# Chapter 18
# Variability in Web Services

**Matthias Galster and Paris Avgeriou**

***What you will learn in this chapter***
- *Why service-based systems need special treatment of variability*
- *What types of variability can exist in service-based systems*
- *How we can address variability in service-based systems*

## 1 Introduction

Service-based computing and associated development paradigms, including service-oriented architecture (SOA), web services, or the idea of "Software as a Service," have gained significant attention in software engineering industry and research. The aim of this chapter is to provide an introduction to *variability* in service-based systems. Within this chapter, we use the term "service-based" for systems that are largely or entirely built from web services [4], with SOA as the primary architectural style.

To briefly illustrate variability in service-based systems, let us consider the example of an online travel agency that communicates with various external businesses (such as airlines, hotel companies, rental car companies) to obtain airfares, hotel prices, etc., and to make reservations. Web services provide a standardized way to exchange information between the online travel agency and the information systems of these external businesses. In case a web service of an

M. Galster (✉)
Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand
e-mail: matthias.galster@canterbury.ac.nz

P. Avgeriou
Department of Computer Science, University of Groningen, Groningen, The Netherlands
e-mail: paris@cs.rug.nl

airline becomes unavailable, a web service of a different airline that offers the desired flight can be used. In this example, variability could be expressed in a *variation point* for selecting an airline web service. *Variants* (or options of alternative services) in this case would be the different airline web services.

In this chapter, we first provide a brief introduction of service-based systems. Then, we motivate and illustrate variability in service-based systems in a real-world example from the e-government domain and argue why variability handling is useful. We will discuss why service-based systems need special treatment with regard to variability and explore how variability in service-based systems could be addressed.

## 2 Service-Based Systems

Service-based systems are distributed systems that are built from loosely coupled software services. Services are autonomous, platform-independent computational elements that can be described, published, discovered, orchestrated, and programmed using standard protocols for building interoperating applications [1]. Services are developed independent from a particular technology, for example, could be implemented in Java or .NET, as long as they comply with standards and protocols. For example, in local e-government, a municipality can use services offered by the central government, such as a citizen registry, or services to support the processing of taxes. In service-based systems, a service registry enables service consumers to discover, bind, and assemble available services, often at runtime. A service infrastructure (such as an Enterprise Service Bus) connects services to service consumers. Service consumers query the registry and compose applications using a service or a composition of services. Consequently, service-based systems facilitate interoperability and reuse within and across different systems.

Individual services usually correspond to business functions and provide functionality to a large number of anonymous users (end users or other software artifacts), often distributed across organizations [5]. Thus, service-based systems support flexible environments and infrastructures for adaptable business processes. However, the reusability of individual services and service-based systems is determined by their ability to support the variability required to adjust them to different contexts. This means, if services or service-based systems cannot adapt to changing situations and contexts, they can only be reused in a very limited scope. Therefore, enhancing variability in service-based systems and providing methods that help explicitly model and manage variability facilitates highly reusable and configurable services and service-based systems.

## 3 Variability in Service-Based Systems

Before discussing variability in service-based systems, we present an industrial case from the e-government domain to motivate and illustrate variability in service-based systems: the implementation of national laws in local municipalities in the Netherlands. The national government may approve a law, which then is implemented in municipalities. In the Netherlands, there are more than 400 municipalities; each municipality would implement the law including processes and software systems that help implement the law autonomously. Differences between municipalities are too big to have solutions as one product for all municipalities, yet not too big to be covered by one generic solution to cover possible variants.

A concrete example for this phenomenon is the implementation of the Dutch Law for Social Support (known as the Wet Maatschappelijke Ondersteuning—WMO law). This law mandates rules for providing social support to citizens, such as domestic care. The responsibility and the execution of the WMO law lie with the municipalities. This means, even though the law has been approved by the Dutch national government, the solutions chosen to implement this law differ substantially between municipalities. Variability must cope with static variability (originating from differences between municipalities) but also dynamic variability (i.e., changes to the WMO law once a solution is deployed) and the evolution of the system in its environment. Throughout Dutch e-government initiatives, the "Software as a Service" (or SaaS) model is used. As a result, software providers offer solutions for the WMO law as software services, in a municipality-independent way. To cover the needs of as many municipalities as possible, the SaaS must be customizable to fulfill variations in business processes, functionality, and quality requirements of municipalities.

