Journal of Universal Computer Science, vol. 19, no. 2 (2013), 233-253
submitted: 7/5/12, accepted: 31/12/12, appeared: 28/1/13 © J.UCS

Domain-Oriented Customization of Service Platforms: Combining Product Line Engineering and Service-Oriented Computing

Klaus Schmid, Holger Eichelberger, Christian Kröher
(University of Hildesheim, Hildesheim, Germany
{schmid, eichelberger, kroeher}@sse.uni-hildesheim.de)

Abstract: Service-Oriented Computing (SoC) has been established as an important paradigm over the last decade. A particularly important part in a service-oriented solution is the service-oriented platform. This provides an environment and infrastructure for a number of service-oriented applications. An important challenge in complex application areas is the need to customize these platforms to the demands of a specific context. Product line technologies can support this by providing the concept of variability management to SoC. In this paper, we will provide a reference model for (domain-specific) service platforms and describe different approaches that provide customization possibilities in a service platform context. The complexity of handling the customization of large-scale service platforms in an integrated manner will be addressed by introducing the concept of production strategies for variability implementation techniques.

Key Words: domain-oriented, variability, customization, service, platform, software product line
Categories: D.2.3, D.2.13

1 Introduction

Traditionally, service-oriented computing aimed at the development of individual applications. More recently, especially in the context of cloud-computing, the focus broadened to complete and integrated platforms for service-oriented systems. Service platforms in the sense that we will discuss them here provide environments and infrastructures to support a number of related service-oriented applications. This growing interest in service platforms is partially driven by the increasing importance of cloud-computing, but still the main focus for service platforms is actually in internal scenarios, where platforms are built or customized to suit the needs of a single customer – or a small group of customers. Today domain-oriented customization is particularly relevant to these internal scenarios.

In this section, we will introduce the problems that arise from the customization of large-scale service platforms. These problems will be discussed in [Section 1.1]. In [Section 1.2], we will introduce a running example that we will use throughout this article to illustrate problems and solutions to the customization of service platforms.

1.1 Customization of Service Platforms – The Problem

A customization of service platforms is always relevant, if the needs of potential customers may differ too widely, excessive resource consumption may be an issue (removing unnecessary parts may reduce resource consumption significantly), or safety and reliability issues play a role (parts that could be removed cannot fail). This situation can be found, in particular, in cloud environments, where a (domain-specific) platform might be needed to be offered to customers with different requirements. This can be supported by adapting the service platform. Typically, such a customization affects several different aspects of a service platform ranging from the modification of platform service implementation to the adaptation of business processes. Further, customized platforms may differ in infrastructure service interfaces, the offered service programming model and technical aspects like protocols and techniques (SOAP [W3C 07] vs. REST [Fielding 00]), or supported business process engines.

Customization techniques for software have been thoroughly researched in the context of Product Line Engineering (PLE) [Clements 07]. Here, variability management provides the basis for the systematic customization of software products. However, so far comparatively little of this knowledge has been applied and adapted to the context of SoC, as we found in a related survey [Indenica 11a]. We will also discuss that most of these approaches are specific to an individual programming model or service technology. Moreover, the existing approaches do only address individual layers of a service platform. An integrated approach for customizing domain-specific service platforms is needed to overcome these short-comings.

This situation has been the basis for the INDENICA-project[1] on customization and integration of service platforms. This project includes industrial partners like Siemens or SAP who face significant demands for the customization of their domain-specific service platforms. This provided the context for the work presented here.

In this article, we will focus on the demand for the customization of service platforms and its ramifications from a variability management point of view. This includes also customizations in cloud environments as this can be interpreted as a special form of service platform customization. However, the main focus is on the customization of service platforms in general. Further, it should be noted that we exclude adaptivity in the sense of autonomous customization from our discussion. Rather, we focus only on customization in the sense of deriving a specialized platform, based on external decisions. Moreover, this decision will usually only be made once and not reverted at a later point.

We will start in [Section 2] with developing a reference model for *domain-specific service platforms*, which will guide us through the further discussion of service platform customization. In [Section 3], we will discuss the current state of variability management techniques that are used for the customization of parts of service platforms. In [Section 4], we will discuss the ramifications the current approaches have for further research in the context of variability management techniques and introduce the concept of production strategies as an integrated approach to service platform customization. Finally, in [Section 5] we will conclude.

[1] INDENICA-project, funded by European Union: www.indenica.eu

1.2 Running Example

In this section, we introduce a running example, which we will use throughout this article. The example focuses on a flexible yard management system (YMS) and is motivated by one of the industrial use cases in the INDENICA-project. A yard refers here to the place in front of large warehouses where trucks arrive and are loaded. A YMS is responsible for the efficient coordination of shipment processes on a yard. Yards differ in size, complexity, yard topology, and degree of automation so that each installation needs to be customized individually.

We will now discuss different aspects of variability in a YMS. Customization of this platform may either affect a single aspect or multiple aspects simultaneously. Please note that this list only contains aspects relevant to this article. A complete description of the YMS use case can be found in [Indenica 11b].

1.2.1 Variability in the YMS Technical Infrastructure (Layer)

A service platform, as we will discuss in Section 2, contains a technical infrastructure and domain-specific services. A technical infrastructure of a YMS may provide infrastructure services like persistence, connectivity, and authentication. Each of them may be customized for a specific installation:

- *Persistence*: Different databases might be supported. For each of them it might be necessary to handle specific characteristics.
- *Connectivity*: Connections to the platform services may be available via remote procedure call (RPC), SOAP-based services, or REST.
- *Authentication*: The YMS does not provide authentication. As a result, for a specific installation a local customization is required.

1.2.2 Variability in the YMS Services

The services provided by a YMS may differ in terms of their presence in a specific installation and their provided functionality. This depends on the requirements of a specific yard:

- *Presence*: Yard workers typically communicate with the YMS via a desktop user interface at fixed terminals. Optionally, mobile communication services enable yard workers to interact with the YMS using mobile devices.
- *Functionality*: A yard jockey service is responsible for scheduling yard jockeys who fetch trailers from the parking lot and transport them to a specific dock. The yard jockey service is mandatory, but the detailed characteristics of the scheduling depend on the mechanism that locates the jockeys on the yard. Examples are manual state-based tracking, which relies on terminals at specific locations, or continuous tracking via GPS. In an installation exactly one of these will be present.

The above examples only illustrate some variability; in the real world YMS have significantly more variability. The combination of the YMS technical infrastructure and the YMS services form a domain-specific service platform that can be customized to fit the needs of a specific customer (yard). We will use these examples to illustrate important points in the remainder of this article.

