modeling in service-oriented computing. We aim to provide a general description. However, since Web services are currently the dominant implementation of service-oriented computing, most modeling approaches focus on them. In addition, most concrete modeling approaches are developed for Web services. Hence, the description is based mainly on Web service modeling. Nevertheless, it seems that similar approaches are used in other kinds of service-oriented computing as well.

Modeling in service-oriented computing is typically driven by different standards, such as WSDL and BPEL in Web service modeling. However, the standards are not established or do not typically go through a rigorous standardization process [37]. Nevertheless, the different methods are developed within a community, such as the World Wide Web Consortium (W3C) [31]. The notation used for models is usually XML, although some graphical notation is used as well.

Services differentiate between the description and implementation of a service: A service description is a model of the service consisting of the service capabilities, interface, behavior, and quality [23]. On the basis of the service description, the service can be used, i.e., found, bound, and composed in an application. The state of the art in Web services is to use WSDL in service descriptions [39, 33]. WSDL is an XML-based notation. However, current WSDL and many other descriptions have limitations in describing semantics of the service [33].

Service compositions describe how services and operations of services are glued together

to provide composite services. That is, a composition model of services specifies the order in which the service operations are executed in a composite service or an application [29]. Several different service composition approaches have been presented, such as BPEL [5] and OWL-S [22].

Besides simple service composition, extensions have been proposed to cover concepts at a more abstract level. Typically, such concepts try to model business processes. Orriens et al. [21] present a business collaboration development framework and modeling method including language for specifying rules. The framework takes into account different levels of abstraction and different points of view. Business-driven automated composition is another approach that roughly means specifying requirements at the business level and then, from the requirements, deriving service composition automatically [25]. Business Process Modeling Notation (BPMN) [6] of OMG provides notation for a high-level description of a business process. In addition, a mapping from BPMN to BPEL providing automatic generation has also been described [38].

Several different aspects for Web services are described in specifications referred to as WS-*. However, there are numerous different specifications, and few of them have gained an established position despite being called standards [37].

C.4 COMPARISON

Software product family modeling and service modeling have several similarities but also differences. In this section, we compare these similarities and differences.

C.4.1 Domain and Product Modeling

Software product family modeling involves domain and product models. The entities of domain modeling are instantiated in product models. However, the main focus in software product families is on modeling the domain and describing the variability of a software product family. Further, not all approaches explicitly address instance models. Service-oriented computing focuses mainly on modeling the products, i.e., service compositions. Despite service models using WSDL being considered models of reusable entities, there is no modeling of possible service compositions or rules for service composition similar to models containing variability in software product families.

C.4.2 Composition vs. Decomposition

A software product family typically decomposes artifacts into fine grained artifacts, whereas service-oriented computing is a bottom-up compositional approach to combine artifacts into larger entities. Decomposition or top-down modeling means that a software product family architecture specifies the decomposition of a family into architectural components. However, there are also software product family approaches, such as product populations modeled using Koala [35], in which the approach is a mixture of bottom-up and top-down approaches. In service-oriented computing, there is typically no special architecture that specifies the decomposition. Rather, the SOA defines only architectural style for applications, and application development is a compositional approach from small services into larger composite services that finally form the applications. The models in service-oriented computing are developed similarly to describe such compositions bottom up from single services to compositions of service. However, technically, there is nothing to

prevent decomposition in services or composition in software product families.

C.4.3 Modeling Concepts

Both approaches use different modeling artifacts, such as those corresponding to requirements and executable software entities. However, both approaches focus primarily on modeling concerns at architectural level entities: In service-oriented computing, these are services, while in software product families, these are different kinds of architectural entities, such as processes and components. Central in both approaches are also entities roughly corresponding to requirements, i.e., features in software product families and business processes in service-oriented computing. Modeling in a software product family, especially in case of OVM [26], can also take into account software models such as detailed design artifacts. The main difference is that modeling in a software product family concerns different kinds of entities, including static and dynamic ones, whereas service-oriented computing models concerns only dynamic entities. Typically, software product families focus on static entities.

C.4.4 Relations in Models

Both software product families and service-oriented computing model basic relations between entities, such as the compositional structure of components or services and connections between the interfaces of components or services. In addition, both approaches aim at relationships between the requirements models and the implementation models. Such relationships can be used to generate the composition of lower-level entities. That is, from features can be derived component compositions in software product families, and from business processes can be derived service compositions in service-oriented computing. However, in a software product family, also modeled are more complex relations such as required or excluded relations in a variability model.

C.4.5 Modeling Notations

Software product families rely on different kinds of modeling notation, some of which build on or augment state of the practice notations, such as UML, and some being peculiar to software product families, such as feature modeling. Typically, such notation has graphical syntax, although its textual counterpart is sometimes also specified. Often, each variability modeling approach introduces its own notation or at least changes existing notation a bit. Service-oriented computing, in turn, relies mainly on XML-based notation. Consequently, the modeling notation in software product families and service-oriented computing differ quite significantly.

C.4.6 Establishment

Software product family modeling is characterized by different modeling initiatives, whereas service-oriented computing strives for standards. However, the standards in service-oriented computing are not clearly established. Instead, there are several competing standards. Frequently, standardizations merely claim a notation to be standard without passing through a rigorous standardization process. Nevertheless, the modeling approaches in service-oriented computing are often created by a community or at least several companies, whereas for software product families, the modeling methods are created by individual researchers or research groups. For example, numerous different feature modeling methods have been proposed that do not differ significantly from each other [1].

C.4.7 Stakeholders

Software product family modeling takes into account a wider scope of stakeholders than is typically done in service modeling. That is, software product family modeling adheres to conventions of viewpoint-based software architecture description that acknowledge a large group of different stakeholders (cf. [7, 27]). Software product family modeling takes into account stakeholders from developers to customers. Service orientation, in contrast, typically does not address operation or deployment; hence, modeling is, in that respect, more limited.

C.5 DISCUSSION

A major difference between the approaches from the service point of view is the lack of domain modeling or variability modeling in service-oriented computing, although service-oriented computing aims at efficiently composing different composition services, i.e., product variants, from existing services. Such a variability model would express rules for different applications or service compositions.

On the one hand, service-oriented computing is, in principle, a compositional approach in which services are composed to applications. Hence, establishing a domain model in service-oriented computing would restrict the composition and would stand in stark contrast to service composition principles.