As discussed in the previous section, service-based systems provide some degree of flexibility by definition. However, it is difficult to build generic service-based systems that can be adapted in different organizations and changing situations. Even the eight fundamental design principles of service-orientation do not consider variability as a key issue when designing service-based systems [6]. Thus, there is a need for handling variability in service-based systems [12] for the following reasons:

- It helps meet Quality of Service and optimize quality attributes. When a currently deployed service does not perform adequately, it can be replaced by a better performing service. This means, configurations of service-based systems can be changed. In e-government, there are multiple vendors for the same software service. Due to the regulated nature of e-government, these services have to provide the same functionality but may differ in terms of reliability or performance. Based on these differences, services can be selected.
- It can enhance the availability of the system. When a service becomes unavailable, an alternative service with the same functionality can be used.

- It allows for runtime flexibility. This means, rebinding of services can be performed at runtime, potentially automatically when needed.
- It allows to handle different instances of one service-based system in different organizations and versions and to adjust the system to operate in diverse environments. This means, in Dutch e-government, the same system could be used in multiple municipalities.
- Individual services are usually not designed with variability in mind to make services highly customizable. By handling variability, artifacts in service-based systems (such as specifications and models) can be designed with variability for planned and enforced reuse [15].
- Related to a more technical aspect, if variability is not handled systematically and parts of a service-based system are adapted in an uncontrolled manner, interoperability problems occur [10]: Other parts of the system affected by the adaptation might not be adjusted properly. For example, in the case of the WMO law, many external parties (such as health care providers, doctors) are involved in completing a business process. If a service is changed in an unsystematic manner, it might not work with these external parties anymore.

## *3.1 Why Service-Based Systems Need Special Variability Treatment*

Service-based computing includes its own design paradigm and design principles, design patterns, a distinct architectural model (SOA), technologies, and frameworks [6]. Thus, in this section, we explore why service-based systems are different compared to traditional reuse-based paradigms (such as product line engineering, component-based development, or object orientation) and therefore need special "treatment" with regard to variability.

- *Dynamic execution environment*: The most significant difference to other reuse-based paradigms is that the dynamic execution environment of service-based systems allows changing systems at runtime. This means, services can be replaced or reconfigured while the system is running [18]. Product lines, for example, focus on compile-time support. Consequently, to fully support variability in service-based systems, events that occur in such systems must be linked to rules to reason about alternatives [7]. This is particularly true in the context of a volatile, distributed service composition in which services can change, fail, become temporarily unavailable, or disappear.
- *Different levels of abstraction*: Service-based systems usually comprise different levels, i.e., a business process level, the architecture level, and a service level with the actual implementation. Each of these three levels might again comprise different levels or descriptions (e.g., the architecture level usually consists of different architecture views and/or layers). The alignment of business and IT is a key concern in service-based systems, more than in other domains. This means,

variability must be traceable through all these levels and variation points in one level need to be translated into variation points in other levels to ensure business/ IT alignment. This also means that variability in service-based systems occurs at different levels of abstraction. For example, variability might be provided through parameter values used to invoke a service (service level), or by replacing complete services (architecture level), or by changing the sequence in a workflow (business process level).

- *Nature of individual services*: Another difference lies in the nature of individual services. Services as computational units must accommodate the challenge of meeting requirements for each organization that might use them while crossing boundaries between organizations. This means, when handling variability in service-based systems, there is no centralized authority that handles variability concerns in the individual parts of the system. Also, a high heterogeneity in customer requirements occurs due to anonymous service users. As a result, the range of possible variations between services and service-based systems might be very broad and extremely difficult to anticipate.
- *Organizational issues*: Organizational differences occur as services and service-based systems are no longer developed, integrated, and released in a centrally synchronized way [14]. Instead, services are developed and deployed independently and separately in a networked environment. Developers need to consider the integration of services, third-party applications, organization-specific systems, and legacy systems. Thus, rather than self-contained and isolated software development, service-based systems extend towards enterprise level collaborative exchange of services and components over networks. Therefore (and similar as stated in the previous paragraph), coordinating variability concerns is problematic during service-based development.
- *Quality attributes*: Quality attributes (e.g., performance, maintainability, security, reliability) in service-based systems are more diverse than in other domains and difficult to achieve [8].

