Combining Service Orientation with Product Line Engineering

Jaejoon Lee and Gerald Kotonya, Lancaster University

Dynamic product reconfiguration involves making changes to a deployed product configuration at runtime. Examples include dynamically adding, deleting, or modifying product features and making dynamic changes to architectural structures. Researchers have studied dynamic product reconfiguration for various application areas, including self-healing systems, context-aware computing, software component deployment, and ubiquitous computing. Detecting a change in operational context could trigger a product reconfiguration to provide context-relevant services or meet quality requirements such as performance. Thus far, these dynamic reconfiguration approaches have focused on the specific problems of each application area (for instance, behavior models for dynamic changes, context recognition from software or hardware environments, and autonomous management of software component versions). However, some researchers are beginning to investigate development issues for reusable and dynamically reconfigurable core assets, called dynamic software product lines (DSPLs).

A service-oriented product line (SOPL) is a DSPL application domain that’s built on services and a service-oriented architecture. An example of an application area for an SOPL approach is a virtual office (VO). A VO includes many business peripherals with various services that interact with one another and respond to their various environments to assist office workers. However, such a system presents several software engineering challenges. First, VOs must be highly self-adaptive so that they can dynamically reconfigure their behavior to respond to changes in the environment and to coordinate with other services in the vicinity. For example, an office worker might use different types of devices (a notebook, a PDA, or a mobile phone) at various locations in an organization (an office room, a meeting room, or a product showroom). The VO system must support the current tasks (email checking, video conferencing, or presentation) as much as possible by using and coordinating available devices and...
services, regardless of which devices are used and where they’re located.

Second, a VO’s longevity implies that products should evolve over time. New devices and services could be added, removed, or updated, and the overall requirements governing how available services should work together could change. At the same time, the VO system must continue to function effectively and satisfy its overall goals. These first two challenges concern the service orientation (SO) characteristics of a VO system. However, we also must shorten the time to market by systematically reusing core assets of the VO system—the third challenge—and this is an SPLE characteristic. Therefore, establishing an SOPL of the VO system is the key to successful deployment of multiple VO products for various organizations having their own specific requirements.

In this article, we identify the main challenges we experienced in developing an SOPL, and then briefly describe a possible solution that combines feature-oriented analysis with a QoS-aware SO framework.

**SPLE and SO Paradigms**

SPLE and SO have different engineering goals. So, the activities at each software life-cycle phase differ as well. Table 1 summarizes the major engineering activities of these two paradigms.

SPLE’s main engineering goal is the development of core assets that enable systematic reuse. To achieve this goal, the analysis phase focuses on identifying product line commonality and variability, and using this information to specify product line requirements. The product line architecture and components provide an infrastructure for efficiently instantiating various products of that product line. Ensuring a valid, working, environment-relevant product configuration is critical to deploying and maintaining a product because an invalid configuration could lead to a system crash or malfunction.

SO’s main goal, on the other hand, is system agility through runtime flexibility to cope with rapidly changing business environments. The key idea of SO is software as a service, which promises agile and flexible system development through a dynamic runtime architecture that allows for adding third-party functionality on demand. The service-oriented architecture is the conceptual structure for realizing this vision. The analysis phase’s major focus is identifying and defining how to orchestrate services in an application. A service broker provides runtime support for service discovery and selection. As such, key development issues include design considerations and constraints for efficient, dependable, and correct matching between service consumers and providers.

**Challenges to Building an SOPL**

We experienced four major challenges in building an SOPL: different notions of first-class objects as engineering drivers (features versus services), dynamic characteristics of a service-based system, involvement of third-party service providers, and variation (product configuration) control and management.