On the other hand, service-oriented computing is usually applied in the context of business processes. It seems that such processes have several constraints in terms of how they can be composed. The constraints may originate from meaningful process order, i.e., some information needs to exist before a process can proceed - from policies set by a company, i.e., certain information may not be shared with outsiders. Therefore, it seems feasible to introduce domain modeling in service-oriented computing to constrain service composition at least to certain application domains.

In addition, although originally software product families have been strictly decomposition-driven approaches such that the products of a software product family are determined by the software product family architecture [3, 8], recently different initiatives toward more composition-oriented approaches have been proposed [35, 4]. Consequently, a challenge also to variability modeling is to develop methods that do not require strict structural architecture but rather enable the expression of principles, design rules, and design constraints [4].

From the point of view of software product families, modeling in service-oriented computing seems more restricted in terms of scope, which focuses, at the architectural level, mainly on the behavior of systems. That is, there are several different viewpoints adhering to the concept of an architectural viewpoint that can be used to model a software product family, whereas service-oriented computing models mainly behavior at the level of service and business processes. However, software product families typically concentrate on the static modeling of components, and other concepts have not received as much attention. In fact, many architectural variability modeling methods contain constructs primarily for static elements. Hence, the modeling concepts for modeling dynamic aspects in software product families could be taken from service-oriented computing.

Further, service-oriented computing studies different kinds of verification techniques for behavior [25, 19]. These techniques could

also be applied to verification of the application of a software product family.

In addition, at the level of user visible characteristics, software product families predominantly rely on feature models, although use cases and other methods have been proposed as well. Nevertheless, other modeling concepts that could be used in software product families are business process modeling of services. In particular, business process modeling seems feasible for software product families of information systems.

This plethora of modeling concepts in software product families and the few concepts in service-oriented computing raises the question of what modeling concepts should be used in service-oriented computing or software product families. Not all modeling concepts of software product families are directly applicable to service-oriented computing. Nevertheless, it seems that service modeling could be based on a similar viewpoint-based approach [16], as architectural modeling can also be applied in software product families. However, the modeling concepts of service-oriented computing can be at least partially different than those typically applied in software architecture modeling. For example, four different viewpoints have been proposed for configurable service modeling [13].

A notable difference is that modeling in service-oriented computing is mostly based on XML-based languages and developed within a certain kind of community, although such a community can be relatively small [37]. Some methods have gained an established position relatively quickly such as BPEL or WSDL. Typically such methods are described thoroughly in standards, and many are familiar with them. Within software product families similar established notations are lacking. Instead, there is a plethora of different notations, which differ from each other slightly and which are even hard to differentiate from each other. Established methods are needed in service-oriented computing, since such service can be developed by different parties. Software product families differ in that they are not typically intra-organizational, hence understanding of the methods need not, in that respect, be as wide. Nevertheless, since service-oriented computing has succeeded in achieving such established forms of notation, it seems that software product family modeling could also aim at a more coherent conceptual basis and notation. This is especially needed, if variability modeling is to be applied in a wider context than intra-organizationally, such as in service-oriented computing.

Finally, despite the differences, combined modeling methods could be developed, e.g., for behavior modeling, in which the same concepts are used for software product families and service-oriented computing. Such an approach could even combine notation: modeling in software product families could provide graphical representations, whereas modeling in service-oriented computing could provide the textual format.

C.6 CONCLUSIONS

In this paper, we discussed and compared modeling in service-oriented computing and software product families. While the aim of both approaches is relatively similar, there are notable differences. This study suggests the following challenges for further study: First, extensibility and feasibility of variability modeling should be studied in the context of service-oriented computing. Second, variability modeling in software product families should take a lesson from behavior modeling and analysis of services and business processes in service-oriented computing. Third, the necessary concepts for the

modeling of services and software product families should be studied more thoroughly. Fourth, variability modeling in software product families should aim toward unifying the fragmented conceptual foundations and notation. Finally, it seems feasible for both approaches to apply and reuse modeling methods from other approaches.

ACKNOWLEDGMENTS

The authors acknowledge the financial support of Tekes, the Finnish Funding Agency for Technology and Innovation.

REFERENCES

[1] T. Asikainen, T. Männistö, and T. Soininen. A unified conceptual foundation for feature modeling. Software Product Line Conference, 2006.

[2] T. Asikainen, T. Männistö, and T. Soininen. Kumbang: A domain ontology for modeling variability in software product families. Advanced Engineering Informatics, 21(1), 2007.

[3] J. Bosch. *Design and Use of Software Architecture.* Addison-Wesley, 2000.

[4] J. Bosch. The challenges of broadening the scope of software product families. Communications of the ACM, 49(12), 2006.

[5] BPEL. www.ibm.com/developerworks/library/ws-bpel/. visited June 2007.

[6] BPMN. http://www.bpmn.org/. visited June 2007.

[7] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond.* Pearson Education, 2002.

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

[9] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley, 2000.

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

[11] H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley, 2004.

[12] G. Halmans and K. Pohl. "Communicating the variability of a software-product family to customers." *Software and Systems Modeling, 2*(1), 2003.

[13] M. Heiskala, J. Tiihonen, A. Anderson, and T. Soininen. "Four-worlds model for configurable services." Joint Conference IMCM'06 & PETO'06, 2006.

[14] P. Heymans, P.-Y. Schobbens, J.-C. Trigaux, R. Matulevicius, A. Classen, and Y. Bontemps. "Towards the comparative evaluation of feature diagram languages." Software and Services Variability Management - Concepts, Models and Tools Workshop, 2007.

[15] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor. *Configuration in Industrial Product Families*. IOS Press, 2006.

[16] IEEE. IEEE Std. 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems-Description, 2000.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021, SEI, 1990.

[18] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: "A feature-oriented reuse method with domain-specific reference architectures." Ann. Soft. Eng., 5(1), 1998.

[19] N. Milanovic and M. Malek. "Current solutions for web service composition." IEEE Internet Computing, 8(6), 2004.

[20] N. Milanovic and M. Malek. "Search strategies for automatic web service composition. International." Journal of Web Services Research, 3(2), 2006.

[21] B. Orriens, J. Yang, and M. Papazoglou. "A rule driven approach for developing adaptive service oriented business collaboration." IEEE International Conference on Services Computing, 2006.

[22] OWL. http://www.daml.org/services/owl-s/. visited June 2007.