## 3.2 Types of Variability in Service-Based Systems

As mentioned in the previous section, there are different reasons why service-based systems require special treatment with regard to variability. This also means that there are particular types of variability in service-based systems that may not occur in other types of systems. In this section, we therefore discuss types of variability that exist in service-bases systems. However, rather than providing a complete taxonomy, we provide an overview of basic types as well as where variability might occur in service-based systems. Please also note that we do not discuss variability based on technological aspects. A good overview in this regard has been presented by Robak and Franczyk [16]. From a high-level perspective, we differentiate two categories of variability:

1. Variability inside a service, with services as reusable units that can be adapted for different contexts [2]. In the e-government case, one example of variability inside a service is a difference in quality requirements of municipalities with regard to response time.
2. Variability in the service-based architecture (i.e., the composition of services). In the e-government case, examples for variability in the service-based system are different situations in which a service for assessing the need for a wheelchair through a government authority is called, or is not called, based on local regulations in the municipality.

Going into more detail of the first category, variability inside a service, the following types of variability can be found:

- Variability in parameters required by a service [20]: The type of data sent at service invocation can vary. For example, data sent to a service might be a single variable or an array of variables. This type of variability is usually expressed in Web Service Definition Language (WSDL) documents.
- Variability in parameter values [11]: The value of a parameter used at invocation might vary. For example, the age requirement for a wheelchair subsidized by a municipality differs between municipalities.
- Variability in the protocols [20]: Different protocols might be used by clients to communicate with services.

On the other hand, more detailed types of the second category, i.e. variability in a service-based architecture, include the following:

- Logic variability [2, 20]: A service includes operations for providing a certain functionality. The logic is the algorithm or logical procedure used in the operations of the service. A service can provide different logics depending on the requested functionality.
- Variability in the web service flow [2, 17]: A web service flow is a composition of service using a process-based approach. It specifies sets of tasks which are executed by the participants of a process. Additionally, a web service flow defines the execution order of tasks, the data exchange among the participants, and business rules. Web service flow variability expresses that services can be alternatively or optionally executed in a workflow, in different orders. This type of variability is described in Business Process Execution Language (BPEL) specifications for business process models and service flows. A detailed discussion of variability in a web service flow, which also considers technical and implementation aspects (e.g., message exchange), can be found in [17]. The service flow has two abstractions: the service and the related business process. Services are modeled as sets of operations, while the business process defines a flow of activities. Each activity is implemented by executing an operation on a service. Variation points affect the description of the flow needed to perform a set of ordered operations (see variability model in [11]).
- Composition variability [2]: The business process consists of several services to fulfill end user needs. For one service in the workflow, there may be more than

one possible service interface which implements the service with different implementation logics or quality attributes. Variability occurs in selecting the most appropriate service.

- Variability in quality attributes: Quality attributes might vary from one system to another. For example, one municipality might have higher privacy or performance requirements than another one. This type of variability might be specified in Web Service Level Agreement (WSLA) specifications for web service level agreements between the service consumer and the service provider [11]. This type of variability is most difficult to handle and even a major challenge in software product lines.

## 4 How to Address Variability in Service-Based Systems

We consider three major strategies for addressing variability in service-based systems. First, as currently argued by many authors, we could adopt product line approaches in the service domain [13]. Referring to the types of variability from the previous section, this strategy can be applied for handling variability in the composition of a service-based system if services are treated as features.

Second, as product lines focus on feature and decision models, we can apply new methods and concepts—beyond the product line domain. This strategy has not yet been thoroughly explored. Examples for this strategy include modeling variability from a pattern point of view [20]. Here, pattern approaches can be used to describe variation points. Again, referring to variability types from the previous section, this strategy could be applied for variability in quality attributes.

