Annales Univ. Sci. Budapest., Sect. Comp. 31 (2009) 23-41

TOWARDS A PROFILE-BASED APPROACH TO MANAGE SOA-TO-SOA INTEGRATION CHALLENGES

T. Systä and M. Hartikainen

(Tampere, Finland)

Abstract. Service-Oriented Architecture (SOA) has been rapidly and widely adopted in software companies and in IT sector in general. Migrating existing software systems to SOA is a challenge many companies are currently facing. The more SOA has been adopted and realized using various techniques, the more often a new challenge, namely, how to integrate two SOA-based systems, is encountered.

In this paper we discuss the challenges related to SOA-to-SOA integration, based on the experiences we have gained in a practical case study. These challenges can relate to e.g. service descriptions, communication mechanisms and service composition techniques. We propose an approach, relying on the use of UML profiles, as an aid for managing such integration projects.

1. Introduction

According to OASIS (the Organization for the Advancement of Structured Information Standards), Service-Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [14]. Service-orientation aims at loosely coupled services to support the requirements of business processes and users. Ideally, a client application, which itself may also act as a service, can, at run-time, search for services from a service registry based on a certain search criteria and then start communicating with the selected services. These main roles of SOA are depicted in Figure 1. Besides enabling flexible point-to-point communication, more

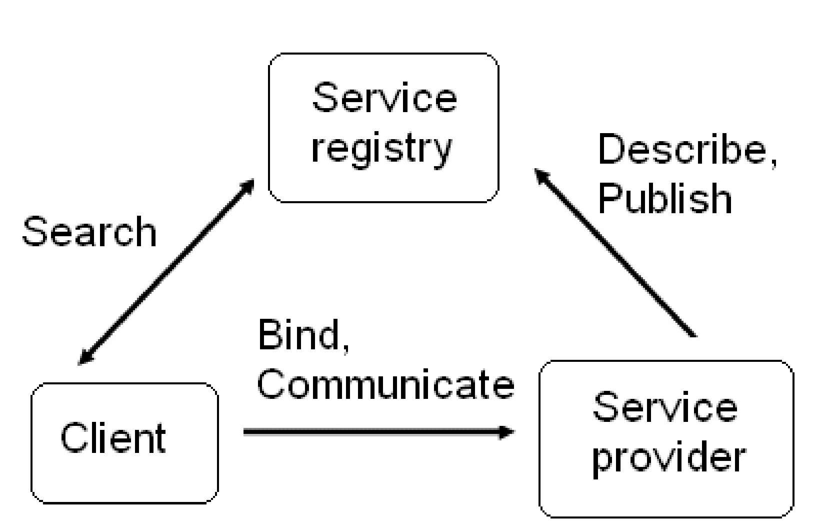

Figure 1. The main roles of SOA

complicated applications and business processes are aimed to be composed at run-time from the services available. SOA itself does not define the formats used in e.g. for communication or for describing and publishing services. They are decided by a particular *SOA realization*. In a Web services system, the language used for describing and locating services is Web Service Description Language (WSDL) [30, 31], and the communication among the clients, services, and a registry is handled with Simple Object Access Protocol (SOAP) [32]. SOAP, in turn, uses some transportation protocol, e.g. HTTP. Finally, Universal Description, Discovery and Integration (UDDI) [15], for instance, can be used as a Web service registry.

The interest towards adaptation of SOA in IT companies and organizations in general has grown rapidly. A common problem related to SOA adoption is how to migrate existing in-use software systems to SOA. Various wrapping approaches have, for instance, been proposed for that purpose [3, 27]. The idea is to wrap existing software interfaces and provide them as services for other, external software systems or components. Such approaches aim at minimal changes to the existing systems. After the set of services has been agreed on, realization techniques of SOA need to be decided. Commonly used Web service languages and techniques are often chosen for that purpose. These languages have their inadequacies, too. Web service standards, for instance, have limited support for communication security and for expressing service-level agreements. Thus, in some cases other, perhaps, in-house solutions, to realize SOA are chosen.

As SOA gets more generally adopted and it is realized using various different techniques and languages (and their different versions), the problem of how to integrate different SOA-based systems arises. Rebuilding one SOA realization to follow the languages and mechanisms used in another is not always an option. Instead, a loose integration might be desired, allowing the different SOA systems to remain unchanged and usable for their original purposes and in their original environments. To solve the integration problem, the software engineer needs to identify the similarities and differences, e.g. in terms of semantics and expressiveness, among the different languages and technologies used. In essence, mappings between different registry solutions, between different services descriptions and between different messaging mechanisms, at least, need to be defined. For instance, assume that a proprietary SOA realization uses a specific, perhaps company-defined format X for communication between clients and services, and that this services system is then intended to be integrated with a Web services system. Then the differences between SOAP and X in terms of what kind of support they provide for messaging need to be identified. If, for instance, X has support for message routing information, which is not supported by SOAP (except through extensions), a decision has to be made whether to ignore that information or implement it somehow on the Web services side.