[23] M. Papazoglou. "Service-oriented computing: concepts, characteristics and directions." Fourth International Conference on Web Information Systems Engineering, 2003.

[24] M. Papazoglou and D. Georgakopoulos. "Guest editor introduction: Service oriented computing." ACM SIGSOFT Software Engineering Notes, 46(10): 24–28, 2003.

[25] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Kramer. "Service-oriented computing: A research roadmap." Dagstuhl Seminar Proceedings, 2006.

[26] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques.* Springer, 2005.

[27] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2005.

[28] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: "A framework for modeling variability in software product families." Software Product Line Conference, 2004.

[29] S. Staab, W. van der Aalst, V. R. Benjamins, A. Sheth, J. A. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon. "Web services: Been there, done that?" IEEE Intelligent Systems, 18(1): 72, 2003.

[30] M. Svahnberg, J. van Gurp, and J. Bosch. "A taxonomy of variability realization techniques." Software — Practice and Experience, 35, 2000.

[31] The World Wide Web Consortium (W3C). http://www.w3.org/. visited May 2007.

[32] S. Thiel and A. Hein. "Systematic integration of variability into product line architecture de sign." Software Product Line Conference, 2002.

[33] M. Turner, D. Budgen, and P. Brereton. “Turning software into a service.” IEEE Computer, 36(10), 2003.

[34] A. van der Hoek. “Design-time product line architectures for any-time variability.” Science of Computer Programming, 53(3), 2004.

[35] R. van Ommering. “Building product populations with software components.” International Conference on Software Engineering, 2002.

[36] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. “The Koala component model for consumer electronics software.” Computer, 33(3), 2000.

[37] S. Vinoski. “WS-nonexistent standards.” IEEE Internet Computing, 8(6), 2004.

[38] S. A. White. “Using BPMN to model BPEL process.” Whitepaper, http://www.bpmn.org/, 2005.

[39] WSDL. http://www.w3.org/TR/wsdl. visited June 2007.

Appendix D: Identifying and Specifying Reusable Services of Service Centric Systems Through Product Line Technology

Jaejoon Lee, Dirk Muthig,
and Matthias Naab
Fraunhofer Institute for Experimental
Software Engineering (IESE),
Kaiserslautern, Germany
{jaejoon.lee, dirk.muthig, matthias.naab}@iese.fraunhofer.de

Minseong Kim, Sooyong Park
Sogang University,
Seoul, R.O.Korea
{minskim, sypark}@sogang.ac.kr

ABSTRACT

The concept of service orientation (SO) is a relevant promising candidate for accommodating rapidly changing user needs and expectations. Adopting SO in practice for real software and system development, however, has uncovered several challenging issues, such as maintaining consistent system configuration or integrity of dynamically composed services, or identifying reusable services at the right level of granularity. In this paper, we propose an approach that addresses the latter issue, which we map to the well-known challenge of defining reusable software assets. The approach is adapted from the analysis technique of product line engineering, which is the most successful approach for establishing reuse in practice. We present how reusable services can be identified and specified based on features: these features identify variations of a family of products from a user's point of view and thus will be the subjects of reconfigurations of service centric systems at runtime.

D.1 INTRODUCTION

The concept of service orientation (SO) is a relatively new paradigm for software development: systems are no longer developed, integrated, and released in a centrally synchronized way, but services are developed and deployed independently and separately, as well as composed as late as at runtime if and when needed only. That is, service consumers are mostly decoupled from service providers. This corresponds to the main property of SO: a great amount of inherent flexibility. This flexibility leads to perfect scalability characteristics because a network can be populated by as many services as wanted but only affect the systems that actually require them.

User needs and expectations change continuously, and thus software systems must evolve rapidly, to accommodate user expectations. More and more software systems are connected to the Internet, and thus their evolution could be supported and accelerated by dynamically adding and integrating services. Hence, the SO paradigm is a relevant promising candidate for addressing evolution challenges. Thus SO has gained great attention by practitioners, as well as by researchers.

Adopting SO in practice for real software and system development, however, has uncovered several challenging issues, such as maintaining consistent system configuration or integrity of dynamically composed ser-

vices, or identifying services at the right level of granularity. In this position paper, we propose an approach that addresses the latter issue by mapping it to the well-known challenge of defining reusable software assets. In SO, a service is the basic building block for system construction. Thus integrating existing services, which were developed in potentially different contexts by different people, means nothing else than reusing software.

The reuse process consists of several steps: identification of reuse candidates, evaluation of these candidates, selection of the best reuse candidate for the given context, and adaptation and integration of the selected candidate into the system under development. There are many experience reports that emphasize problems and challenges in implementing software reuse in general. Reusing a service corresponds to the reuse of a component providing a single method only. From our point of view, realizing the reuse of services is nevertheless more challenging than realizing the reuse of components. That is because the SO reuse process is supposed to be executed automatically by a software system at runtime without any consultation of human experts.

In our research, we investigate this reuse aspect as an inherent part of the SO paradigm's nature. We apply the concepts of product line engineering—which is the most successful approach for establishing reuse in practice—to the SO paradigm. That is, we tailor Fraunhofer PuLSE™ (Product Line Software and System Engineering)[6] [1] to the SO paradigm and thus enable the efficient construction and evolution of service centric software systems.

D.2 APPROACH OVERVIEW

In this position paper, we propose a technique for identifying and specifying reusable services. This technique is based on analyzing and specifying features that may vary from a user's point of view and thus will be subjects of reconfigurations at runtime.

Figure D-1 shows activities and their relationships to the technical components. These activities are executed iteratively; the arrows in Figure D-1 indicate the flow of data and which work products are used by each activity.

A feature analysis organizes product family features into an initial model, which is then refined by adding design features such as operating environments, domain technologies, or implementation techniques. Within the feature model, the subsequent binding analysis identifies binding units and determines their relative binding times among each of the others [2].

The service analysis consumes the results of these analyses. Each binding unit is further analyzed to determine its service category (i.e., orchestrating service or molecular service) with respect to the particular family at hand. We assume here families whose variations can be described best by variations in workflows executed by the system users. Additionally, the context and the technical infrastructures available vary, and thus dynamic reconfigurations of product variants are expected.

The mass of low level services, that we call atomic services, are grouped into richer services as required by the family. These richer services are (virtually) composed of atomic services and are thus called molecular services. Note that each product family has thus its own specific set of molecules, the basic building blocks for constructing family members. Due to the definition of those molecules based on product line processes, molecular services are more reusable than atomic services (in the context of a particular product family).