Third, we can combine existing methods from the product line domain and concepts based on the specific requirements of service-based systems. For example, to support variability at the architecture level of service-based systems, new viewpoints for the product line architecture could be introduced. This strategy could be applied to variability in BPEL, WSDL, or WSLA specifications; variability in parameters requirements by a service and actual parameters; variability in protocols and logic; as well as variability in the web service flow.

In the remainder of this section, we first list principles for how to address variability. Then, we discuss handling variability by utilizing product line paradigms together with traditional web service development (third strategy) as the strategy that is followed most. The other two strategies are omitted due to lack of space.

### *4.1 General Principles*

The four following principles, as general guidelines, can be used to address variability in service-based systems [3]:

1. Recognize commonalities and variations across the scope of multiple service-based products within and across an organization. In the e-government example, commonalities and variations occur due to differences in business processes, local regulations, and infrastructures.
2. Leverage the recognized commonalities by building "core assets" that exist in all product variants, including services across product variants with established points of variation. In the e-government example, core asset services are services that are required in all municipalities, such as billing citizens for obtaining a wheelchair.
3. Address enterprise integration needs that service-based systems must offer. Integration must take variability into consideration. Integration could encompass many systems and involves sharing services across systems. For example, in municipalities, existing legacy systems or third-party software (such as enterprise resource planning systems or data base systems) must be integrated. Legacy systems and third-party software occur in all municipalities but vary between municipalities.
4. Address end-user needs for variation within the service-based system. For example, municipalities can select services as needed to accommodate unique workflows.

## *4.2 Product Lines and Web Service Development*

As mentioned at the beginning of this section, using the product line paradigm to address variability in service-based systems has been the most popular strategy so far [19] as service-based systems and product lines share certain commonalities [3]. A technical comparison between the variability in product lines and variability in service-oriented computing is given by Chang and Kim [2].

Recently, Sun et al. proposed a framework and a tool suite for modeling and managing variability of web service-based systems [18]. This framework addresses runtime and design-time variability and is an extension of COVAMOF. COVAMOF was originally developed to manage variability in software product lines. Sun et al. use UML diagrams and a specifically developed UML profile to model variability. This work builds on VxBPEL an extension of BPEL to support variability in web-based systems [12]. VxBPEL has extra XML elements to support the expression of variation points and variants in a BPEL process. An example of a VxBPEL fragment to code a variation point is shown in Fig. 18.1 (adapted from [12]). Variation points can be placed inside a BPEL process at any place where a single activity can be placed.

VxBPEL supports service replacement, different service parameters, and changing system composition. Compared to Sun et al., VxBPEL variability is only modeled in the implementation layer rather than at higher levels of abstraction. Sun et al. make full use of COVAMOF to model variability also at the architectural level to help understand the composition of a service-based system.

**Fig. 18.1** VxBPEL fragment to code a variation point (adapted from [12])

```
<vxbpel:VariationPoint name="VP1">
  <vxbpel:Variants>
    <vxbpel:Variant name="default">
      <vxbpel:VPBpelCode>
        <invoke .../>
      </vxbpel:VPBpelCode>
    </vxbpel:Variant>
    <vxbpel:Variant name="alternative1">
      <vxbpel:VPBpelCode>
        <sequence ...>
          ...
        </sequence>
      </vxbpel:VPBpelCode>
    </vxbpel:Variant>
  </vxbpel:Variants>
</vxbpel:VariationPoint>
```

In the example of WMO implementation, the VxBPEL fragment above could contain a `VariationPoint name="PersonalBudget"`. This variation point expresses variability in the way the personal budget of a citizen who applies for societal support is determined. Variants for this variation point are `Variant name="Inhouse"` and `Variant name="External"`. Based on the selected variant, either a service for in-house processing of the personal budget is invoked or the request for budgeting is sent to an external provider.

## 5 Outlook

Handling variability in service-intensive systems enables systems which are highly adaptable and reusable in different environments. Handling variability helps construct systems that do not only reuse services but can be reused as a whole. In this chapter, we highlighted the need for and benefit of variability handling in service-based systems. Moreover, types of variability as it might occur in service-intensive systems were discussed as well as how variability could be addressed.