The integration of the different languages, and possibly their different versions and/or extensions, used in SOA-based systems should not, however, be solved solely based on the syntaxes of those languages. The integration goals, namely how and for what purposes the integrated environment is intended to be used should drive and guide the technical integration decisions and activities. Also, the semantics of these languages should be taken into account. The importance of semantics and metadata has been acknowledged in various Internet-based information management systems, e.g. Semantic Web solutions [20]. The semantics should be given in a well-formed fashion to allow programmatical processing. The need and importance of semantics has also been emphasized e.g. in many model-driven software development (MDD) [18] techniques. They heavily rely on using (UML [17]) profiles and metamodels: they define the concepts in a subject domain and constraints related to them. The profile information can then be used e.g. for code generation [8, 12, 21] and for validation [16, 25]. Surprisingly enough, Web service development techniques, and SOA-based approaches in general, have had limited emphasis on semantics in practise. While the idea of semantic Web services is not new, practical applications are still rare.

In this paper we discuss the challenges related to SOA-to-SOA integration. We further propose a flexible approach to define common rules and requirements for the different languages and formats used for the same purposes (e.g. communication or service descriptions) in different SOA realizations. The approach relies on use of UML profiles. The stereotypes defined represent the relevant concepts that must be supported, one way or the other, in different systems.

In the case of messaging, they could be concepts related to operation requests, replies and error handling. In addition, the profile may include other rules, independent form capabilities or expressiveness of the languages, e.g. those related to the usage purposes of the integrated system. With proper tool support, they can be used to aid comparison and validation of the different languages used for the same purposes in different SOA realizations and can thus be used to manage SOA-to-SOA integration activities. This requires that the grammars are first transformed to UML representations [10] and then stereotyped according to the profiles. Then, UML model comparison operations [24] could be used to identify the corresponding parts in these grammars and in instance documents (e.g. two service descriptions given in different languages). Further, conformance checking operations [10] could be used to validate the instance documents against the language grammar and SOA rules.

2. Challenges in integrationing SOA-based systems

Integration of two SOA-based systems that are realized using different technologies can be quite challenging. One reason for this is the large variety of (possibly extensible) languages and techniques needed and used to realize SOA. That, in turn, implies that several transformations between different languages and formats are needed. Also the dynamic nature of services systems causes challenges; clients search and bind to services at run-time, requiring also dynamicity from the integrated system.

In this chapter we discuss various challenges related to SOA-to-SOA integrations. The points presented are based on our practical experiences, gained in a real world case study. In this case study we integrated a SOA-based Plenware PlugIP software and development platform with a Web services system. Plenware PlugIP uses proprietary formats for messaging and service descriptions. It further uses a registry that differs from UDDI in terms of formats used and functionality provided. The aspects discussed in what follows are by no means meant to be a covering analysis of the challenges, but are intended to give light on the complexity of the problem.

2.1. Variation points

Two SOA realizations can vary a lot in terms of standards followed, techniques used and functionalities they offer. At an abstract level, all parts depicted in Figure 1 can vary

- service registries,
- service implementations,
- service descriptions,
- client implementations,
- communication media used and
- middleware technologies used.

Registry solutions used in different SOA realizations may vary from traditional database solutions to more enhanced registry solutions. ebXML registry [13], for instance, allows not only storing service descriptions but also provides advanced categorization schemas and support for storing service profiles and business scenarios.

From the point of view of SOA-to-SOA integrations and use of services in the first place, variation in techniques and technologies used for service implementations is not very relevant. For calling the service, only the public interface of the service, given as a service description, is relevant. However, the variations in the implementations may influence different quality of service (QoS) attributes, such as response times. The service description formats used, in turn, may vary in terms of language used, expressiveness, bindings to messaging mechanisms, structure, extendibility etc.

Besides implementation techniques, client applications may vary in terms of their dynamicity. For instance, *static clients* know the services it uses already at compilation time, while *dynamic clients* search and bind to services at runtime. This naturally effects on performance, flexibility and maintainability. It also effects on what information is needed at the compilation time.

To enable communication among registries, services and clients, a mechanism depending on various technologies is needed. This implies that variation may occur as well at the transportation protocol level (http, ftp, smtp, etc.) as at the message format level.

Besides the technical and purely functional aspects, SOA realizations may vary concerning various utility services, such as security, QoS attributes, routing, service monitoring etc.

In what follows we discuss the SOA-to-SOA integration problem mainly from the point of view of communication mechanism, service descriptions and registries. This focus naturally addresses only on a small part of all the possible variation points, listing of which would be an impossible task. This focus was motivated by the practical integration project we carried out. Thus, an overview to this case study is presented first.

2.2. Integration viewpoints

Before considering the integration and transformation details, decisions have to be made at least on the following: what to transform and integrate, when the transformations should take place and by which party, and to what kinds of future extensions and changes should be prepared for.

Deciding what to transform greatly depends on what are the services and functionalities the systems to be integrated offer and especially on how and for what purposes the new system is intended to be used. The intended usages may significantly scope the integration and transformation needs. It also helps in selecting the most suitable integration solution from a large set of choices, which will be discussed in the sequel. The intended use also influences in the decision on when the transformations should be made and by which part. These decisions have a great impact on the future maintenance of the integrated system. For example, to enable communication between a client application in one SOA realization and a service in another one, it should be decided whether the client is responsible for sending a message in a form understandable by the service, the service should do the appropriate interpretations, or a "neutral party", e.g. a transformation gateway should be used.