[TM] PuLSE is a registered trademark of the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany.

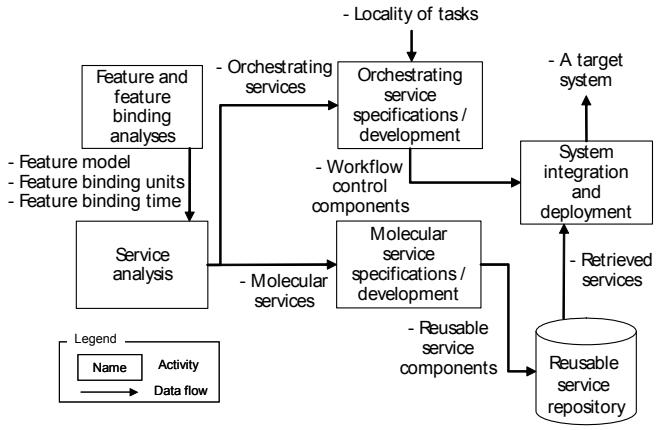

Figure D-1: Activities of the Approach

From a technical viewpoint, the identified services are specified first as workflows and their constituting tasks. Then, their pre/post conditions, invariants, and service interfaces are specified. Note that also the quality of services (QoS) may vary due to different service configurations. Finally, the system integration and deployment activity form a product by integrating the reusable services provided by the previous activities.

For illustrating the approach presented in this paper, we selected a case study in the domain of the virtual office of the future (VOF). The VOF product family consists of systems, which control and manage collections of devices to provide any-time anywhere office environments [9].

D.3 FEATURE ANALYSIS

In this section, activities of feature analysis—which includes feature modeling and feature binding analysis—are introduced. Feature modeling is the activity of identifying externally visible characteristics of products in a product line and organizing them into a model called feature model [10]. Figure D-2 shows, for instance, a part of the feature model for the VOF product line. The primary goal of feature modeling is to identify commonalities and differences of products in a product line and represent them in an exploitable form, i.e., a feature model.

Common features among different products in a product line are modeled as mandatory features (e.g., *Resource Manager* and *Follow Me*), while different features among them may be optional (e.g., *Auto Log-on*) or alternative (e.g., *User Positioning Method*). Optional features represent selectable features for products of a given product line, and alternative features indicate that no more than one feature can be selected for a product. Details of feature analysis and guidelines can be found in [10].

Once we have a feature model, it is further analyzed through feature binding analysis [2]. Feature binding analysis consists of two activities: feature binding unit identification and feature binding time determination. Feature binding unit identification starts with identification of service features. A service feature represents a major functionality of a system and may be added or removed as a service unit. In VOF, *Follow Me*, *Resource Management*, *Virtual Printer*, and *Smart Business Trip* features are examples of service features.

A set of features that should be included in a feature binding unit are identified by traversing the feature model along feature relationships. For example, *Follow Me*, *User Authentication*, *Manual Log-on*, *Auto Log-on*, *User Positioning Method*, *Access Point based Method*, and *RFID based Method* belong to the *FOLLOW ME* feature binding unit. Note

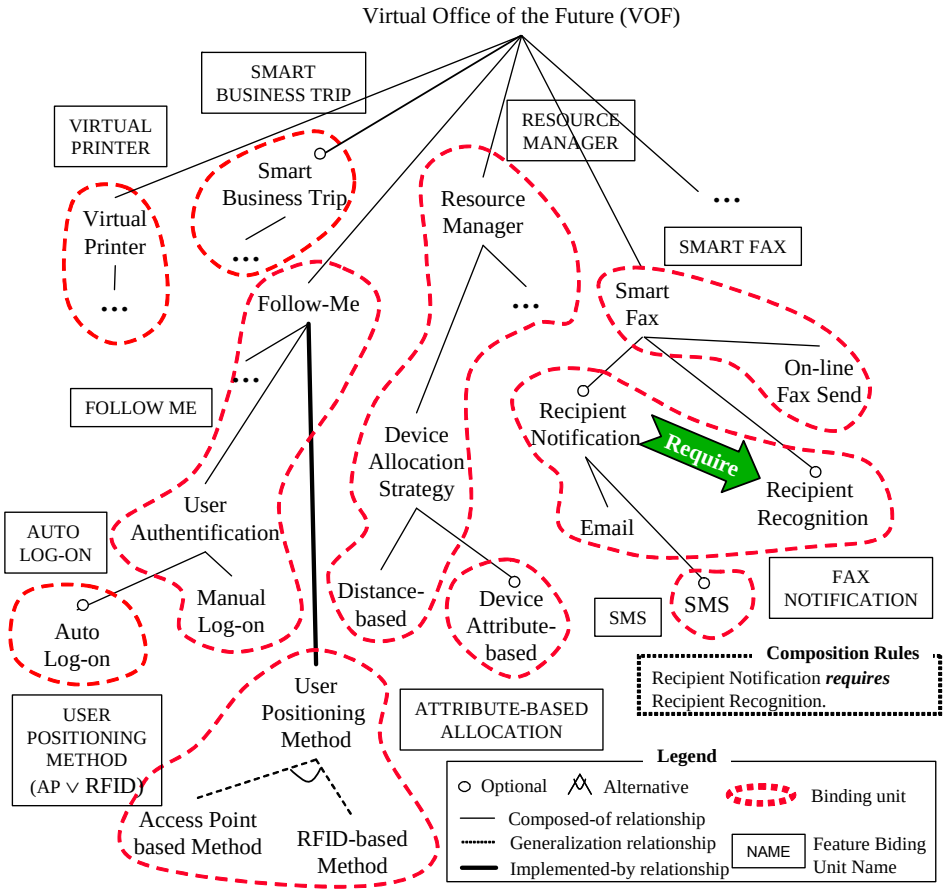

Figure D-2: A Feature Model and Binding Units of VOF [11]

that the optional *AUTO LOG-ON* and the alternative *USER POSITIONING METHOD* are identified as separate feature binding units, because they may have different binding time from their parent feature binding units. (See Figure D-2 for their identification.) Note that alternative variants of an alternative feature binding unit are listed in parentheses (e.g., *AP* or *RFID* for *USER POSITIONING METHOD* in Figure D-2.)