2 A Reference Model for Domain-Specific Service Platforms

In this section, we introduce the term *domain-specific service platforms* and discuss a reference model for them. Broadly speaking, a service platform is the basic infrastructure for service-oriented applications. It includes all those parts of the system, including services, which are not specific to an individual application. This is a deliberately broad definition, as we will discuss below. But before we turn to this, we will start with discussing the concept of platform, service platform and domain-specific service platform in a more precise manner.

- We use the term *platform* to denote a combination of infrastructure assets such as storage capabilities, communication middleware, operating systems and other general purpose software infrastructure like web servers, etc. This may include arbitrary parts of software (not necessarily service-oriented), like the persistence of the YMS technical infrastructure in [Section 1.2].
- We use the term *service platform* to denote any platform, which follows the principles of service-orientation. In particular, this means that the platform offers explicit services, which can be accessed by applications or also by other service platforms like an authentication service in the YMS example.
- A *domain-specific service platform* is a platform, which particularly supports the demands of a specific application domain. Thus, it does not only support a single application, nor is it generic in the sense that it supports nearly arbitrary applications. For example, the services offered by the YMS are specific to the domain of yard management systems but are generic in the sense that they can be further tailored to fit to the needs of a specific yard.

In order to describe (domain-specific) service platforms in a rather general form, we developed a reference model, which we will discuss in detail now.

2.1.1 Layers of the Reference Model

The reference model for domain-specific service platforms as shown in [Figure 1] consists of two layers. The lowest layer is the *technical platform*, which provides the basic technical capabilities for a service-oriented system. This includes generic, domain-independent services like a typical middleware does. On top of this is the *(domain-specific) service* layer. This contains the actual domain-specific capabilities in the form of specialized services or complete, domain-specific business processes. Both, technical platform layer and service platform layer together provide the required functionalities for *application* (service) deployment and execution. The domain-specific platform layer is involved in this as it may provide specifically adapted functionality for deployment handling, monitoring, reconfiguration and so forth. We will now discuss each individual layer of the reference model including possible customizations in each layer. [Figure 1] also shows for illustration applications, which are built on top of the domain-specific service platforms. However, this is outside the scope of our discussion.

2.2 Technical Platform Layer

The technical platform is the basic level for service provisioning. It consists of the service platform infrastructure, which hosts technical platform services. Examples for

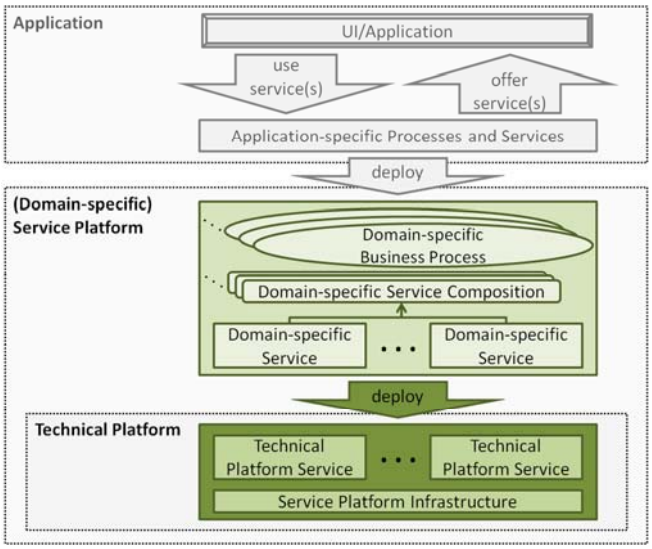

Figure 1: Reference model for domain-specific service platforms

technical platforms are the Service Component Architecture (SCA [OASIS 12]), as e.g., realized by Apache Tuscany [Tuscany 11] and Fabric3 [Fabric3 11], the OSGi [OSGi 12] platforms Eclipse Equinox [Equinox 12] or Apache Felix [Felix 11], or the .NET platform Microsoft AppFabric [Microsoft 12]. Of course, many more exist. We will now discuss both parts of the technical platform in detail:

- The *Service Platform Infrastructure* is the basic platform implementation, which cannot be further refined into specific services. The service platform infrastructure can be realized in an arbitrary way, in particular, in a non-service-oriented fashion. Thus, customization techniques developed for traditional PLE such as those discussed in [Muthig 02] [Svahnberg 05] can be applied here. In the YMS example, customization can address certain quality attributes like real-time capabilities or scaling.
- *Technical Platform Services* are provided by the service platform infrastructure and enable functionality like the registration of services or access to other infrastructure capabilities. These services may also be subject to customization, e.g., regarding the exact range of offered services or their exact behavior. Examples could be user authentication mechanisms or persistency in the YMS technical infrastructure (cf. [Section 1.2]).

2.3 Domain-specific Service Platform Layer

This layer is what makes a domain-specific service platform truly domain-specific by introducing services and processes that address the need of a specific domain (e.g., for yard management systems). A domain-specific service platform combines the capabilities of a technical platform with domain-specific capabilities.

Our reference model in [Figure 1] refines the domain-specific service layer into:

- *Domain-specific Services* realize domain-specific functionality by utilizing capabilities of the underlying technical service platform. Customization on this level may include any variability where a service is modified, augmented by additional functionality, or otherwise adapted. For example, the scheduling algorithm of the yard jockey service in the YMS may be customized to fit a specific yard topology.
- *Domain-specific Service Compositions* implement higher level services as aggregations of domain-specific services. Here, customizations can be performed in terms of whether the compositions are available or not or by changing the specific set of composed services, respectively the form of their integration. In the YMS example, the mobile communication service may be available or not. Further, different mechanisms (WLAN/UMTS) or devices may be used.
- *Domain-specific Business Processes* use domain-specific services and service compositions as specific activities during a workflow execution. This is a particular form of service composition: A business process aggregates services, service compositions, and (maybe) other business processes, thus it can be considered a service composition, which aggregates services. Thus, also the same forms of customization apply. However, as a service composition is typically interpreted as a higher-level service and a business process typically represents a set of activities on a different level of abstraction, we clearly separate both elements.

Finally, the domain-specific services as well as the entire platform (including the technical platform) need to be *deployed*. Here, customization may impact the deployment process. This covers any form of variability that influences the specific deployment of a service (e.g., not deploying individual services, the specific location of deployment, determination of deployment parameters, etc.).