**Different Notions of First-Class Objects**

Features and services are two different notions of key engineering drivers in software development under the SPLE and SO paradigms. Features are abstractions of a product line’s user- or developer-visible characteristics. Feature orientation to analyze a product line’s commonality and variability appeals to many product line engineers because

---

**Table 1**

Comparison of the major engineering activities of software product line engineering and service orientation

<table>
<thead>
<tr>
<th>Engineering paradigm</th>
<th>Major life-cycle phases</th>
<th>Main engineering goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software product line</td>
<td>Analysis</td>
<td>■ Commonality and variability analysis</td>
</tr>
<tr>
<td>engineering</td>
<td>Design and implementation</td>
<td>■ Product line architecture and components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>design with variation points</td>
</tr>
<tr>
<td></td>
<td>Product deployment and maintenance</td>
<td>■ Production plan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Valid product configuration</td>
</tr>
<tr>
<td>Service orientation</td>
<td>■ Service identification</td>
<td>■ Systematic reuse of core assets</td>
</tr>
<tr>
<td></td>
<td>■ Service orchestration specification</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Information broker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Service protocol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Service provider and consumer registration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Service matching</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ System agility through runtime flexibility</td>
<td></td>
</tr>
</tbody>
</table>

---
features are an effective medium for supporting communication among a product line’s diverse stakeholders. Products are typically discussed and described in terms of features gathered from market surveys, individual customers, research labs, or technology roadmaps. Therefore, it’s natural for people to express commonality and variability of product lines in terms of features. Also, a feature model provides a basis for subsequent development, parameterization, and configuration of various reusable assets such as product line requirement models, reference architectural models, and code.

Features. Feature modeling identifies a product line’s features by identifying externally visible product characteristics in a product line and organizing them into a model. Product features are identified and classified in terms of capability, domain technology, implementation technique, and operating-environment features. Capability features are user-visible characteristics that can be identified as distinct services, operations, and nonfunctional characteristics. Domain technology features represent ways of implementing services or operations. Implementation technique features are generic functions or techniques for implementing services, operations, and domain functions. Operating-environment features represent the environments where the applications are used. Common features among different products are modeled as mandatory features, whereas different features among them may be optional or alternative. Optional features represent selectable features for a particular product line’s products, and alternative features indicate that no more than one feature can be selected for a product. In addition, composition rules supplement the feature model with mutual dependency and mutual-exclusion relationships to constrain the selection from optional or alternative features.

Services. A service in SO is a collection of capabilities grouped together within a particular functional context. The service contains the logic required to carry out these capabilities and provides a service contract describing which are available for invocation. The advertisement and discovery of services is a key principle of SO and an integral part of the service-oriented architecture model. In this model, service providers publish descriptions of their services to a registry. The registry then advertises these services to consumers. Service consumers typically use a standard service discovery protocol such as Universal Description, Discovery, and Integration (UDDI) to locate services in the registry.

From a consumer perspective, the process of identifying and specifying services can be either top down or bottom up. A top-down approach involves identifying an organization’s broad functional domains and then identifying high-level candidate services associated with those domains. This approach lets developers understand the system’s business context and establish the service-oriented system boundaries. In a bottom-up approach, the developer typically begins with a use-case analysis of small system components or the business process. In this approach, the developer determines an initial list of services only after conducting a significant use-case analysis for common functional scenarios and subsystem identification. This is usually the case with large projects at a functional domain or department, or at the business unit level. In practice, both approaches are often used simultaneously.

In addition, service ontologies can automate the advertisement and discovery of services. An ontology lets consumers and providers share a common set of terms for describing service qualities and constraints. Also, service descriptions include a service-level agreement (SLA) that concerns the terms and conditions of service provision and use—that is, what consumers can expect from a provider and restrictions on what consumers can demand from that provider.

Comparison. Capability features (services, operations, and nonfunctional characteristics) are, thus, similar to services in an SO. However, features identify a product line’s commonality and variability and configure reusable assets, whereas services identify a collection of functionalities along with an SLA of providers and specify an ontology for automated service advertisement and discovery. Hence, on the one hand, dynamic and automated feature binding considering the features’ quality attributes is basically missing in SPL. On the other hand, product line variations are difficult to capture explicitly using the notion of services in SO.