Because a feature binding unit contains a set of features that need to be bound together into a product to provide a service correctly and share the same binding time, a product can be considered as a composition of feature binding units. By taking these feature binding units as a key driver for service analysis, we could alleviate the difficulties for identifying candidate services with right granularity, i.e., reusable services.

In the next section, how the identified candidate services (i.e., feature binding units) are further classified and refined is explained.

D.4 SERVICE ANALYSIS

Through the previous activities, we now have a feature model and feature binding information, which provides an insight into a targeting domain in terms of product features, basic units of binding, and their binding time.

Then, the feature model is refined and restructured by introducing a separation of two distinctive service characteristics: behavioral (workflow) and computational (tasks) service characteristics.

A behavior oriented service is mainly to define a certain sequence of tasks, i.e., workflows. We call services in this category orchestrating services, as their main role is the composition of other services in a harmonious way. A computation oriented service is to provide computational outputs (i.e., a predefined task) in response to given inputs. We call services in this category molecular services, as they are the basic building blocks and will be reused as-is by orchestrating services. Details of services that belong to each category are explained in the following sections. (See Figure D-3 for the refined feature model with the two service layers.)

D.4.1 Orchestrating Service

For orchestrating services, correctness of their overall control behavior is the foremost concern. For example, providing an expensive color-printing service with proper authorization and billing processes is critical for virtual office service providers. Therefore, adopting a formal method framework to specify, validate, and verify is the most suitable way for developing orchestrating services. In our approach, we adapted a workflow specification language [11] with pre/post conditions and invariants to enhance the reliability of specifications.

Figure D-4 shows a workflow specification example for a business trip service. Each orchestrating service has pre/post conditions and invariants. In this example, a user should be logged in to trigger the service, and the workflow is completed only after the user submits a postmortem report about her/his business trip. Also, the invariants (i.e., the user is employed and the business trip is not cancelled) should hold through the whole workflow process. When ever the invariants become invalid, the workflow is terminated with proper notifications to relevant stakeholders.

Moreover, each task of the workflow can be specified with its pre/post conditions and invariants. For example, a secretary should achieve the access rights to organizational data such as the charged project's budget information and the traveler's bank account number to proceed with the reservation task. Such conditions can be defined as the precondition of the reservation task and checked when a secretary is assigned for the task. Note that the consistency of invariants between a workflow and its constituting tasks should be checked when an orchestrating service is specified.

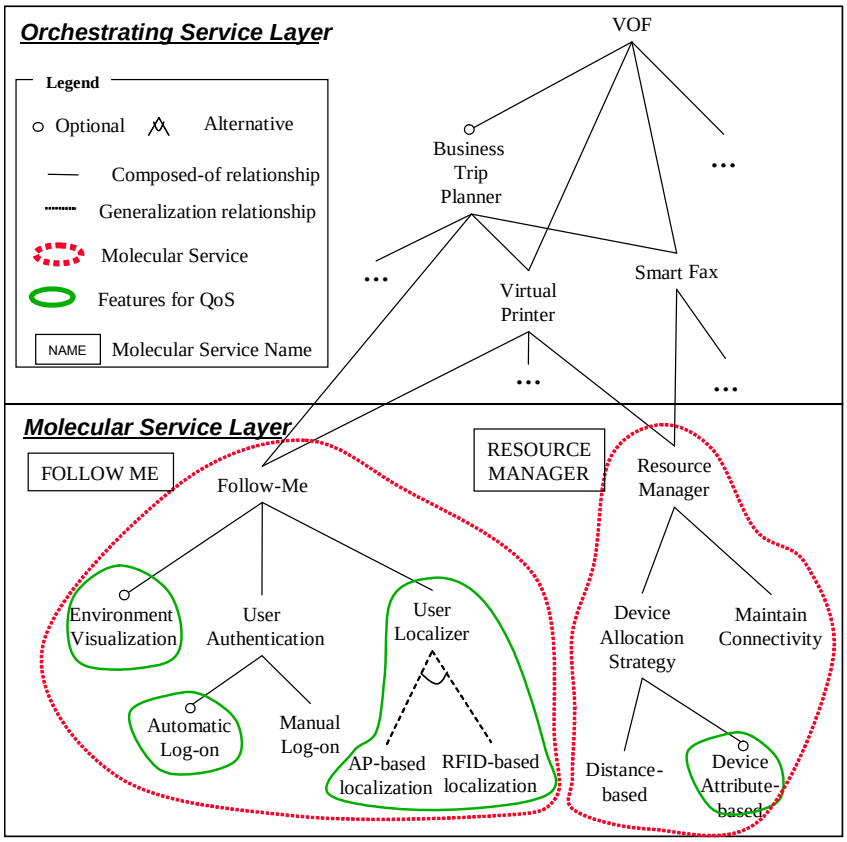

Figure D-3: A Refined Feature Model Based on Two Service Categories

In addition to the identification of tasks and their pre/post conditions and invariants for an orchestrating service, the locality of each task should also be identified for high availability of services. By locality we mean that the information of the responsible person of a task and her/his physical location where the task is performed. The locality information is particularly important for a domain. In addition to the identification of tasks and their pre/post conditions and invariants for an orchestrating service, the locality of that should support mobility of users like the VOF systems. For instance, the visa process and reservation tasks are local to a secretary, and they can be processed without the coordination at the global level. This means that the secretary can perform the tasks locally although she/he is disconnected from a network. Also, the physical location is important to assign the most relevant business peripherals such as a printer or a fax machine. However, the approval status of a business trip by a deciding staff should be managed at the global level to trigger tasks that belong to other persons.