2.4 Application Layer

The application layer in our model (cf. [Figure 1]) subsumes all parts of a specific application (UI-elements, application-specific services, etc.) that is deployed on and, thus, uses the domain-specific service platform. This layer is described only for the sake of completeness. We will not further discuss application-layer customization; first, as we want to focus on domain-specific service platforms, second, as the constituents of the application layer – and thus the customization possibilities – are very similar to the domain-specific service platform layer.

3 Customization Techniques for Domain-Specific Service-Platforms

In this section, we will discuss the customization of domain-specific service platforms. In particular, we will discuss a set of available techniques for each of the elements of the reference model introduced in [Section 2], which will lead us to a set of short-comings with respect to domain-specific service platform customization. The identified short-comings form the basis for the introduction of production strategies as the next step beyond individual customization techniques in [Section 4].

The basis for the discussion is a literature study that we performed in order to systematically survey existing customization techniques for SoC and categorize them according to the elements of the reference model. Due to space limitation, we will not detail the literature study here. However, a description can be found in [Eichelberger 12]. Further, we selected a representative set of techniques for each layer of the reference model to illustrate the range of available techniques and their individual characteristics. While more techniques exist, we cannot discuss all of them in detail in this article due to space limitations. A more detailed description and discussion of the techniques identified in the survey can be found in [Indenica 11a]. This also includes further aspects like supported forms of variation (e.g., optional, alternatives, etc.), binding times or the technology background. These aspects will also be subject to the discussion of the short-comings in [Section 3.3].

3.1 Technical Platform Layer

As discussed in [Section 2], the service platform infrastructure is basically a non-service-oriented part of software. Thus, it can be customized with classical variability techniques like those surveyed in [Muthig 02] [Svahnberg 05]. Therefore, we will not discuss this further in this article. This leads us to the technical platform services as the main part of interest here.

3.1.1 Technical Platform Services

The technical platform services are those capabilities of the technical platform that can be identified as providing distinguishable services. These services provide infrastructure capabilities for other services or the environment as a whole. Typical examples for this are registration and identification of services, monitoring capabilities, event handling, logging, or higher level capabilities like the authentication service of the YMS example introduced in [Section 1.2].

Customization of these services can be achieved by a large number of different approaches like those identified in our survey [Indenica 11a]. Out of those we want to illustrate the overall potential by discussing a few examples. These are:

- The use of existing management functionality of the service platform,
- Modification of interceptor chains,
- Aspect-oriented composition.

A basic approach to customize a technical platform service is to rely on the *management functionality or configuration settings***,** which are already built into the platform. Liu uses in [Liu 11] configuration settings to customize virtual cloud appliances and the technical platform, e.g., a database or a virtual machine. The configuration settings specified in terms of configuration files are then interpreted by the existing infrastructure as part of normal operation. In the YMS example, the required interface of the authentication service may be specified in terms of a configuration parameter at installation time in order to establish a connection to the authentication system in place. The specific parameter value is then read by the YMS to load the corresponding interface at start-up time. This is described in [Svahnberg 05] as a (traditional) variability management technique: “condition on variable”.

A further approach to modify technical platform services is called *interceptor chains*. This allows adding, replacing, or removing existing management services by

modifying explicit call chains provided by the technical platform. Instead of calling a callee directly from a caller, the call is represented as a call object and passed through a chain of so-called interceptors. Each interceptor may work with the data, modify, consume or reissue the call object. Froihofer et al. describe in [Froihofer 07] runtime variability of platform management services by adding, replacing or removing related interceptors. For each type of call the platform must provide appropriate interceptor chains as a basis for customization. Custom interceptors may handle calls in different, domain-specific ways. In this approach, customization is performed by adding, replacing or removing interceptors (even at runtime). For example, in some YMS installations it might be important to trigger communication with a third-party-system when receiving a service call. In such an installation, a specific interceptor may be added that triggers the communication by reading the service call. However, interceptor-based customization works only, if the platform already provides support for interceptor chains.

Typically, aspect-oriented techniques (AOT) add variabilities to a small core platform [Muthig 02]. When using *aspect-oriented composition* for customization, variabilities are represented as aspects and bound to the core platform using an aspect weaver, i.e., a tool, which injects the selected variabilities into the core platform at defined variation points. In the YMS example, this might be used to integrate specific capabilities like a specific database connection, for a single installation into the platform. The aspect weaver must be available at the point in time when the variability is bound, which is typically at compile time, but possibly also at runtime. This approach can be combined with enhanced methods like explicit variability models to ensure proper handling of dependencies and conflicts [Walraven 08] [Walraven 10].

3.2 (Domain-Specific) Service Platform Layer

Certain forms of customization techniques can also be used on the service platform layer. The specific techniques that apply vary for the different elements of the layer. We will start with the discussion of the customization of domain-specific services. Then, we will move a level higher and discuss service composition and business processes. Finally, we will turn to deployment. Influencing deployment provides a different way of customization.

3.2.1 Domain-Specific Services

Domain-specific services are technically comparable to other forms of services. They address the concerns of a specific (application) domain. The main difference to other kinds of application-specific services is that they are part of a platform that may be reused across a number of different applications of the same domain.

Customization techniques for domain-specific services may only modify the implementation or they may as well have an impact on the service interface. Changing the implementation may impact the behavior of the service, like the scheduling mechanism of the yard jockey service in the YMS (cf. [Section 1.2]), and/or its Quality of Service (QoS) characteristics. In case an interface customization is performed, typically also a modification of the implementation is necessary. Thus, the later can be seen as a superset of the former. As pure implementation

customization is typically simpler, several approaches only address implementation modification for a constant interface. We will first discuss customization techniques that only impact the service implementation and in a second step we will enlarge the focus to techniques that also perform a customization of the service interfaces.