Dynamic Characteristics of a Service-Based System

Service-based systems are distributed and composed of various services that can be discovered and replaced at runtime. This dynamic characteristic of SO is closely related to QoS and dynamic-service orchestration.

Quality of service. QoS has traditionally been as-
Most SPLE approaches focus on configuring product line variations before deployment and don’t consider dynamic-service composition.

Involvement of Third-Party Service Providers
SPLE promotes systematic reuse within an organization and usually doesn’t consider external organizations when developing reusable assets. Moreover, relying on third-party providers and promoting the use of their services was out of scope for SPLE. The closest thing to third-party involvement that SPLE considers might be the use of commercial off-the-shelf (COTS) components. In SO, however, third-party involvement is one of the main drivers that makes this approach attractive, and it leads to several initiatives, including service negotiations, service monitoring, and service reputation systems.

Service negotiations. Negotiation refers to a communication process that supports coordination and cooperation. In terms of a software-as-a-service model, negotiation involves the interaction between a service consumer and one or more service providers identified through discovery, or providers that are already known to the service consumer. Negotiation with service providers can lead to SLAs for services that better meet consumer requirements.

Service monitoring. Service monitoring is another process for detecting service failures and SLA violations at runtime. This is an increasingly important research issue, as continually more companies conduct business over the Internet. Service monitors can determine whether services meet the terms and conditions of the SLAs between service consumers and providers. An SLA contains specific guarantees for the QoS that a consumer expects a provider to supply. SLAs must be monitored and audited for service provider compliance to provide real QoS guarantees to the consumer. The SLA could also contain conditions of use that the service provider imposes on the consumer, and service monitors could ensure compliance with these conditions as well. Emergent system qualities can result from the service composition process, and changes in the runtime environment. These emergent qualities require a dynamic runtime quality assurance approach that service monitors can facilitate.

Service reputation systems. In a service-oriented marketplace, transactions often occur between parties that haven’t previously interacted. Reputation systems are collaborative mechanisms that address trust issues between such parties, and they help distinguish between low- and high-quality service providers. Including provider reputation in the service selection criteria benefits the quality assurance process.
**QoS framework.** Traditional SPLE approaches don’t consider these three key aspects of dynamic-service provision, but SOPL methods should incorporate them. Therefore, we propose a QoS-aware framework that provides automated runtime support for service discovery, negotiation, monitoring, and service provider rating. QoS awareness lets consumers handle recovery from SLA violations, service failures, and runtime environment limitations by renegotiating and substituting problematic services.

**Variation Control and Management**

In SO, given the emergence of a service marketplace where multiple providers supply functionally equivalent services that implement a common service type, nonfunctional QoS properties are the criteria for distinguishing one provider from another. Service providers, therefore, need a standard method for describing the nonfunctional characteristics of the services they offer. Several recently proposed ontology-based initiatives can reduce the ambiguity of describing nonfunctional attributes and enable better service selection.10

However, in SPLE, it’s also important to explicitly tailor a product’s configuration to consumer needs. For example, if a specification stipulates that a product configuration must not include a service feature, the associated service shouldn’t be bound to that product even though the service providers are available at runtime.

Our approach adapts the C2 architecture style, which provides flexibility through a layered structure and modular components, called bricks.6 In this approach, we establish an explicit mapping relation between features and architectural components (bricks) so that selecting the features for a product generates a corresponding product configuration. After configuring a product architecture, the developer integrates reusable components into each brick by following the specifications described in the components, such as selecting precoded components, filling in skeletons, or instantiating parameterized templates.

**Our Feature-Based Approach**

Our approach provides a possible solution for developing core assets in a SOPL.6,7 This approach overcomes the challenges we discussed earlier by

- identifying appropriate dynamic services,
- adapting feature analysis to supplement service identification techniques,
- adapting service orientation to enhance the runtime flexibility of product lines, and
- incorporating variation points into services to more explicitly control product line configurations.