To further leverage variability in service-oriented systems and web services, new tools would need to be created that support the concepts discussed in this chapter. Moreover, to facilitate dynamic service variability, "self-adaptive" and "plug-and-play" architectures can be investigated. Recently, utilizing dynamic product lines in a service-oriented context has been explored [9].

## References

1. Asadi, M., Mohabbati, B., Kaviani, N., Gasevic, D., Boskovic, M., Hatala, M.: Model-driven development of families of service-oriented architectures. In: First International Workshop on Feature-Oriented Software Development, Denver, CO, pp. 95–102. ACM (2009)
2. Chang, S.H., Kim, S.D.: A variability modeling method for adaptable service in service-oriented computing. In: 11th International Software Product Line Conference, Kyoto, Japan, pp. 261–268. IEEE Computer Society (2007)
3. Cohen, S., Krut, R.: Managing Variation in Services in a Software Product Line Context. CMU SEI, Pittsburgh, PA (2010)
4. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the web service web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Comput. **6**(2), 86–93 (2002)
5. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, NJ (2005)
6. Erl, T.: SOA Design Patterns. Prentice Hall, Upper Saddle River, NJ (2009)
7. Gannod, G.C., Burge, J.E., Urban, S.D.: Issues in the design of flexible and dynamic service-oriented systems. In: International Workshop on Systems Development in SOA Environments, Minneapolis, MN, pp. 118–123. IEEE Computer Society (2007)
8. Gu, Q., Lago, P.: Exploring service-oriented system engineering challenges: a systematic literature review. Serv. Orient. Comput. Appl. **3**(3), 171–188 (2009)
9. Hallsteinsen, S., Jiang, S., Sanders, R.: Dynamic software product lines in service-oriented computing. In: 3rd International Workshop on Dynamic Software Product Lines, San Francisco, CA, pp. 28–34 (2009)
10. Jiang, J., Ruokonen, A., Systa, T.: Pattern-based variability management in web service development. In: Third European Conference on Web Services, Vaxi, Sweden, pp. 83–94. IEEE Computer Society (2005)
11. Kim, Y., Doh, K.: Adaptable web services modeling using variability analysis. In: Third International Conference on Convergence and Hybrid Information Technology, Busan, Korea, pp. 700–705. IEEE Computer Society (2008)
12. Koning, M., Sun, C., Sinnema, M., Avgeriou, P.: VxBPEL: supporting variability for web services in BPEL. Inf. Softw. Technol. **51**(2), 258–269 (2009)
13. Lee, J., Kotonya, G.: Combining service-orientation with product line engineering. IEEE Softw. **27**(3), 35–41 (2010)
14. Lee, J., Muthig, D., Naab, M.: An approach for developing service oriented product lines. In: 12th International Software Product Line Conference, Limerick, Ireland, pp. 275–284. IEEE Computer Society (2008)
15. Medeiros, F.M., de Almeida, E.S., de Lemos Meira, S.R.: Towards an approach for service-oriented product line architectures. In: Workshop on Service-oriented Architectures and Software Product Lines, San Francisco, CA, pp. 1–7. Software Engineering Institute (2009)
16. Robak, S., Franczyk, B.: 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, pp. 120–128. Springer (2003)
17. Segura, S., Benavides, D., Ruiz-Cortes, A., Trinidad, P.: A taxonomy of variability in web service flows. In: First Workshop on Service-oriented Architectures and Product Lines, Kyoto, Japan, pp. 1–5. SEI (2007)
18. Sun, C., Rossing, R., Sinnema, M., Bulanov, P., Aiello, M.: Modeling and managing the variability of web-service-based systems. J. Syst. Softw. **83**(3), 502–516 (2010)
19. ter Beek, M., Gnesi, S., Fantechi, A., Zavattaro, G.: Modelling variability, evolvability, and adaptability in service computing. In: First International Workshop on Automated Configurations and Tailoring of Applications, Antwerp, Belgium, pp. 14–19. CEUR (2010)
20. Topaloglu, Y., Capilla, R.: Modeling the variability of web services from a pattern point of view. In: European Conference on Web Services, Erfurt, Germany, pp. 128–138. Springer (2004)