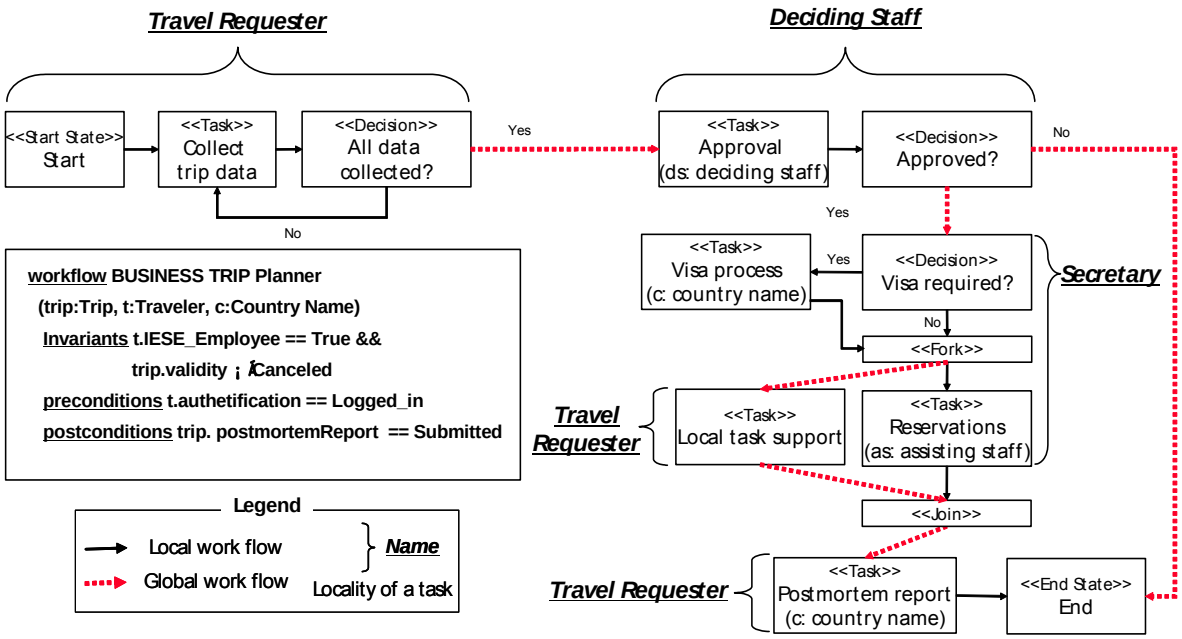

Figure D-4: An Example of Workflow Specification for an Orchestrating Service: Smart Business Trip

Next, the identification and specification of molecular services are explained.

D.4.2 Molecular Services

The identification of molecular services with right granularity is the key factor to enhance reusability of the service centric system development. Molecular services are the basic units for reuse, and orchestrating services should be able to compose them as-is through their interfaces during development time or their runtime. For their identification, feature binding units are analyzed and refined with consideration of the following guidelines. A molecular service should be

- self-contained (local control and local computation)
- stateless from service user's point of view
- provided with pre/post conditions
- representative of a domain-specific service

The first three guidelines are to decouple service consumers from providers. Based on these guidelines, a service consumer only needs to know the service providers' interface and their conditions for use. This means that any changes (performance improvements bug patches, etc.) within an identified molecular service must not be propagated to other services.

The last guideline is the key factor to determine the right granularity of a molecular service based on the feature binding unit and time information, and domain experts' professional judgment. For instance, the feature binding units related to *Follow Me* and its descendent feature binding units in figure D-2 are identified and reorganized as the *FOLLOW ME* molecular service in Figure D-3. The rationale for this determination is as follows:

- the *Follow Me* feature is a mandatory service for every user of the VOF product line
- each localizing device (e.g., RFID, access points of wireless networks, etc.) uses different localization techniques, but its expected outputs are the same (e.g., a user's physical location)

```
1: molecular service FOLLOW ME (user User)
2: invariants user.IESE_Employee == true
3: precondition user.authentification == logged_in
4: postcondition none;
5:   option Environment Visualization
6:     binding time run time
7:     precondition user.device == desktop ∨ notebook
8:     postcondition none;
9:   option Automatic Log-on
10:    binding time run time
11:    precondition user.rank == director ∨ manager and
12:        RFID bases user location method == available
13:    postcondition user.access == granted ∨ rejected;
```

Figure D-5: An Example of Molecular Service Specification

- the implementing algorithms for localization evolve rapidly to improve their accuracy
- it is a computation oriented service without any workflows in it

Based on this decision, the *FOLLOW ME* molecular service is designed and implemented to provide the user localization service to the orchestrating services, if they abide by the pre/post conditions of *FOLLOW ME*.

Each molecular service may have its QoS parameters, which are identified during the feature binding analysis in terms of optional or alternative features. For example, the *User Positioning Method* feature binding unit has two alternatives (e.g., AP-based and RFID-based method), and their levels of accuracy are different (e.g., the error range of the RFID-based method is less than 1 meter, whereas the error range of the AP-based method is less than 10 meters). Depending on available devices near a user, one of the alternative positioning methods is selected and used.

In our approach, each molecular service is specified by using a text-based specification template, and Figure D-5 shows the specification of *FOLLOW ME*. (The characters in the bold font are reserved words for the specification.) The *FOLLOW ME* service is for the current employees, who passed the authentication and logged in. Also, the *Automatic Log-on*, which is optional for higher quality of the service, is only available at runtime when the requesting user's job function is director or manager, and an RFID device is available nearby. (See the lines 9 to 13 for the specification of optional feature *Automatic Log-on*.)

In this section, concepts and guidelines for analyzing and specifying orchestrating and molecular services are explained. The next section discusses and evaluates our approach.

D.5 RELATED WORK

While our approach concentrates on achieving reusability by means of proper identification and specification of services using product line technologies, in [3], reusability is claimed to be achieved by the structure of systems and the interaction mechanisms. This mainly means the availability of a service repository and the concepts for discovering, negotiating, and binding services.

IBM developed a method, called "Service-Oriented Modeling and Architecture" [4, 5]. It provides guidelines for three steps towards

SO systems: identification, specification, and realization of services, flows, and components. In particular, a combination of three complementary ideas is proposed to identify services in [4]. First, the domain of the respective software systems is analyzed and decomposed. Second, existing legacy systems are explored in order to discover parts to be reused as services. Third, business goals are taken into account to complete the identification of services. The first and the third ideas are reflected in our approach. Also, our approach supports the service identification by the proven method of feature-oriented analysis and design and thus puts additional structure on the method.

The approach of IBM further suggests organizing services in a hierarchy of services of different granularity. By comparison, our approach adds the dedicated layer of molecular services that form reusable assets in the specific domain. According to the respective domain, the molecules would be composed in different ways to optimally fit the requirement of reuse. Thus, reuse becomes easier by only selecting from a rather small number of assets with well-tailored granularity. Additionally, the concept of flows of services is mentioned to be important in [4]; however, there are no details about the identification or specification of these flows. On the other hand, our approach incorporates the defined molecular services as the building blocks with which to orchestrate workflows.

Another approach of using feature-oriented analysis to identify services for an SO system is described in [6]. Their main focus is on reengineering towards SO systems. They claim to do a feature analysis of the particular system and use the result as input for the service identification. Yet, they do not provide concrete guidelines on how to come up with services of the right granularity.