It is unlikely that complete (in terms of coverage) transformations can be made between two SOA-based systems, since the different systems rarely support exactly the same features. The transformation becomes much easier if not all the features in one system need to be brought to another system as well. That, however, would mean "smallest common dominator" fashion transformations, which may significantly limit the capabilities of the integrated systems.

One option to integrate different realizations of SOA is to build a transformation gateway, where a set of translators are used. In this solution each translator acts as a converter between different formats. For example, a service description translator could be implemented for translating a service description given in one format, e.g. WSDL, into a service description given in another format, e.g. a proprietary service description language. Even in the case where two SOA realizations are to be integrated, this solution might get rather complicated. It may also cause performance overhead. We have observed this in our practical case study. Even with its complexities, this case is relatively simple, since (bidirectional) one-to-one transformations were able to be used. When more systems are involved, the number of converters quickly grows if the solution solely relies on one-to-one transformations. Then, a better solution would be to use an internal data model, against which the different formats are matched. Semantic Web solutions for defining semantics and metainformation, such as ontology language OWL [29], have been proposed for that purpose [20].

In our practical case study we also noticed that the integration effort may reveal inadequacies and errors in the interfaces of the SOA-based systems. For instance, not all the provided services and features are necessarily used in "typical usage scenarios" of the system. This may imply that faults, errors and inadequacies in them may have not been found earlier. They may, however, be exposed during the integration and especially after using the integrated system.

After the transformation principles have been decided, the next step is to identify the correspondences and differences between different languages to be transformed. Besides the differences, problems may also raise due to limitations and/or extensibility of the languages. In addition, tools and implementations used by the SOA realizations and those used to support the integration efforts (if appropriate ones are available) might influence the transformations. For instance, the tools might limit the ways the languages are or can be used. For Web services systems, for example, tools are available for generating WSDL descriptions from an existing service interface implementation. Also, tool support is available for generating SOAP messages based on WSDL descriptions. In our earlier study, we have observed that the tools to e.g. generate service descriptions differ; we noticed that different vendors' tools generated different WSDL descriptions from the same service interface [9].

While many varying points may exist in the languages to be integrated, the overall goal for the integration and the usage purposes can and should be used to guide the integration. To further high-light the complexities of the transformations, we will next discuss them from the point of view of service descriptions, messaging and registries.

2.3. Case study

In this chapter we discuss various challenges related to SOA-to-SOA integrations. The points presented are based on our practical experiences, gained in a real world case study. In this case study we integrated Plenware PlugIP software and development platform with a Web services system. PlugIP is a platform that allows developing highly distributed software systems. It is based on message dispatcher architecture and it uses a registry that differs from UDDI in terms of formats used and functionality provided. PlugIP has been implemented according to SOA principles. It uses proprietary formats for messaging and service descriptions. PlugIP systems' software components (plug-ins) are divided into *client plug-ins*, *service plug-ins* and *core plug-ins*. Core plug-ins are part of the plugIP core system and offer features such as authentication, validation and registration of service plug-ins, a watchdog, logging and system monitoring. Most of those features are not supported by the Web service concept. Each added plug-in will be attached to a product group. A group limits the visibility of a plug-in to the group it belongs to. The main idea of service plug-ins is the same as with Web services; the purpose is to offer a service for anyone who needs it. A client plug-in can be any independent application that uses service plug-ins.

This case study had two goals. First, from the point of view of plugIP, a Web service needs to look like a plugIP interface. Second, correspondingly, PlugIP plug-ins need to look like a Web service for Web service clients. In this case study, the integration problem was solved by building a gateway between these two SOA-based systems [4]. The aim for gateway was to have dynamic solution for this integration need. Gateway translates all incoming messages to target system's format and delivers translated messages to a service that is being called. It translates service descriptions from one format to another at run-time and keeps registries synchronized. This allows systems to communicate with each other and enables a client to find a service from both systems, no matter if the client is a plugIP client or a Web service client.

Due to the differences the two SOA-based systems offer, we followed the "smallest common dominator" approach and did not enhance either of the system with functionalities provided by the other. The aspects discussed in what follows are by no means meant to be a covering analysis of the challenges, but are intended to give light on the complexity of the problem of integrating two SOA realizations.

2.4. Service descriptions

A mapping between different service description languages is needed for at least two reasons. First, a client application in one SOA realization needs to be able to communicate with services in another SOA realization; it formulates the service request based on the service description. Second, the service descriptions stored in one registry need to be able to be used and found by clients of another SOA realization.

In Web services systems, WSDL is the current standard way to describe and locate Web services. They thus act as IDLs for the services and are a key for interoperability. A service description should contain all the information needed for the client applications to call the service, including the service interfaces, data types used in them, address of the service and communication mechanisms supported by the service. In WSDL 2.0, for example, they are given in XML format with elements *interface*, *types*, *service* and *binding*, resp.