Most of the techniques that can be used to *customize the service implementation* apply at implementation or compile time. They typically rely on standard variability implementation techniques from classical software engineering and extend it by taking into account the specific structure of service implementations (e.g., separate, explicitly described service interfaces). One specific example for customizing service implementations is to introduce a component layer within the service implementation and to realize variability by exchanging individual components at compile time (e.g., the different scheduling mechanisms of the yard jockey service in the YMS) [Medeiros 09]. As these components are only constituents of the service implementation, this approach is not able to modify the interfaces. Similar restrictions apply for approaches like conditional compilation techniques.[2]

The customization of service interfaces requires an approach that is able to modify the interfaces and the implementation in a systematic way. Generative techniques [Narendra 08] or aspect-oriented techniques like those used in SAP NetWeaver [SAP 05] or for cloud service customization in [Jegadeesan 09] support both, the customization of service implementations and the customization of service interfaces in an integrated manner. In the YMS example, this allows customizing the scheduling mechanism of the yard jockey service as well as the corresponding input parameters consistently (cf. [Section 1.2]). These techniques are typically effective at implementation or compile time. However, these approaches directly manipulate the service interface (e.g., by removing or adding a specific parameter), but they do not provide guidance to ensure that the interface modifications and the service implementation match. As the modifications are also local to the services, there is no support to ensure that not only the call interface is customized, but also the callee. Some approaches also support service variability at (initialization time or) runtime. One specific example is the Aspect Service Weaver (ASW) pattern described by Monfort et al. [Monfort 09]. It relies on the aspect service weaver tool [Tomaz 06] to intercept service calls (SOAP messages) and if a message includes a request for a method that the service does not support currently, advice services are woven in. These advice services hold the implementation of new methods, like scheduling mechanisms for the yard jockey service that can be woven into existing services by the ASW. A similar approach is given by Merle et al. in [Merle 11]. Their reflective SCA framework FRASCATI provides an adaptive multi-cloud platform. Possible variants are discovered and enacted at runtime via reflective techniques. Thus, Merle et al. leverage a technique for the customization of technical platform services to the level of domain-specific services.

---

[2] To be more precise: conditional compilation as practiced in languages like Java (i.e., using `if` with constant conditions) is not capable of interface adaptations. In case a more powerful approach like the C-preprocessor is used, also interface adaptation is possible. However, this approach gives rise to different problems as it is not capable to assure that interface customization and implementation customization are aligned.

3.2.2 Domain-Specific Service Compositions and Processes

While the notion of service composition and processes are overlapping, there are also fundamental differences: a business process provides a form of service composition, but not all service compositions are processes. Thus, we will discuss service composition and processes here together. First, we discuss a technique to modify general service combinations, then a technique which applies to business processes.

Li et al. present in [Li 10] a technique for modifying service compositions at runtime to meet given QoS requirements. In this *QoS-based service composition* approach, each Web service is annotated with QoS attributes, e.g., response time or cost. Given a set of available services with their QoS attributes and a new QoS constraint, the presented algorithms calculate the QoS of the overall system with regard to the general service composition. As long as the QoS of the system does not meet the new QoS constraint, one or more services are replaced to meet the constraint. In the YMS example, the mobile connection service of the mobile communication service may be annotated with an attribute for connection signal strength. If the signal strength is below a specific average rate (specified by a constraint), the mobile connection service will be replaced, e.g., from WLAN-connection to UMTS-connection service.

Moon et al. present an approach called Business Process Family Model (BPFM) for customizing processes [Moon 08]. BPFM is a *variability-enhanced business process model* expressed as a Unified Modeling Language (UML) [OMG 11] activity diagram augmented with variation points, variation point bindings and variant regions. The variabilities in the BPFM are resolved in terms of a configuration. For example, depending on the type of goods and the size of the yard it might be necessary to customize the yard management. This configuration is used as input to a transformation mechanism, which produces a specific process instance in Business Process Execution Language (BPEL) [OASIS 11]. Further, conceptually similar approaches exist that augment and extend existing process and activity modeling notations for capturing variability [Puhlman 06] [Rosemann 07] [Hallerbach 10]. For the cloud context, van der Aalst advocates in [vanDerAalst 10] the standardization and application of configurable process models.

3.2.3 Deployment of Domain-Specific Platforms

A further approach to customization is the modification of the deployment process and even the artifacts at this point of time. The deployment of services has a profound impact on the final service platform. It impacts not only physical layout but also the logical content of a platform as it may change which services actually get deployed. It is, however, a rather crude instrument to influence the details of functionality. Deployment may influence many characteristics of the running platform at deployment time and even at runtime. In literature, variability of the deployment is usually realized by generative techniques. We describe here two techniques relying on generation of deployment artifacts like models, descriptors or scripts.

In the first technique, the variability model is annotated by deployment information for each variant. Mietzner et al. describe in [Mietzner 09] that WS-BPEL deployment and undeployment scripts for deployment time as well as for runtime can be generated from additional information in the variability model. In particular, the

authors derive single or multi-tenant SaaS applications from a service-oriented core application. Therefore, they deploy the technical platform, the application and individual services, which are either capable of single- or multi-tenant processing to a cloud environment.

The second technique relies on generic artifacts, which contain all possible variabilities (e.g., all possible scheduling mechanisms of the yard jockey service in the YMS). The core artifact is a generic deployment plan, which describes the deployment of each possible variant including the initial deployment. Ayed and Berbers extend in [Ayed 07] the standardized CORBA Component Model (CCM) deployment model [OMG 06] by variation points in order to express a context-aware deployment plan. At runtime, the generic plan is instantiated by generative techniques to a concrete plan based on context information. In the YMS example, such context information may be the availability of GPS-sensors. If these sensors are available, a tracking-aware scheduling mechanism is deployed, while otherwise the mechanism will be manual state-based tracking.

3.3 Discussion

In this section, we discuss the capabilities of the customization techniques for domain-specific service platforms from a more general perspective. As stated in the introduction of [Section 3], we will also refer to material that can be found in [Indenica 11a] here, including the relevant summaries for the discussion.

In the preceding sections, we discussed the different layers of our reference model for domain-specific service platforms and provided for each layer a representative set of customization techniques to illustrate the range of available approaches. In a more detailed analysis [Indenica 11a] we found the following shortcomings:

SC1: Most techniques we found only support a single layer element as basis for variability out of those described above. For example, the techniques discussed in [Section 3.2] can only be applied to domain-specific services, service composition and processes, or service deployment respectively. Only few techniques support two or even three types of those layer elements, like aspect-oriented composition (cf. [Section 3.1]), which can be applied to technical services as well as domain-specific services. Thus, supporting full customization of a domain-specific service platform requires the combination and integration of multiple techniques. However, to our very knowledge, such an approach does not exist.

SC2: Most of the techniques only support a single form of variation (we did not discuss forms of variation here in detail due to space restrictions, but discussed this in more detail in [Indenica 11a]). For example, the QoS-based service compositions approach in [Section 3.2] only supports alternative variation, while other forms like multiple selection, etc. are not supported. In particular, only few techniques explicitly support an extension, i.e., the introduction of a variability, which is initially not known, like in the ASW approach (cf. [Section 3.2]). However, supporting all forms of variation is mandatory to cope with any form of variation arising in real-world scenarios.