Figure 1 shows the main activities in this approach.

**Feature Analysis**

By using both feature and service analysis, we can reconcile the different notions of first-class objects. Feature analysis identifies externally visible characteristics of the products in a product line in terms of features and organizes them into a feature model. This model captures a product line’s commonality and variability by using different types of features: mandatory, optional, and alternative. Figure 2 shows a feature of a VO product line.

**Service Analysis and Specification**

Service analysis provides a mechanism for mapping the feature model onto services (addressing the dynamic characteristics of service-based system challenges). This activity classifies features into one of two categories: workflow services and dynamic services. Workflow services define service transactions (behaviors), whereas dynamic services are used to execute workflow services at runtime. The next activities (see Figure 1) specify and develop the workflow services, as well as the identified dynamic services and their QoS levels. A workflow service’s behavior can’t be changed at runtime, but the foremost concern is the correctness of its behavior when it uses dynamic services.

The availability of dynamic-service features shared by workflow services depends on the ser-
that the service provides. The quality description schema allows for expressing arbitrary qualities in terms of constraints, measurement units, values, and types. The strategy schema enables the expression of strategies, which describe ideal services and service compositions. Strategies also specify acceptable limits on the values of service- and operation-level quality constraints.\(^7\)

**System Integration and Deployment**

In the C2 architecture style, a brick can have its own thread, and it can send and receive messages to and from other bricks through its top and bottom ports. We’ve extended the C2 style to include two different types of bricks: workflow and dynamic service. Workflow bricks deploy orchestrating services, whereas dynamic-service bricks deploy dynamic services. For dynamic services, it’s possible to deploy either a real service or a proxy to an external service as a brick. Additionally, a configurator manages dynamic reconfiguration of the deployed product at runtime.

The runtime system interacts with service providers through an automated negotiation broker. This broker incorporates a consumer strategy with pluggable QoS negotiation, a QoS ontology, and SLA evaluation, as well as a provider-rating system to ensure service acceptability.

Our proposed QoS-aware framework facilitates system integration and deployment. Through this framework, dynamic-service consumers and providers supply the broker with templates specifying strategies for the SOPL services they require or provide. For consumers, the strategy describes the ideal QoS requirements of the functional services they wish to use. The negotiation broker incorporates an engine builder, which uses the templates to assemble a custom-service broker engine for processing negotiation messages and service proposals. The proposal engine creates and evaluates service proposals, and matches a consumer with a service provider by considering the QoS requirements.

The QoS framework also provides a service-monitoring system, which actively monitors the quality of negotiated services for emergent changes, SLA violations, and failures. The framework’s primary monitoring approach is a passive model, which transparently intercepts service requests and responses between service consumers and providers. This monitor is always aware of the QoS requirements and triggers a new negotiation whenever the SLA is violated at runtime. Finally, the framework includes a reputation system that provides consumers with a method of shar-

---

**Figure 2. A feature model of a virtual office.**
We're exploring better ways to tailor the service granularity of an SOPL to enhance reusability. We also plan to incorporate consumer context monitoring to improve quality assurance.

References


About the Authors

Jaejoon Lee is a lecturer in the School of Computing and Communications at Lancaster University. His research interests include software product line engineering, software architecture, and service-based software engineering. Lee has a PhD in computer science and engineering from Pohang University of Science and Technology. Contact him at j.lee@comp.lancs.ac.uk.

Gerald Kotonya is a senior lecturer in the School of Computing and Communications at Lancaster University. His research interests include software architecture, and service-oriented and component-based software engineering, especially novel ways of architecting, visualizing, and evolving self-managing, hybrid, service-oriented systems. Kotonya has a PhD in computer science from Lancaster University. He’s a chartered engineer and a member of the IEEE Computer Society. Contact him at gerald@comp.lancs.ac.uk.