When transforming one service description given in, say WSDL, to another format, the semantically corresponding parts should be first identified. A simple solution, as discussed above, would be to transform only the information that can be expressed with both of the formats. That might, however, yield to difficulties; e.g. the expressiveness of the transformed service description might be insufficient. For instance, if different types of registries are used in the different

SOA realizations, storing, categorizing and searching services might get difficult.

2.5. Messaging formats

Transformation from one messaging mechanism to another is a more complicated matter than the transformation of service descriptions. One reason for this is the several dependencies the messaging format has with other languages and protocols. For instance, support for the transportation protocol (e.g. HTTP) used may differ. Responsibilities of the different levels in the protocol stack may differ in different SOA realizations. This may be the case e.g. with the security aspects; for one solution it might be sufficient to rely on SSL if HTTP connections are used, while in another solution digital signatures and encryptions are required, which are handled with the messaging mechanism itself. Also, the messages themselves may require information, e.g. concerning routing, from the transportation protocol level. Also, sending messages may require additional information from the service descriptions.

As mentioned above, to enable communication between a client application in one SOA realization and with a service in another, it should be decided which party is responsible for the transformations. In this case, a service call can be implemented at least in two ways. One option is to convert a message in one format to a message in another, either by the client or by a third party, for the called service to be able to understand and process it. Another option might be to generate a run-time wrapper for the called service and let the conversion be a responsibility of the service. The latter option is, however, rarely applicable, mostly depending on the dynamicity of the services systems and since the requirements of the called service (i.e. information needed) need to be taken into account.

2.6. Registries

To support storing and searching of available services, external service registries and databases have been used in practical services system applications. Besides information relevant for the services systems, such registries and databases may also include other information. In the integration it should take care that only the right and relevant services are accessible in all SOA realizations.

The same problem of varying support and features that applies to messaging formats and service descriptions, also apply to registries. In the case of ebXML [13], for instance, the registries support storing and managing of business process specifications, collaboration agreements, business scenarios and profiles etc. These features are not similarly supported with e.g. UDDI, which is often used in Web services systems. In cases where the registries are used solely for the purposes of the service-based systems, replacing the several registries with one common registry might be a good solution. That would also provide more efficient means for service categorizations and linkage in the registry, which, in turn, gives better support for the clients in searching services.

3. Profile-based support for SOA-to-SOA integrations

We propose a UML profile based approach to support the identification and management of the similarities and differences of different formats used in different SOA realizations. This approach could serve as an initial step towards tool support for linking and integrating such formats. Such linkage requires well-defined descriptions of the features the formats should support. Ontology languages, such as OWL [29] have been proposed in the literature [20] for defining semantics and metainformation for Web services systems. These languages typically come from the Semantic Web domain. We, instead, propose the use of UML profiles, which are used in software engineering essentially for defining concepts, rules and also semantics for certain domain and/or purpose. The benefits of using UML profiles include the following:

- model analysis and checking tools are straightforward to be implemented, some of such already exist,
- the notation is familiar to software engineers,
- various other notations and views can be derived from them, and
- tool support is available.

To allow linkage of different service description languages, messaging formats and registries, well-defined descriptions of semantics is need. If provided, the integration efforts could even be at least partly automated. Rather than aiming at language-to-language transformations, mappings to a common model would be desirable. The goal would be to have a model with well enough support for expressing semantical rules and metainformation to allow automation (at least, partly) of such evolutional and integration activities. Profiles for SOA and its realizations could be constructed as follows.

Step 1 Constructing a profile for the chosen SOA concerns. For instance, profiles can be constructed to define general rules and requirements concerning service descriptions, messaging and registry usage.

Step 2 Transforming particular language grammars into UML profile representations. In case the grammar is given e.g. as an XML Schema document, the process is rather straightforward, as demonstrated in [10].

Step 3 Extending the language profile (formed in Step 2) to also support the generic SOA rules (formed in Step 1), i.e. rules for a family of formats (e.g. rules for all service description languages).

As an illustrative example, we will next discuss the steps to construct a SOA profile for WSDL service description language.

3.1. Example: construction of a SOA profile for service descriptions

Selonen and Xu have proposed the concept of *architectural profiles* in [25]. Architectural profiles are extended UML profiles specialized for describing architectural constraints and rules for a given domain. The structural constraints and rules are given in a form of a class diagram. While the architectural profiles were developed for validation of UML models, special operations we developed that can be used to check whether a given UML model conforms to the rules given in a profile (or a set of them). This approach has been implemented in artDECO toolset [1, 19] and applied when maintaining a large-scale product platform architecture and real-life product-line products built on top of this platform [23]. We have earlier applied the approach also for WSDL validation [10]. An architectural profile has two parts: stereotype definitions and constraint definitions. The former defines the stereotypes used in the architectural profiles and in the actual UML models and views. The latter states the constraints and rules that the views must conform to. Together they define the WSDL 2.0 grammar rules.

Figure 2 shows a stereotype definition part of WSDL 2.0 profile. The classes have two kinds of stereotypes: «metaclass» and «stereotype». The former refers to an element in standard UML metamodel. A class with the latter stereotype extends the metamodel element it is attached to with a dependency relationship. Here we follow UML 1.4 with Action Semantics. All new model elements are extended from UML metaclass "Class" or "Attribute". All of them, except «Extension» classes, represent key elements given in the WSDL specification. Stereotype «Extension» is designed for extensible elements that are not defined in WSDL specification nor in our architectural profiles.

Figure 3 shows the constraint definition part of the WSDL 2.0 profile. The WSDL specification introduces many optional attributes for various elements. Therefore, we use stereotypes «required» and «optional» to distinguish the required attributes from the optional ones. The aggregation relationships indicate a containment relationship. E.g. a message element may have zero or more (multiplicity 0..n) part elements as its subelements, while a particular part element belongs to exactly one message element (multiplicity 1).

The WSDL 2.0 profile can be used to validate instance WSDL descriptions, as demonstrated in [10]. The validations are done at UML level, which im-

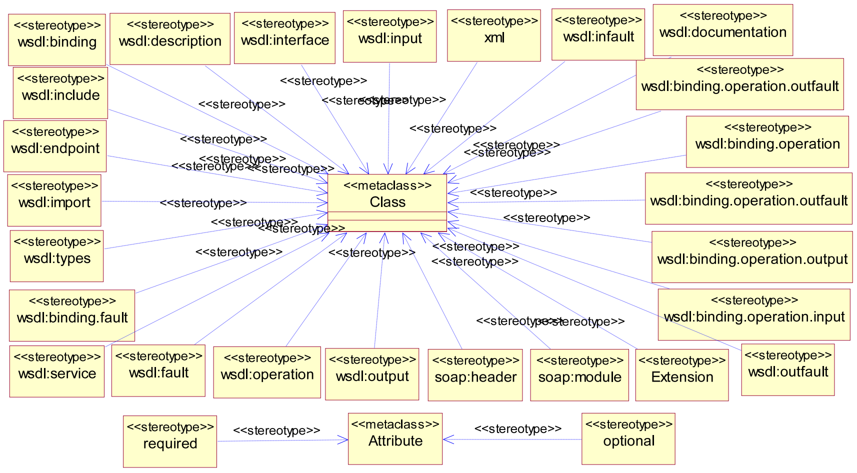
<<stereotype>> wsdl:binding
<<stereotype>> wsdl:description
<<stereotype>> wsdl:interface
<<stereotype>> wsdl:input
<<stereotype>> xml
<<stereotype>> wsdl:infault
<<stereotype>> wsdl:documentation
<<stereotype>> wsdl:include
<<stereotype>> wsdl:binding.operation.outfault
<<stereotype>> wsdl:endpoint
<<stereotype>> wsdl:binding.operation
<<metaclass>> Class
<<stereotype>> wsdl:import
<<stereotype>> wsdl:binding.operation.outfault
<<stereotype>> wsdl:types
<<stereotype>> wsdl:binding.operation.output
<<stereotype>> wsdl:binding.fault
<<stereotype>> wsdl:binding.operation.input
<<stereotype>> wsdl:service
<<stereotype>> wsdl:fault
<<stereotype>> wsdl:operation
<<stereotype>> wsdl:output
<<stereotype>> soap:header
<<stereotype>> soap:module
<<stereotype>> Extension
<<stereotype>> wsdl:outfault
<<stereotype>> required
<<metaclass>> Attribute
<<stereotype>> optional
<<stereotype>>

Figure 2. Stereotypes for a WSDL 2.0 profile

plies that the instance WSDL documents are first reverse engineered to UML representation. The tool developed transforms a WSDL document into a UML class diagram using stereotypes specified in the stereotype definition parts of the architectural profiles: elements are modeled as classes and attributes of the elements are placed as attributes of the corresponding classes. The aggregation relationships are formed based on element-subelement relationships in the WSDL document. Since the actual mapping between the model and the profile is based on stereotypes, the classes in the reverse engineered model are given stereotypes according to their class names. Finally, the UML representation of the WSDL document is validated against the profile, which defines the grammar for WSDL. In this approach, also additional profiles were used, e.g. one for defining WS-I Basic Profile recommendations for WSDL [28].

In this paper, we propose extending such language-dependent profiles with SOA-related rules. In the case of the WSDL 2.0 profile, extensions are made to also support the SOA service description profile, namely, the profile that defines the requirements for all service description languages (see Figure 4). In Figure 4, for example, *ws:Address* element (belonging to the SOA profiles) is extended with *wsld:endpoint* element (belonging to the WSDL 2.0 profile). This means that all the constraints defined in the WSDL 2.0 profile (Figure 3) and the constraints defined in the constraint definition part of SOA service description profile, if exists, apply to *wsld:endpoint* element. Note that the constraint definition part of SOA service description profile may be missing altogether or it may include (also) constraints that are not structure-related. In the latter case, the conformance checking needs to be geared accordingly.

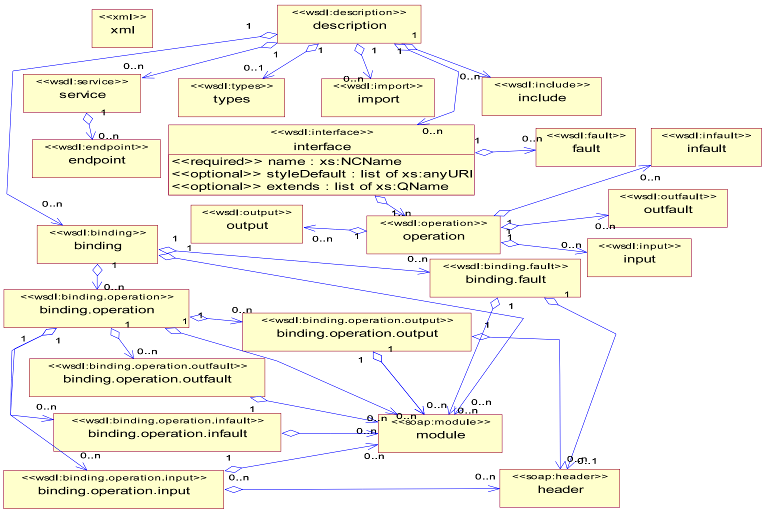

Figure 3. Constraint definitions for WSDL 2.0 profile

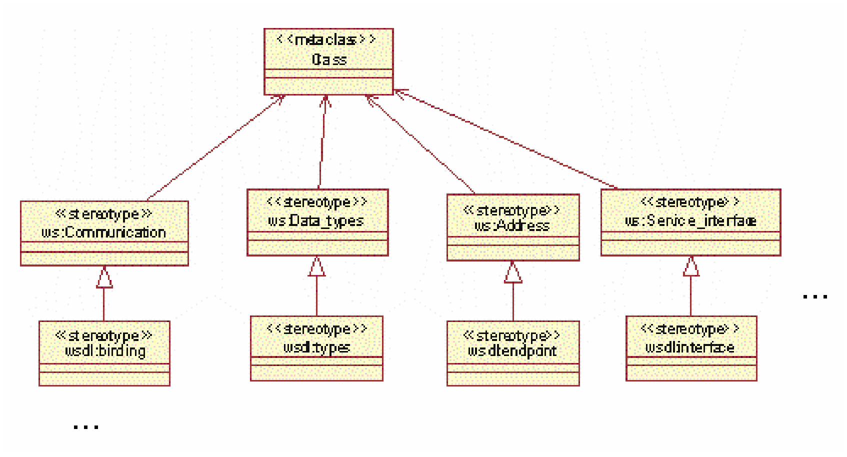

Figure 4. Extending the WSDL 2.0 profile

3.2. Using the SOA profiles

Profile-based support for validating and comparing languages used in different SOA realizations is sketched in Figure 5. From the process of constructing the full profiles (Steps 1-3) explained above, only the last step (profile extension) is shown in the figure as activities “0 (extend)". The constructed profiles can then be used for various purposes, e.g.:

1. for comparing the language grammars and instance documents at UML level visually;
2. for validating individual instance documents (e.g. WSDL description) against the language profile (e.g. WSDL profile) [10] and against the SOA profile to identify possible violations or to ensure that the document follows the rules defined in the profiles; and
3. to support future maintenance activities and managing the evolutionary changes of the languages: the profiles can be used to identify and study the impacts of the changes.

Comparison of language grammars. The comparison of two languages (activity 5 in Figure 5), say WSDL and language Y that is also used for service descriptions, can be (partly) automated after the corresponding profiles have been constructed. This requires specific operations that can take two (or more) profiles as input and identify the corresponding and differing parts in them. Such tools have been developed at TUT [24]. Also several other tools for comparing UML diagrams exist [33, 11]. The approach proposed by Selonen and Kettunen in [24] is flexible and would thus well fit for the purposes of comparing language grammars at UML level. For instance, the approach and the implementation allows users to adjust correspondence rules, that is rules for identifying corresponding parts in the diagrams. This is a useful feature, since the application purpose may effect on what parts are to be identified as corresponding. In the case of UML class diagrams, for instance, in some cases the classes need to match completely, including their names, stereotypes, attributes and methods, as well as their nearest neighborhood (e.g. associations related to them). In some other cases, matching simply based on the class names is sufficient. Correspondence rules in our case should be based on stereotypes.

The comparison can also be done at instance document level (activity 4 in Figure 5), which allows e.g. comparing two service descriptions given in different languages. This requires that the instance documents are first reverse engineered to UML representations (activities 1 in Figure 5), as explained in Section 3.1.

Validating instance documents. The profiles also allow, according to their intended usage, validation of models. The validation toolset artDeco [23, 25] could be e.g. used for that purpose. Since the language profiles (e.g. WSDL 2.0) are extended to support also the SOA service description profile, the validation of instance documents includes both language syntax validation and SOA-level validation, which are marked as activity 2 and 3 in Figure 5, resp. Also the application of artDeco requires that the instance documents are first reverse engineered to UML representations.

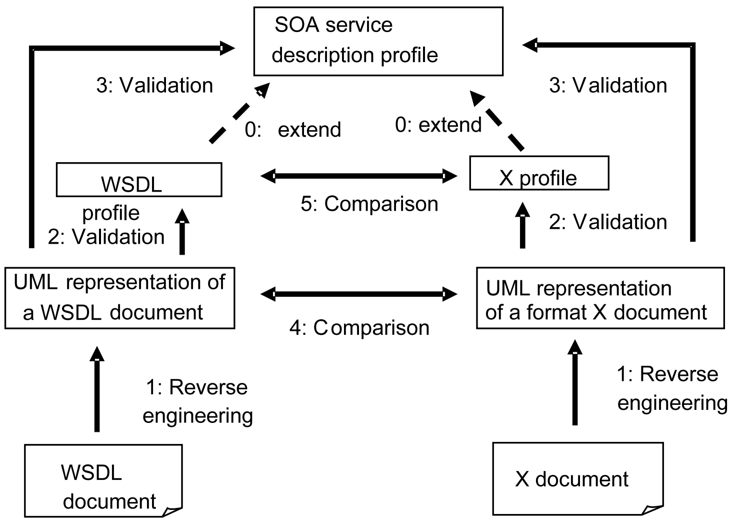

Figure 5. Profile based support for validation and comparison of languages used in SOA-based systems.

Maintenance and future evolution. The use of UML profiles also supports maintenance and management of future evolutionary changes to be made in the integrated system. For example, when yet another services-based system is intended to be integrated either with the whole system or one of the SOA realizations already integrated to it, the profiles are again used to guide the integration. Also other evolutionary changes may occur. For instance, the format used may evolve. In this case, too, the profile can be used to identify the differences between different formats. In [9] we have demonstrated how the differences in two WSDL format versions (1.1 and 2.0) can be conveniently identified by applying the comparison operations by Selonen and Kettunen [24]. Migrating a WSDL 1.1 profile to a WSDL 2.0 profile is also rather easy, since the UML profiles are models that can be easily edited [10].

Besides supporting language updates, as discussed above, easy and flexible changeability and extensibility of UML profiles allows the proposed approach to be geared according to specific needs. For instance, if there is a need to take some of the variation points discussed in Section 2.1, say security, into account, the existing profiles could be extended or new profiles could be added to support that. This requires naturally that the tool support (e.g. validation) is geared accordingly. It should be noted, however, that use of the profile-based approach proposed does not solve any possible mismatches, it just supports in their identification.

4. Summary

Recent software development methods and principles, such as model-driven development (MDD), emphasize the role of abstractions and automation. Abstractions aim at reusability: different kinds of lower level models for different domains can be derived from the same abstract models. The transformations of the models, in turn, are assumed to be fully or semi-automated. To enable this, the metamodels and (UML) profiles are used to define the domain-specific terms, relations and rules.

The MDD principles are potentially useful also in the development of SOA-based systems. This has already been acknowledged in various techniques and tools for transforming high-level workflow descriptions into concrete and executable business process descriptions, given e.g. in BPEL [2, 6, 22, 26]. Also, UML profiles for SOA have been proposed [5, 7]. In [5], Heckel et al. propose a UML profile to define the appropriate syntax of SOA in a well-formed fashion. They further suggest defining the semantics of the profile, which allows interpretation of the behavior of SOA applications, using UML collaboration diagram notation. The SOA profile proposed by Johnson [7] also aims to cover various activities through the development lifecycle and to provide views to different stakeholders. These approaches provide valuable support for development of SOA-based systems. However, to the best of our knowledge, semantics and MDD-fashion approaches have not been as successfully used to support integration and evolution of services and service-based systems.

In this paper we propose application of MDD principles and UML profiles in particular to support SOA-to-SOA integration activities. First, as an analogy to MDA, SOA can be thought of corresponding to the platform-independent level and SOA realizations to the platform-specific level. The transformation needed in the integration of two SOA realizations then corresponds to horizontal transformations in MDA. Second, (UML) profiles and metamodels play a key role in MDD to allow reuse and evolution. In this paper we propose an approach, also relying on the use of UML profiles, to guide the transformations needed when integrating two or several SOA-based systems. The profiles are used to define SOA-level rules and requirements that should apply to all its realizations. The same way platform-independent rules and requirements in MDA should apply to all the platform-specific models and to the actual implementations. The essence in using profiles is to define concepts, metainformation, and semantics in a well-defined way.

The approach proposed was motivated by a concrete SOA-to-SOA integration case study we have applied. In this work we integrated Plenware's proprietary SOA-based system, called PlugIP, with a Web services system. In this paper we also discuss the challenges related to such SOA-to-SOA integration projects from various points of view, to emphasize the complexity of the problem.

As part of our future work we plan to fully implement the approach proposed in this paper.

Acknowledgement

This work has been financially supported by TEKES, Nokia, Plenware and Solita. The authors would like to thank Imed Hammouda and the reviewers for their valuable comments.

References

[1] **Airaksinen J., Koskimies K., Koskinen J., Peltonen J., Selonen P., Siikarla M. and Systä T.**, xUMLi: Towards a tool-independent UML processing platform, *Proc. of the 10th NWPER Workshop, IT University of Copenhagen, Denmark, 2002,* ed. K. Østerbye, 1-15.

[2] **Business Process Modeling Initiative,** *Business process modeling language*, `http://www.bpmi.org/`, 2005.

[3] **Canfora G., Fasolino A.R., Frattolillo G. and Tramontana P.**, Migrating interactive legacy systems to web services, *Proc. of CSMR*, IEEE Computer Society, 2006, 24-36.

[4] **Hartikainen M.**, *Integration gateway for SOA-based systems*, MSc thesis, Tampere University of Technology, 2007.

[5] **Heckel R., Lohmann M. and Thöne S.**, *Towards a UML profile for serivce-oriented architectures*, MDAFA, 2003.

[6] **IBM**, *The Emerging Technologies Toolkit (ETTK)*, `http://www.alphaworks.ibm.com/tech/ettk` (last visited 2007)

[7] **Johnston S.**, *UML 2.0 Profile for Software Services*, IBM whitepaper. `http://`

`www-128.ibm.com/developerworks/rational/library/05/419_soa/` (last visited 2007)

[8] **I-Logix**, *Rhapsody*, `http://modeling.telelogic.com/` (last visited 2007)

[9] **Jiang J., Lipponen J., Selonen P. and Systä T.**, *Visualizing and comparing Web service descriptions in UML*, NWUML, 2005.

[10] **Jiang J. and Systä T.**, UML-based support for designing and validating Web service descriptions, *Journal of Web Services Research (JWSR)*, **3** (2) (2006), 101-120.

[11] **Kelter U., Wehren J. and Niere J.,** A generic difference algorithm for UML models. *Proc. of the SE 2005, Essen, Germany, 2005,* 295-304.

[12] **Meunier J.-N., Lippert F. and Jadhav R.,** RT modeling with UML for safety critical applications: the HIDOORS project example, *Proc. of SVERTS, co-located with UML*, 2003.

[13] **OASIS**, *Electronic Business XML*, `http://www.ebxml.org/` (last visited 2007)

[14] **OASIS**, *Organization for the Advancement of Structured Information Standards*, `http://www.oasis-open.org/` (last visited 2007)

[15] **OASIS**, *Univeral Description, Discovery and Integration (UDDI) specifications*, `http://www.oasis-open.org/` (last visited 2007)

[16] **Ober I., Graf S. and Yushtein Y.**, Using an UML profile for timing analysis with the IF validation toolset, *Dagstuhl-Workshop* **06022** *– MBEES 2006, Model-Based Development of Embedded Systems*, IEEE, 2006.

[17] **OMG**, *Unified Modeling Language Specification v. 2.0.*, `http://www.omg.org/` (last visited 2007)

[18] **OMG**, *Model Driven Architecture (MDA)*, `http://www.omg.org/` (last visited 2007)

[19] **Peltonen J. and Selonen P.**, An approach and a platform for building UML model processing tools, *Proc. of the ICSE Workshop WoDiSEE'04*, 2004, 51-57.

[20] **Pollock J.T. and Hodgson R.**, *Adaptive information - improving business through semantic interoperability, grid computing, and enterprise integration*, Wiley, 2004.

[21] **Rational Software**, *Rational software architect*, `http://www-306.ibm.com/software/rational` (last visited 2006)

[22] **Rito-Silva A., Fernandes S., Martins J. and Domingos D.**, *Microworkflow component framework supporting service composition. INESC-ID, Deliverable IST-2001-37724 ACE-GIS D4.2*, 2003.

[23] **Riva C., Selonen P., Systä T. and Xu J.**, UML-based reverse engineering and model analysis approaches for software architecture maintenance, *Proc. of ICSM*, 2004, 50-59.

[24] **Selonen P. and Kettunen M.**, Metamodel-based inference of inter-model correspondence, *Proc. of CSM* , 2007, 71-80.

[25] **Selonen P. and Xu J.**, Validating UML models against architectural profiles, *Proc. of ESEC / SIGSOFT FSE*, 2003, 58-67.

[26] **Skogan D., Grønmo R. and Solheim I.** Web service composition in UML, *Proc. of EDOC*, 2004, 47-57.

[27] **Sneed H.M.** Integrating legacy software into a service oriented architecture, *Proc. of CSMR*, IEEE Computer Society, 2006, 3-14.

[28] **Web Services Interoperability Organization,**
`http:/ /www.ws-i.org/` (last visited 2007)

[29] **World Wide Web Consortium (W3C)**, *OWL-S, Semantic Markup for Web Services*,
`http://www.w3.org/` (last visited 2007)

[30] **World Wide Web Consortium (W3C)**, *Web Service Description Language (WSDL) 1.1.*,
`http://www.w3.org/` (last visited 2007)

[31] **World Wide Web Consortium (W3C)**, *Web Service Description Language (WSDL) 2.0.*,
`http://www.w3.org/` (last visited 2007)

[32] **World Wide Web Consortium (W3C)**, *Simple Object Access Protocol. (SOAP)*,
`http://www.w3.org/` (last visited 2007)

[33] **Xing Z. and Stroulia E.**, UMLDiff: An algorithm for object-oriented design differencing, *Proc. of ASE*, ACM Press, 2005, 54-65.

T. Systä and M. Hartikainen
Department of Software Systems
Tampere University of Technology
P.O. Box 553
FIN-33101 Tampere, Finland
`{tarja.systa,mikko.hartikainen}@tut.fi`