While methods for the identification of orchestrations of services are hard to find, there are a number of languages to express such orchestrations. For instance, in the field of Web Services, BPEL4WS (Business Process Execution Language for Web Services) [7] is widely used to realize SO systems. It represents a language to specify orchestrations of services that are then accessible as higher-level services. While BPEL is well-suited for the pure orchestration of services, it has some deficiencies in the area of business processes that comprise human interaction during the business process. We addressed this by combining ideas from workflow-management, which is explicitly designed for human interaction, and service orientation. Thus, in our approach, orchestrated services are described as workflows.

A further concept we transferred to service composition is “Design by Contract” [8]. We enriched the composition language and service description by pre/post conditions and invariants that can be automatically verified. Hence, the reliability of service-composition, static as well as dynamic, can be improved by checking the correct usage of services. The reusability of services is also improved with advanced description, since automatic checks can reduce the number of feasible candidate services, which makes selection easier.

D.6 CONCLUSION

We have transferred product line technology into industry since 1998, and we’ve experienced in nearly all cases a quick increase of the number features, as well as required variants. Hence, the management of features and their variations becomes soon one of the

major challenges in maintaining and evolving viable reuse infrastructures. The environment and context of service-oriented systems is typically very dynamic and always distributed. Our experience with such service-oriented product lines has shown that the challenge of managing variations and keeping services reusable and useful over a long period of time is even bigger than for other systems.

In this position paper, we propose an approach that alleviates this difficulty through the grouping of features into feature binding units of the same binding time, as well as by interpreting these units as key drivers for identifying reusable services, that is, molecular services.

The practical applications of our approach in our lab infrastructure demonstrated that product line technology can significantly help in mastering this challenge. The key property of the approach is its support for identifying reusable services at the right level of granularity abstraction.

Nevertheless, our approach is still in an early phase, where its fundamental properties are worked out in detail, as well as validated in small case studies in our prototyping environments. Currently, we have established a demonstration facility within our institute to execute real scenarios of a virtual office of the future. The infrastructure of this demonstration facility has been defined by following our approach, which has already provided useful conceptual insights and lessons learned from a practitioner's perspective.

Additionally, we are working on completing the approach to fully cover the overall product line life cycle including the evolution of product line infrastructures. As part of these activities, an architectural prototype emulating an SO environment was built and has been used to refine the architectural styles and patterns required to prepare the SO paradigm for practical contexts.

ACKNOWLEDGEMENTS

This research is partially carried out in the Cluster of Excellence "Dependable Adaptive Systems and Mathematical Modeling" project, which is funded by the Program "Wissen schafft Zukunft" of the Ministry of Science, Education, Research and Culture of Rhineland-Palatinate, Germany, AZ.: 15212-52 309-2/40 (30).

REFERENCES

[1] J. Bayer et al., PuLSE: A Methodology to Develop Software Product Lines. *Proceedings of the Fifth Symposium on Software Reusability. SSR'99,* ACM Press (1999).

[2] J. Lee and K. Kang. Feature Binding Analysis for Product Line Component Development. In: van der Linden, F. (eds.): Software Product Family Engineering. *Lecture Notes in Computer Science*, Vol. 3014. Springer-Verlag, Berlin Heidelberg (2004) 266-276.

[3] H. Zhu, "Building reusable components with service-oriented architectures," presented at IEEE International Conference on Information Reuse and Integration, (2005).

[4] A. Arsanjani, "Service-oriented modeling and architecture - How to identify, specify, and realize services for your SOA," (2004).

[5] A. Arsanjani and A. Allam, "Service-Oriented Modeling and Architecture for Realization of an SOA," in Proceedings of the IEEE International Conference on Services Computing: IEEE Computer Society, (2006) 521.

[6] F. Chen, S. Li, and W. C.-C. Chu, “Feature Analysis for Service-Oriented Reengineering,” in Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC’05) - Volume 00: IEEE Computer Society, (2005) 201-208.

[7] Andrews, Curbera, Dholakia, Goldand, Klein, Leymann, Liu, Roller, Smith, Thatte, Trickovic, and Weerawarana, “Business Process Execution Language for Web Services,” (2003).

[8] B. Meyer, “Design by Contract,” in Advances in Object-Oriented Software Engineering, D. Mandroli and B. Meyer, Eds.: Prentice Hall, (1991) 1-50.

[9] Competence Center for “Virtual Office of the Future,” http://www.ricoh.rlp-labs.de/index.html.

[10] K. Lee et al., Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Gacek, C. (eds.): Software Reuse: Methods, Techniques, and Tools. *Lecture Notes in Computer Science*, Vol. 2319. Springer-Verlag, Berlin Heidelberg (2002) 62-77.

[11] J. Lee and D. Muthig. Feature-Oriented Analysis and Specification of Dynamic Product Reconfiguration in: H. Mei (eds.): ICSR 2008. *Lecture Notes in Computer Science*, Vol. 5030. Springer-Verlag, Berlin Heidelberg (2008) 154-165.

[12] JBoss jBPM 2.0 jPdl Reference Manual, http://www.jboss.com/products/jbpm/docs/jpdl.

Appendix E: 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@uni-magdeburg.de

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

Abstract

A software product line is a paradigm to develop a family of software products 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 where 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 Web Portals and Portlets.

E.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, software product lines are primarily targeted 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 Architectures (SOAs) 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 even be 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 the portlet is 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 a scenario a *Service-Oriented Product Line* (SOPL).

This situation is 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 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 SOA. This reduces human inter-

vention during product line discovery and minimizes human intervention from consumption. This way, the challenge is how to enable the automatic consumptions of products from a third-party product line, which we address in this paper.

E.2 Service-Oriented Product Lines

There is nothing new about how multiple, distribute and heterogenous 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 2 products from product lines are manually combined together). We envisage SOPL towards the automation of multiple product lines combination.

The issue of how that product is coupled into the whole end-product is faced by *product populations*, which describe an architecture-centric approach to attain this coupling [19] (see Section E-5). This approach requires uman intervention.

We envisage for SOPL to compose products supplied from different product lines with little human intervention. To this end, several issues should 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 operation of SOPL where performance, production schedule, bill of materials and other elements should be considered beforehand, and *(iv)* adequate existing tool support.