SC3: Only very few techniques explicitly support multiple binding times, e.g., runtime and compile time, like the aspect-oriented composition approach in

[Section 3.1]. In particular, this is often initialization and runtime as these only differ when the mechanism is called, e.g., a runtime mechanism which is used only during start-up can be used for binding at initialization time. However, supporting multiple binding times is mandatory to enable service platform vendors offering additional customization options of the same platform to their customers. For example, platforms for low-end customers will be completely configured by the vendor at compile time, while high-end customers will be able to configure their platform (or parts of it) at runtime.

If one reviews the above considerations, major problems for domain-specific service platforms become obvious: such a platform is a very complex and large-scale combination of services, components, etc. Thus, if we want to provide meaningful customizations, they also need to be large-scale and will probably consist of a large number of individual customizations. These customizations must be carefully orchestrated to achieve the desired outcome and to avoid introducing defects in the customization process. It is very hard to imagine how this can be achieved by a set of individual techniques which are rather unrelated. We will approach these problems by introducing the concept of production strategies as a step beyond individual customization techniques in the next section.

4 From Customization Techniques to Production Strategies

In the previous section, we discussed a representative set of existing state of the art techniques that support the customization of parts of our reference model for domain-specific service platforms (cf. [Section 2]). Based on this discussion and the results from [Indenica 11a], we identified a set of short-comings of the existing state of the art techniques (cf. [Section 3.3]).

In this section, we will introduce a model that addresses these problems by providing a general characterization of variability implementation: *production strategies*. A production strategy enables a meaningful customization of domain-specific service platforms in terms of orchestrating multiple customization steps. An individual step in turn provides for similar customizations as the techniques discussed in [Section 3]. Thus, our focus is not to develop yet another set of specialized variability mechanisms. We rather work based on the assumption that all these techniques have their specific role and make sense in their individual context. What is missing is rather a more unified approach that will provide integrated and coherent capabilities to the management of variability. Individual approaches could then be refined to form instances of this approach. This model provides a refinement of earlier work like [Schmid 04].

4.1 The Production Strategy Model

In this section, we will introduce the *production strategy model* as the basic concept of our integrated approach to the management of variability. The purpose of this section is to discuss the basic approach of production strategies in general, before we go into details in [Section 4.2]. Thus, we will discuss the parts of the production strategy model and illustrate the basic concept by two different, but simple examples.

A *production strategy* defines how customizations are applied, i.e., how variant parts must be assembled for a certain variability resolution. This may involve arbitrary conditions over variation points as a basis for driving the instantiation of the variability. Thus, a production strategy takes several partial source models (or in general, parts of artifacts) of some kind and realizes a selection at a specific variation point in a target model as illustrated in [Figure 2]. As shown there, the introduction of the selected variant element(s) can be combined with the generation of additional glue code (or another form of integration). Before we define the concept of a production strategy more precisely, let's first look at two examples:

- An example of a production strategy is realized by the `#ifdef/#if`-construct in conditional compilation in the C language along with the preprocessor that interprets it. While this may seem like a strange approach in the context of a service platform, it is a realistic technique for an embedded service platform or technical layer, e.g., to adapt the specific behavior of a technical platform service. We also refer to this here, mainly because it is simple and widely known, not because we particularly recommend this technology. In this case the variant elements $elem_1$,.., $elem_n$ are the parts guarded by the `#ifdef`, `#else`, and `#endif` constructs. The *selector* (the specification of how variant sub-elements are defined and combined) is exactly formed by the constituents of the if-clause, while the *variation point* is provided by the position where this construct is placed within the source code. The selector also provides the necessary capabilities to determine which of the variant elements to integrate into the final artifact. In the running example, the different connection types to the platform services (RPC, SOAP, or REST) may be implemented within if-clauses, which enables a selection over the available types in a specific installation (cf. [Section 1.2]). In this case no *glue*-code is needed, because the glue is already defined in terms of source code. Here, only the selected variant element becomes part of the target model, which is of the same type as the source model (C language source code). However, glue could, for example, be integrated by means of macros or an additional generator.[3]
- A second example is provided by a technique where services can be replaced at runtime within a composition, while the interface remains the same. This is actually straightforward in a service environment, like in the YMS example where each scheduling mechanism for the yard jockey service may be provided by an individual service implementation. The selector is given by the mechanism that relates a service reference to a service implementation. The exact approach is different for different service technologies, but it always relies on a service registry. However, this leads to the need for a rather intelligent and flexible service registry, which supports complex decision making. Such a service registry would need to integrate variability management technology, which is currently not the case for any registry we know of. The position of integration for the service is given by

[3] It can be regarded as a short-coming of preprocessor-based variability implementation that multiple selections cannot be well supported. This would require a more powerful technique. However, our basic discussion is not invalidated by this observation, rather it emphasizes the need for flexibility in the usage and combination of variability implementation techniques.

the service reference in the composition. No glue generation is required as only an individual service is impacted by the variability in this example.

The two examples above already show that (a) the model of a production strategy is very generic and (b) this allows describing a large range of different techniques in a unified manner. Thus, this model provides for a common description of the techniques discussed in [Section 3] enabling an integrated specification of customization options for a domain-specific service platform. However, before we will discuss this in more detail, we will have to refine the model in order to precisely define those parts needed to specify any kind of variability (customization) in terms of production strategies.

4.2 Refinement of the Production Strategy Model

Any production strategy consists of certain parts in order to realize a variability. These parts can be easily identified, if we look at the *basic concept of a production strategy* as described in [Section 4.1] and analyze what a production strategy must achieve. A production strategy attaches to a certain point in a model (also source code is seen as a model here), and instantiates the model by selecting some variant elements ($elem_1$,.., $elem_n$ in [Section 4.1]) that are related to this *variation point* via a *selector*. This selection is based on decision values that are used to define a specific product. Further, a production strategy combines the selected variant elements in some form (if necessary) and binds the result to the variation point.

The combination of variant elements is necessary, e.g., if we select multiple values. In this case some *glue* must be produced in order to determine how to combine the individual variant elements appropriately. This may, in particular, imply code generation to provide runtime selection of an appropriate element. However, the notion of glue is meant here more generic. It can basically be any kind of additional model elements, required in order to embed the selection result. The details of what is selected depend on the chosen variation. Thus, in order to describe a production strategy, we need the following parts:

- **Definition and evaluation of a variability value:** A way of mapping decision values into specific resolutions that include the variant elements and include them appropriately into the target model. Using the example of the C-preprocessor, `#ifdef` directly takes an argument. This value must be provided by the environment, e.g., in terms of a string value, which identifies the selected connection type for a specific YMS installation by name. In the case of a selection of a service in a service composition at runtime, this would be a mechanism that could be realized in the service registry and would enable to direct the actual service call to the selected implementation. For example, if GPS is temporarily not available, the service registry of the YMS platform redirect the service calls to the implementation, which uses state-based tracking for scheduling (cf. [Section 1.2]).
- **Variation point identification:** This describes where the variation occurs, in particular, where the result of the production and selection process is placed. In the examples, this is given by the textual context of an `#ifdef` statement in the source code. In the case of a service composition it might be a BPEL activity, which references the service. However, many different means exist

for identifying this. This will always depend on the available technologies and the relevant artifact types.

- **Technique for selecting (and combining) variant elements:** This defines how variant sub-elements ($elem_1$,.., $elem_n$ in [Section 4.1]) are identified and combined. For example, this defines the identification and possible combination of the various *alternative* elements of a selector like the implementations of the connection types and the service implementations of the scheduling mechanisms in the YMS example. A combination is necessary in case of a multiple selection as the various selected sub-elements must be integrated in some way. This is also subsumed as glue in [Figure 2].
- **Technique for introducing selected variant elements (including relevant glue)**: After selecting and combining the final variant elements, it might be necessary to use some sort of binding mechanism to relate the combined parts to the target model (at the position denoted by the variability point identification). Again the specific means are provided by the specific technique within the general framework. In the C-preprocessor case, this is given automatically as the selected code is integrated at the same position it had already previously. In the service example, this is given by the service call, which is redirected appropriately by means of the registry. If aspect-oriented techniques are used, where the modification is described external, but the result needs to be physically integrated, this relation is established by the aspect preprocessor.

The above parts form the basis for describing variabilities realized by different techniques, like those discussed in [Section 3]. For example, these parts allow describing variabilities in the technical infrastructure of the YMS example using interceptor chains (cf. [Section 3.1]) as well as variabilities in the domain-specific services of the YMS platform implemented via aspect-oriented techniques (cf. [Section 3.2]). While we cannot discuss a detailed mapping between the discussed techniques and the concept of production strategies due to space limitations, we would like to point out the following properties of the production strategy model, which enable the support of many different techniques:

- The model only abstractly identifies the various parts that must be integrated. Each specific technique may realize these parts in a very different manner.
- The definition of a common reference description provides the basis for a common integrated implementation.
- Having such a reference description also allows standardizing certain parts without affecting others (e.g., variability description and dependency management). This allows to integrate the various techniques described in [Section 3] in a single customization approach.

4.2.1 Formal Definition of Production Strategies

In order to more formally – and thus, precisely – define what a production strategy is, we need to look at its parts and characteristics discussed above. These include: the type of target model (e.g., C-code), the binding time (e.g., preprocessing), and the kind of selection supported. Nevertheless, a production strategy is still generic. It still provides a very general mechanism that can be applied to a very large number of

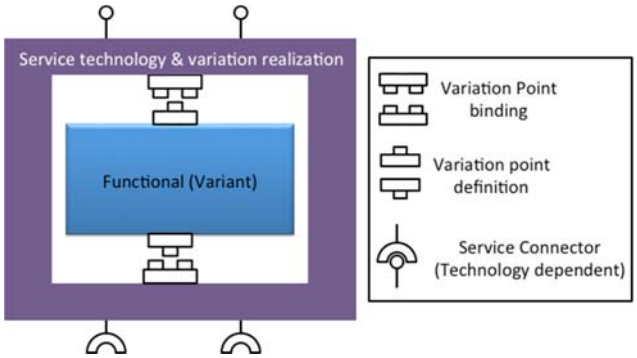

Figure 3: Separation of functional code and variability implementation

variabilities in a product line. In summary, we can interpret an individual production strategy as a transformation function of the form:

$$PS(target, vip, val, elem_1, \ldots, elem_n, sel, selType, bt) \rightarrow model$$

Here, *target* provides the target model where a variability must be resolved. The specific point in the model where the results of the variability resolution shall be inserted is also called a *variability impact point* (*vip*) [Schmid 03]. *Val* defines the value of the decision for a specific product. Further, $elem_1$ to $elem_n$ are the variant elements among which a selection occurs, i.e., the arguments to the variability selector *sel*. Finally, the selection type *selType* specifies the type of variation (optional, alternative, etc.), while the binding time *bt* defines when this selection has to be done. This can be used to more formally describe the generation of instantiated platforms. However, a complete formalization still requires much further work.

4.3 Application of the Production Strategy Model

The production strategy model provides a generic abstraction of variability implementation that does not only enable the integrated application of a multitude of different, individual techniques, as it is required for the customization of domain-specific service platforms; rather, it also enables abstracting from an individual service implementation to the degree that this can be completely separated from the actual variability realization as shown in [Figure 3]. This enables the flexible exchange of variability techniques and hence modifying secondary qualities of a variability like its binding time [Schmid 08b]. As the production strategy model in this sense is less a variability technique, but rather a way of unifying (existing) variability techniques, we can see it as a "meta-technique". Moving to this higher level of abstraction brings with it a number of benefits and helps to address the shortcomings outlined above. We describe this here using the example of flexible binding times, which we described in detail in [Schmid 08a] [Schmid 08b].

The selection of a specific variability implementation technique typically fixes the binding time, i.e., whether the selection of a specific customization is done at development time (e.g., compile time), initialization time, or runtime. However, this is sometimes undesirable and it is preferable that the binding of variabilities can also

be flexible in their binding time. For example, for one platform customization it might be known that it is necessary to be able to switch at runtime among different variations of service behavior, while for another customization of the same platform the decision on the specifics of available service behaviors is more appropriate at development time (e.g., due to resource constraints that forbid to make all possibilities available at runtime).

Examples of approaches that support such variability of the binding time are timeline variability [Dolstra 03], anytime variability [vanDerHoeck 04], or meta-variability [Schmid 08a][Schmid 08b]. These approaches support a dynamic shift of the binding time. Timeline variability and anytime variability are very restrictive, as they consider only a specific set of variability realizations. However, our initial work on meta-variability already followed the basic concept of production strategies (although we were not aware of this at this time). Thus, this work demonstrated the capabilities for exchanging the variability implementation technique, which derive from such an approach.

The core idea of this approach is exactly to abstract from the details of the individual variability technique and move to a more generic model (product strategy model). In particular, the variability is in this case no longer part of the basic implementation (cf. [Figure 3]), but it is rather described separately in the form of a production strategy. The specific way of realizing production strategies that was chosen in [Schmid 08a] was by encapsulating them in the form of aspect templates. These templates only capture the general form of the variability implementation technique and are instantiated for any specific variability for which they must be applied. Then the instantiated aspect is woven into the basic implementation to actually implement the variability. As nearly arbitrary variability implementation techniques can be combined with this approach, this provides a generic framework for realizing variability and can be used to exchange variability techniques in a flexible manner. In particular, we showed in [Schmid 08a] that it is possible to use this approach to exchange variability techniques, without impacting the functional implementation, while achieving flexibility between compile time, initialization time, and runtime binding.

4.4 Benefits of Production Strategies

The concept of production strategies supports the customization of domain-specific service platforms in an integrated manner as it allows to homogenously describe the variability of the different layers, while realizing the variability of the different parts in a technology-specific manner. The production strategies address the short-comings of the state-of-the-art customization techniques, which we identified in the course of the discussion in [Section 3]:

- *Support for different layer elements as a basis for variability (SC1):* In general, any layer element can be subject to a production strategy by defining corresponding variation points. Thus, the model supports all kinds of artifacts like text, code, XML, etc., as long as there is a technique that enables variability in these elements. In this sense it is also an extension of the approach described in [Schmid 04].
- *Support for different forms of variation (SC2):* A variation point accepts arbitrary combinations of variants. As we are rather open with respect to the

specific process of glue generation, this can be adapted to the specific kind of artifacts, the specific form of combination, etc. If there is a restriction, than it is currently with respect to handling explicit extensions, where the variants are not known. This is not yet well represented.

- *Support for multiple binding times (SC3):* We showed in Section [Section 4.3] that the production strategy model can be useful to support multiple binding times. This is not done per se by providing a specific multi-binding-time technique, but rather by providing a framework that makes variability techniques with different binding times interchangeable. We demonstrated this earlier already by supporting compile time, startup time and runtime.

5 Conclusion

In this article, we discussed the problem of the customization of service platforms, a problem, which is currently addressed in the EU-project INDENICA together with a number of industrial partners. As a basis for our discussion we developed a *reference model for domain-specific service platforms*. This model is structured into multiple layers. The lower layer is mostly technical, while the higher layer provides the domain-specific capabilities. This reference model can also be applied to cloud-based solutions like Business-by-Design [SAP 12]. In addition to the layers, the model also identifies individual elements that are subject to the customization. This includes the technical infrastructure, the technical services, the domain-specific services, the service compositions and processes as well as service deployment.

The customization of such a service platform is particularly hard as we need to perform customization in a coherent way to ensure that a customized system is again a valid system. Our analysis showed that the technologies that exist today to customize service platforms are extremely diverse, but each individual technique is only applicable to a very specific situation. This makes it at this point extremely hard, if not impossible, to perform meaningful, large-scale customizations.

In order to address this problem, we introduced the notion of *production strategies*. A production strategy captures the essence of a variability technique, while abstracting from the details. This allows standardizing certain non-essential aspects of techniques and, hence, enabling an integrated, coherent customization of systems and service platforms when using multiple, individual customization techniques. Thus, for example, variability modeling or dependency management among variabilities can be handled on this basis in an integrated manner. If the specific variability implementation technique is clearly separated from the functionality that it is supposed to implement, this also allows replacing individual variability implementation techniques with each other. We discussed this based on an example where different variability implementation techniques were used to realize the different binding times.

Finally, we discussed the possibility to extend this model and introduce even an abstraction from the specific service technology, which is used. We described this only briefly, as this is not a necessary ingredient of domain-specific service platform customization, but initial experiments in this direction are promising.

While several aspects of the model that was described in this article have already been validated, further work is still required. Among others, our group works on a sophisticated variability modeling language, which is powerful enough that it can be used to support all variability modeling concerns relevant to the customization of domain-specific service platforms. Another area of our current research focuses on the development of an integrated tool-set, which is able to perform the necessary variability implementations in a wide range of different circumstances. This combined solution is expected to be validated in the context of the INDENICA project with a number of industrial partners.

Acknowledgements

This work is partially supported by the INDENICA project, funded by the European Commission grant 257483, area Internet of Services, Software & Virtualisation (ICT-2009.1.2) in the 7th framework programme.

References

[Ayed 07] Ayed, A., Berbers, Y.: Dynamic Adaptation of CORBA Component-Based Applications. In *Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC '07)*, 580–585, 2007.

[Clements 07] Clements, P., Northrop, L. M.: Software Product Lines: Practices and Patterns; Addison Wesley (2007).

[Dolstra 03] Dolstra, E., Florijn, G., Visser, E.: Timeline Variability: The Variability of Binding Time of Variation. In J. van Gurp and J. Bosch, editors, Proceedings of the 1st International Workshop on Software Variability Management (SVM '03), 2003.

[Eichelberger 12] Eichelberger, H., Kröher, C., Schmid, K.: Variability in Service-Oriented Systems: An Analysis of Existing Approaches. In Liu, C., Ludwig, H., Toumani, F., and Yu, Q. (Eds.), Proceedings of the 10th International Conference on Service Oriented Comuting (ICSOC '12), pages 516-524. Springer, 2012.

[Equinox 12] Eclipse Equinox OSGi, 2012, http://www.eclipse.org/equinox/

[Fabric3 11] Fabric3 SCA Platform, 2011, http://www.fabric3.org

[Felix 11] Apache Felix, 2012, http://felix.apache.org/

[Fielding 00] Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, 2000.

[Froihofer 07] Froihofer, L., Goeschka, K. M., Osrael, J.: Middleware Support for Adaptive Dependability. In Proceedings of the 8th ACM/IFIP/USENIX International Conference on Middleware (Middleware '07), 308–327, 2007.

[Hallerbach 10] Hallerbach,A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. Journal of Software Maintenance and Evolution: Research and Practice – Best papers from the BPM 2008 Workshops, 22(6-7), 519-546, 2010.

[Indenica 11a] INDENICA project consortium, Deliverable D2.2.1: Variability Implementation Techniques for Platforms and Services, 2011, http://www.indenica.eu

[Indenica 11b] INDENICA project consortium, Deliverable D5.1: Description of Feasible Case Studies, 2011, http://www.indenica.eu

[Jegadeesan 09] Jegadeesan, H., Balasubramaniam, S., A Method to Support Variability of Enterprise Services on the Cloud, Proceedings of the 2009 IEEE International Conference on Cloud Computing, 117-124, 2009

[Li 10] Li, Y., Zhang, X., Yin, Y., Wu, J.: QoS-Driven Dynamic Reconfiguration of the SOA-Based Software. In *Proceedings of the 2010 International Conference on Service Sciences (ICSS '10)*, 99–104, 2010.

[Liu 11] Liu, H., Rapid Application Configuration in Amazon Cloud using Configurable Virtual Appliances, Proceedings of SAC'11, TaiChung, Taiwan, 2011

[Medeiros 09] Medeiros, F. M., de Almeida, E. S., Meira, S. R. L.: Towards an Approach for Service-Oriented Product Line Architectures. In Proceedings of the 3rd Workshop on Service-Oriented Architectures and Software Product Lines: Enhancing Variation (SOAPL '09), 2009.

[Merle 11] Merle, P., Rouvoy, R., Seinturier, L., A Reflective Platform for Highly Adaptive Multi-Cloud Systems, Proceedings of ARM'2011, Lisbon, Portugal, 14-21, 2011

[Microsoft 12] Microsoft Server and Cloud Platform, Windows Server 2008 AppFabric, 2012, http://www.microsoft.com/en-us/server-cloud/windows-server/appfabric.aspx

[Mietzner 09] Mietzner, R., Metzger, A., Leymann, F., Pohl, K.: Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications. In *Proceedings of the 2009 Workshop on Engineering Service Oriented Systems (ICSE '09)*, 18–25, 2009.

[Monfort 09] Monfort, V., Hammoudi, S.: Towards Adaptable SOA: Model Driven Development, Context and Aspect. In *Proceedings of the 7th International Joint Conference on Service-Oriented Computing (ICSOC-ServiceWave '09)*, 175–189, 2009.

[Moon 08] Moon, M., Hong, M., Yeom, K: Two-Level Variability Analysis for Business Process with Reusability and Extensibility. In *Proceedings of the 32nd Annual IEEE International Computer Software and Application Conference (COMPSAC '08)*, 263–270, 2008.

[Muthig 02] Muthig, D., Patzke, T.: Generic Implementation of Product Line Components. In *Proceedings of the 2002 Net.ObjectDays (NODE '02)*, 313–329, 2002.

[Narendra, 08] Narendra, N. C., Ponnalagu, K., Srivastava, B., Banavar, G. S.: Variation-Oriented Engineering (VOE): Enhancing Reusability of SOA-Based Solutions. In Proceedings of the 5th IEEE International Conference on Services Computing (SCC '08), 257–264, 2008.

[OASIS 11] Organization for the Advancement of Structured Information Standards (OASIS). OASIS Web Service Business Process Execution Language (WS-BPEL), 2011. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[OASIS 12] OASIS Open CSA, Service Component Architecture, 2012, http://oasis-opencsa.org/sca

[OMG 06] Object Management Group, Inc. (OMG). Deployment and configuration of component-based distributed applications specification (DEPL), 2006. http://www.omg.org/spec/DEPL/.

[OMG 11] Object Management Group, Inc. (OMG). Unified Modeling Language (UML) Resource Page, 2011. http://www.uml.org/.

[OSGi 12] OSGi Alliance, 2012, http://www.osgi.org/

[Puhlmann 06] Puhlmann, F., Schnieders, A., Weiland, J., Weske, M.: Variability mechanisms for process models. Technical report, BMBF-Project, 2006.

[Rosemann 07] Rosemann, M., van der Aalst, W.M.P.: A configurable reference modeling language. Information Systems 32(1), 1-23, 2007.

[SAP 05] SAP NetWeaver Switch Framework, 2005, http://help.sap.com/erp2005_ehp_04/helpdata/en/42/cbe37a560221e2e10000000a114cbd/content.htm

[SAP 12] SAP Business ByDesign, 2012, http://www.sap.com/solutions/sme/businessbydesign

[Schmid 03] Schmid, K.: A Quantitative Model of the Value of Architecture in Product Line Adoption. In *Proceedings of the 5th International Workshop on Product Family Engineering (PFE '03)*, 32–43, 2003.

[Schmid 08a] Schmid, K., Eichelberger, H.: EASy-Producer – A Product Line Production Environment. In Proceedings of the 12th International Software Product Line Conference (SPLC '08) – Demonstration and Poster Papers, 357–357, 2008.

[Schmid 08b] Schmid, K., Eichelberger, H.: Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects. In Proceedings of the 2nd International Workshop on Variability Modelling of Software-intensive Systems (VAMOS '08), 63–71, 2008.

[Schmid 04] Schmid, K., John I.: A Customizable Approach to Full Lifecycle Variability Management, Science of Computer Programming, Vol. 53, No. 3, 259-284, 2004.

[Svahnberg 05] Svahnberg, M., van Gurp, J., Bosch, J.: A Taxonomy of Variability Realization Techniques. Software – Practice and Experience, 35(8):705–754, 2005.

[Tomaz 06] Tomaz, R. F., Hmida, M. B., Monfort, V.: Concrete Solutions for Web Services Adaptability Using Policies and Aspects. International Journal of Cooperative Information Systems, 15(3):415–438, 2006.

[Tuscany 11] Apache Tuscany SCA Platform, 2011, http://tuscany.apache.org

[vanDerAalst 10] van der Aalst, W. M. P.: Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining, Proceedings of On the Move to Meaningful Internet Systems OTM 2010, 8-25, 2010

[vanDerHoeck 04] van der Hoek, A.: Design-Time Product Line Architectures for Any-Time Variability. Science of Computer Programming, Special Issue on Software Variability Management, 53(30):285–304, 2004.

[W3C 07] World Wide Web Consortium (W3C): SOAP Version1.2 Part 1: Messaging Framework (Second Edition), 2007. Online available at: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[Walraven 08] Walraven, S., Verbaeten, P., AO Middleware Supporting Variability and Dynamic Customization of Security Extensions in the ORB Layer. In Proceedings of the 9th International Conference on Middleware (Middleware '08), 121–123, 2008.

[Walraven 10] Walraven, S., Lagaisse, B., Truyen, E., Joosen, W.: Aspect-Based Variability Model for Cross-Organizational Features in Service Networks. In Proceedings of the 1st International Workshop on Composition: Objects, Aspects, Components, Services and Product Lines (Composition & Variability '10), 57–63, 2010.