E.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 refers to the id, name, and a brief description of the product line. This information is later used during the discovery and registration of the product line.
- Product information describes how products are distinguished in a product line setting. A product is frequently characterized by its features. This is the basic specification we need to build a product. Further information about core assets may be offered as well for descriptive purposes.
- Production interface consists first of information such as production time, delivery time, average product cost, average product LOC, average product size, and so on. An important piece of information is that 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 would be useful when choosing among concurrent product lines.

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

E.2.2 Consumer

A consumer product line demands products from third-party product lines. This demand is specified in terms of supplier´s characteristics (e.g., descriptive information). . 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 together. When they are used to create another product-line, this idea can be regarded as an SOPL.

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

E.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 E-1).

Registration The registration requires the discovery of each product line supplier (i.e., human intervention is required)[1]. Figure E-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.

Consumption The consumption refers to the production and delivery of the product. In general, when the product line production or derivation process is automated, we can

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

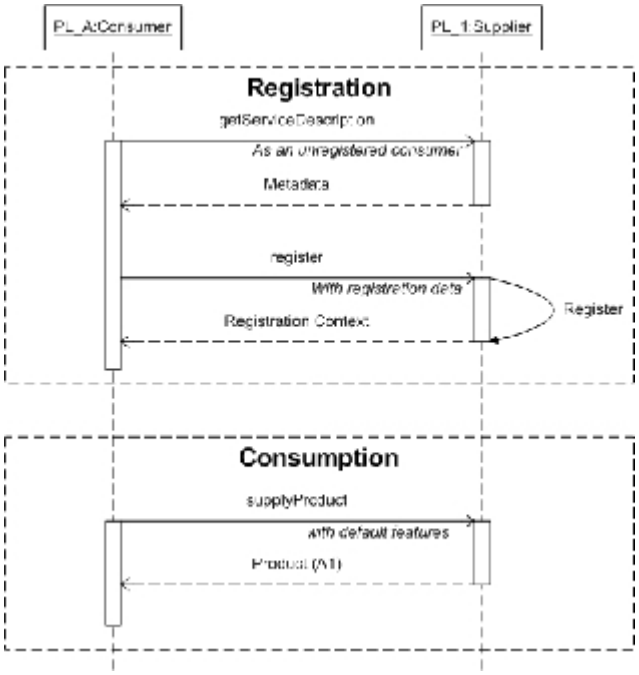

Figure E-1: Operation - Sequence Diagram

invoke such product line specifying desired features and get a product [2, 6].

Figure E-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 E-3). The supplied product is considered as a reusable asset by the product line consumer. Nonetheless, tool support is needed to automate such consumption.

E.2.4 Tool Support

The ideas presented before on consumer/supplier relationship benefit from SOA ideas. More to the point, existing SOA standardization efforts and tool support may readily enable to create such infrastructure.

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

E.3 Example

Portals and Portlets We choose *portals of portlets* to illustrate the idea of SOPL [10]. A *portal* is a Web page that

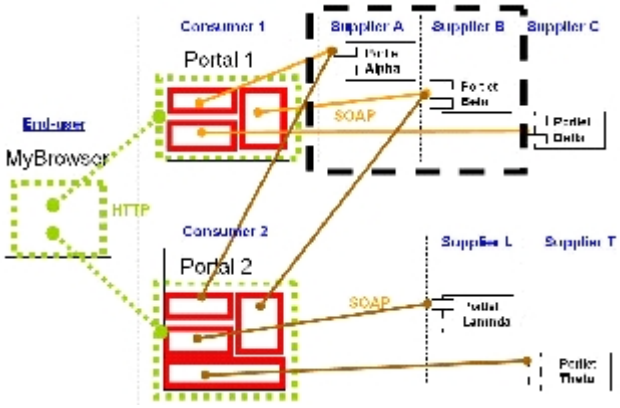

Figure E-2: Portal / Portlet Architecture

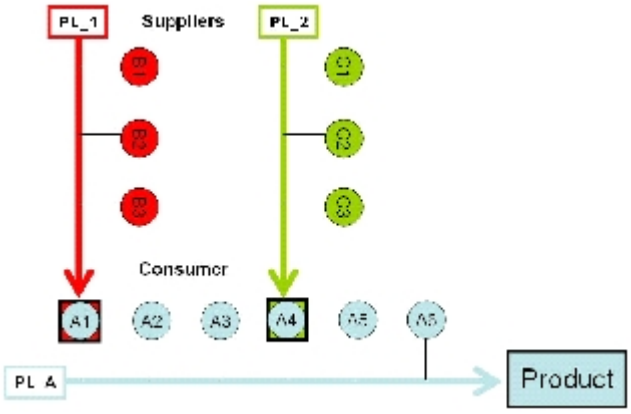

Figure E-3: SOPL Scenario

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 E-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 E-2 shows this where *Portal_2* consists of a version of *Alpha* different of 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 where different ser-

vices are offered. Each company demands a similar, but different version. So, there is a family of products. The services in this 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 E-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 *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.

E.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 fits as artifact of a larger product. This consistency issue appears when features from a consumer require to be propagated 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 viceversa. 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). Doing so, consistency, time and production issues could be considered. 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 along the time into a large product line. When this occurs, both technical and organizational management of the product line becomes 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 principle leads us 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 enable eventually to ease the 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.

E.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 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*

[2]This does not preclude that the portlet is physically deployed on the same machine than 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

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 data is compiled packaged, and deployed. Optionally, it may be measured, tested, versioned or even 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 to invoke 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.

E.6 Conclusions

This paper presents our ongoing work on the vision of SOPL, which consume 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 preliminar representations for consumer and supplier product lines, described the basic operation with registration and consumption, and sketch 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. N. 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. Greenfeld 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. 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.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	May 2008	Final

4. title and subtitle

Proceedings of the First Workshop on Service-Oriented Architectures and Software Product Lines

5. FUNDING NUMBERS
FA8721-05-C-0003

6. author(s)

Sholom Cohen, Robert Krut

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2008-SR-006

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report contains the proceedings of the First Workshop on Service-Oriented Architectures and Product Lines (SOAPL) 2007 that was held on September 10th, 2007 in Kyoto, Japan as part of the 2007 Software Product Line Conference (SPLC 2007). This report includes an overview of the workshop, four invited presentations, details of the workshop's outcomes, and the workshop position papers.

14. SUBJECT TERMS	15. NUMBER OF PAGES
workshop, service-oriented architectures, product lines (SOAPL) 2007, SOA, service features, service variability	74

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL