Domain Specific Language Approach on Model-driven Development of Web Services

Viet-Cuong Nguyen, Xhevi Qafmolla, Karel Richta

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo namesti 13, 121 35 Prague, Czech Republic
nguyevie@fel.cvut.cz, qafmoxhe@fel.cvut.cz, richta@fel.cvut.cz

Abstract: As modern distributed and cloud architecture keep gaining their popularity, web services have become the programmatic backbones of more and more systems. Developing web services requires gathering information from different aspects. Model-driven engineering promises to ease the burden of development and promote reuse of web services by focusing more on a higher level of abstraction. Current approach of modeling web services using UML is not well-suited since UML is created for multiple disciplines and is not specific for web service development. With current growing scale of distributed systems, the challenge is not only in development but also integration and maintenance of web services. Introducing a domain specific language (DSL) for modeling of web services promises to become a novel approach and could be the solution to the current problem with web service modeling and development. This article outlines the analysis as well as the current state of the problem domain and introduces an approach to model-driven development of web services by implementing a domain specific language called SWSM (Simple Web Service Modeling). This approach aims to solve problems that UML could not effectively resolve and promote efficiency with a non-complex language facility for modeling and code generation of web services. Our best practices and observation during the design of SWSM are also presented.

Keywords: web service; model-driven development; DSL; SWSM

1 Introduction

Cloud computing and distributed systems continue to gain more mainstream adoption as more companies move into the cloud. With mobile gradually taking over the desktop experience, cloud computing continues to accelerate and have more significance [17]. Model-driven Engineering methodologies have been applied (as a solution) for better reaction to market trends and aims to increase efficiency as well as bring more agility to the development life-cycle of cloud and

distributed systems. However, since there are many different applicable domains in web applications and distributed systems, it is unattainable to finalize a method or approach that would fit in every situation. This article is an effort towards the solution for this issue by analyzing the concepts involved in key aspects of web service design and introduces an approach to the development of web services by using model-driven techniques with domain specific language. As a result, a DSL for modeling of web services named SWSM (Simple Web Service Modeling) was developed and introduced. To demonstrate this approach, a case study of web service development from modeling to code generation is also illustrated with the associated techniques.

This article is structured as follows: In the next section, we review some knowledge of Model-driven Development (MDD) and domain specific language as background information. The subsequent section discusses the current state of web services development using model-driven techniques. We also highlight the features of the DSL that we aim achieve when designing a new DSL for modeling of web services. In the next section, we introduce SWSM (our designed DSL) for modeling and development of web services and how to apply it at a specific point during design phase. In the last section, we present some conclusions on web service development using SWSM and also related works of our research in the field of Model-driven Engineering.

2 Background

2.1 Model-driven Engineering

Model-driven engineering (MDE) is a software development methodology, which focuses on creating and exploiting domain models. Models can be perceived as abstract representations of the knowledge and activities that govern a particular application domain. Models are developed though-out various phases of the development life cycle with extensive communication among product managers, designers, developers and users of the application domain. MDE aims to increase productivity by maximizing compatibility between systems, simplifying the process of design and promoting communication between individuals and teams working on the system [16].

The Object Management Group's (OMG) initiatives on MDE contain the Model-driven Architecture (MDA) specification. MDA allows definition of machine-readable applications and data models that enable long-term flexibility with regards to implementation, integration, maintenance, testing and simulation [14] [15]. There are two main modeling classes in MDA:

- Platform Independent Models (PIMs): these are models of the structure or functionality, which are independent of the specific technological platform used to implement it.
- Platform Specific Models (PSMs): these are models of a software or business system, which are bound to a specific technological platform.

In the MDA, models are first-class artifacts which are later integrated into the development process through the chain of transformations from PIMs through PSMs to coded application. The mapping and transformation between PIMs and PSMs are based on meta-model concepts. These concepts can be described by technologies such as Unified Modeling Language (UML), Meta Object Facility (MOF) or Common Warehouse Meta-model (CWM) [22, 16]. These languages are considered as general-purpose modeling languages.

Currently, there are many challenges in implementing MDD due to the lack of standardization and tools. There are specific desired aspects for each application within its domain and this makes it difficult to design a tool that fits all.

2.2 Web Services

With the growing demands in recent years, distributed computing and cloud processing systems are made possible by adopting a new paradigm of Service-Oriented Computing (SOC). SOC integrates networks of connected business applications from many different locations. In the SOC paradigm, web services are currently considered one of the most dominant technologies. Web services are software systems designed to support interoperable machine-to-machine interaction over a network. The important components of web services know-how include XML technology, Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI) [21]. There are two popular classes of web services: REST-compliant web services and Simple Object Access Protocol (SOAP) web services [16]. Currently, the development of web services in MDD involves using UML to specify services precisely and in a technology-independent manner. However, UML is by far not the optimal way for modeling of web services. The efficiency could be improved by using a specific language to address the detailed nature of web services. Introducing a new DSL can set up the stage for automatic generation of a part of the XML and code, such as Java code, that implements the services. It also makes it easier to re-target the service(s) to use different web technologies when required.

2.3 Domain Specific Language

In software development and domain engineering, a domain specific language is a programming or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution

technique. The concept is not new. Special-purpose programming languages and all kinds of modeling or specification languages have always existed, but the term has become more popular due to the rise of domain specific modeling [19]. Adoption of domain specific language can be a solution to several problems encountered in various software development aspects. A DSL can reduce the costs related to maintaining software [5].

In comparison to other techniques, DSL is considered as one of the main solutions to software reuse [9]. On the other hand, using DSL also promotes program readability and makes its understanding easier, because it is often written at a good abstraction level. It enables users without experience in programming to create the models or programs as long as they possess knowledge of the targeted domain. Another advantage of a DSL for modeling is the ability to generate more verification on the syntax and semantics than a general modeling language. This can reduce errors (and burden) on the debugging process. DSL for modeling however also has several drawbacks. There is a long learning curve for a new language, even though as a specific language, it would be a lot easier to learn than a general programming language. Another disadvantage is the lack of capable human resources. Since a general language is adopted by more people and staff, it could be much easier to find staff capable of solving the problem using their language knowledge, rather than DSL.

2.4 Current Approaches in Web Services Development

Currently development of web services falls into two main categories associated with the order in which models are developed: bottom-up and top-down. In bottom-up development, the design process starts with a given prototype or presentation of a class. Other web service artifacts are generated from the given prototype. This means part of the implementation must be designed at the first stage. This approach implies that changes made during the first stage will propagate and require changes on all model artifacts. This can bring benefits only when there is an existing system that has a similar business logic, which was already implemented.

In top-down approach we first design the abstraction and description of a web service. After that, we add detail implementation(s) and business logic to it. In top-down process, modeling is a crucial part. A good design in this phase is very important to the overall quality of the web service. The UML approach during this phase has some drawbacks. UML is a tool for generic design, it is not conducive for addressing all aspects of web service. Besides, creating XML/WSDL is a complicated process with a lot of detail information. In contrast, modeling process at the first place is intended to abstract away unnecessary details and makes it easier to understand the system. Hence, there is a need to create a better mechanism to solely support the design of web services. Adopting a dedicated DSL for this purpose can turn into a promising approach in this situation.

3 Challenges

Advances on programming languages still cannot cover all aspects of the fast-growing complexity of web platforms. In a wide range of systems, especially distributed ones, more and more middle-ware frameworks are developed in languages such as Java or .Net, which contain thousands of classes and methods as well as their dependencies. This requires considerable effort to port systems to newer platforms when using these programing languages [17]. Therefore, general programming languages cannot be considered as first-class languages to describe system-wide and non-functional aspects of a system. There is a need to raise the level of abstraction while still providing specific domain attributes for modeling of such systems.

With mobile technology adoption continuing to gain momentum, in the next few years more cloud based and software-as-service (SaaS) systems will grow. As more systems migrate to the cloud, there is a big space for web services to continue gaining popularity. SaaS, and recently Platform-as-a-Service (PaaS), as different layers of cloud computing, require different approach to web service development and deployment. In these infrastructures, the so-called multi-tenancy becomes an essential factor. The multi-tenant architecture (as depicted in Fig. 1) ensures the customization of tenant-specific requirements while sharing the same code-base and other common resources. In this figure, four customizations of different tenants are built based on the shared service implementation and infrastructure. Web services in multi-tenant platforms need a way of abstracting away the configuration and make it possible for every part of the service to be customized for a specific tenant. These platforms are often built from the meta-data driven solution. This therefore means that the application logic can be based on meta-data which later can be customized [17].

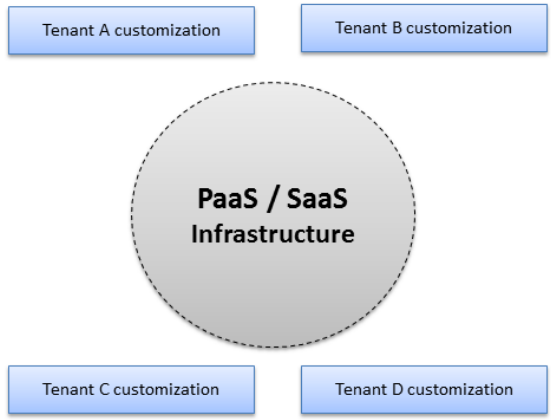

Figure 1
Multi-tenancy architecture: customizations of 4 tenant-specific requirements sharing the same base service implementation and infrastructure

The challenge in this architecture is to adopt or develop a modeling language at the appropriate abstraction level to separate the logical models from its technical aspects. This detaches the definition of service architectures independently from the used specific platforms. A modeling language raising the level of abstraction allows us to reuse models and keeps platform-specific artifacts at a separated tier in the development workflow. Having a modeling language based on services aspects with the ability to set aside technical concerns and still be able to tackle a problem in a specific platform is hard to come across.

There are existing general purpose modeling languages such as UML. UML is often used as a standard language for software systems modeling. It is able to represent various kinds of software systems, from embedded software to enterprise applications. In order to achieve this flexibility, UML provides a set of general elements applicable to any situation such as classes or relationships [1]. However, in the SaaS or PaaS architecture, the class systems in UML often force applications to be represented or surrounded by classes, this could make the models difficult to understand and use. In an effort to improve this, UML provides facilities to specialize for a specific domain, so-called UML profiles. Nevertheless, these mechanisms are not able to represent the semantics behind the domain concepts [1]. The challenge therefore remains in defining a domain specific language that can be suitable for the modeling and development of this infrastructure. In the case of modeling web services, the creation of a high-level DSL turns into a necessity for software reuse, higher development speed and better cost-effectiveness.

4 Features of a DSL as a Modeling Language

Introducing a new DSL with the support for modeling at a good abstraction level is crucial. This DSL can later be used for automatic generation of the model artifacts and code that implement the services. In theory, a general modeling language could also be used for this purpose but an appropriately designed DSL will perform the same job much more effectively. We define a set of features that are essential to the DSL design in model-driven development of web services. All of these features should be considered during the creation of a DSL to ensure the quality of the language.

Effectiveness: The language needs to be able to deliver useable output without having to re-tailor based on specific use case while being easy to read and to understand. This means that the language is able to bring up good solution on specific domain and focus on solving the particular range of problems. Effectiveness also needs to guarantee the unambiguity feature of language expressions and capability to describe the problem as a whole from a higher level.

Automation and Agility: As the modeling language can raise the level of abstraction away from programming code by directly using domain concepts, an important aspect is the ability to generate final artifacts from these high-level specifications. This automatic transformation has to fit the requirements of the specific domain. Agility ensures that models can adapt to changes efficiently. This changes from models described by the language are also propagated to the next phase of development automatically.

Support Integration: The DSL has to be able to provide support via tools and platforms. The DSL needs to be able to integrate with other parts of the development process. This means that the language is used for editing, debugging, compiling and transformation. It should also be able to be integrated together with other languages and platforms without a lot of effort.

When designing and implementing DSLs as executable languages, there is a need to choose the most suitable implementation approach. Related work from Mernik et al. [12] identifies different implementation patterns, all with different characteristics. These patterns provide another perspective to consider when making the design decisions of DSL. These options can be broken down to the following categories:

- *As interpreter*: In this method, DSL constructs are recognized and interpreted using a standard 'fetch-decode-execute' cycle. With this pattern no transformation takes place. The model is directly executable.
- *As compiler/application generator*: DSL constructs are translated to base language constructs and library calls. People are mostly talking about code generation when pointing at this implementation pattern.
- *Using pre-processor*: DSL constructs are translated to other constructs in an existing language (the base language). Static analysis is limited to that done by the base language processor.
- *Embedded design*: DSL constructs are embedded in an existing general purpose language (the host language) by defining new abstract data types and operators. A basic example is application library. This type of DSL is mostly called an internal DSL. The good side of this is that grammar, parsers and tools are immediately available. However, the challenge with an embedded DSL is to tactfully design the language so that the syntax is within the confines of what the host language allows, while still remaining expressive and concise [17].
- *Using extensible compiler/interpreter*: A general purpose language compiler/interpreter is extended by domain specific optimization rules and/or domain specific code generation. While extending interpreters is usually relatively easy, extending compilers is hard unless they were designed with extensibility in mind.

- *Commercial off-the-shelf*: existing tools and/or notations are applied to a specific domain. In this approach, it is not needed to define a new DSL, editor and implement them. One only needs to make use of a Model-driven Software Factory. One example is using the Mendix Model-driven Enterprise Application Platform targeted at the domain of Service-Oriented Business Applications.
- *Hybrid*: a combination of the above approaches [17].

The choice of the approach is very important because it can make a big difference in the total effort to be invested in DSL development. With the success of open source projects like Xtext, development of DSL is made affordable and the development is focused on building the grammar, while support for static analysis and validation of models are possible out of the box.

We aim to maintain the set of features defined in this section while designing SWSM. This allows us to provide automatic transformation, agility and integration to the development cycle. This ensures that the process of model-driven development of web services using SWSM is efficient.

5 Model-driven Development of Web Service using SWSM as a Domain Specific Language

Web service technologies depend on the use of XML, SOAP, WSDL. These standards are important, but they do not effectively support automation of code evolution at different phases in the development cycle. A DSL for modeling web services is therefore useful because it can effectively support automation in model-driven development. In the process of designing a suitable DSL for this purpose, we consider some valuable lessons described in the work of Wile [20]:

Lesson T2: You are almost never designing a programming language. Most DSL designers come from language-design backgrounds where the admirable principles of orthogonality and economy of form are not necessarily well-applied to DSL design. One must be careful not to embellish or over-generalize the language.

Lesson T2 Corollary: Design only what is necessary. Learn to recognize your tendency to over-design [17].

Keeping these principles as an effective approach during design, we created SWSM as a modeling language for web services at the appropriate abstraction level. As a proof of concept, this language aims to increase the efficiency of the development process by letting users focus only on modeling of the essential aspects that comprise the web service.

The syntax needs to be simple, yet expressive and concise. The possible set of simplified syntax diagrams for the components of this DSL can be depicted as follows:

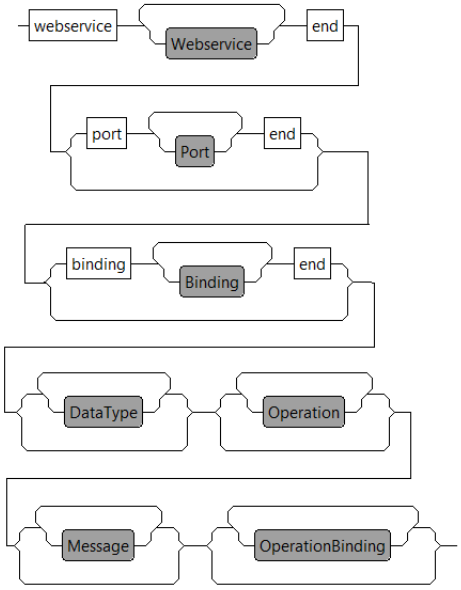

Figure 2
Simplified syntax diagram of web services with SWSM

To describe the service as an aggregation of several ports, the keyword *webservice* is used for modeling web services. Below is the syntax diagram for this model declaration:

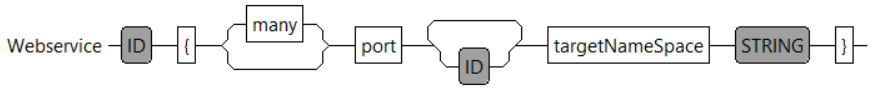

Figure 3
Web services syntax with SWSM

The semantics of the language expressions starts with the web service definition identified by its name (ID). There could be a number of ports associated with a web service and this mapping is described by the *port* keyword followed by a string identifier of a port. ID is a term representing the name (identification) of an element. The value of the target namespace is a string followed by *targetNamespace* keyword. This enables developers to specify the relationship between a port and a particular web service. In many cases, the association is a one-to-many mapping. The syntax diagram of ports can be depicted as follows:

Figure 4
Port syntax with SWSM

A port is identified by a name (ID term). It consists of one or many operations, each operation is then defined by input and output. This structure can be seen in the syntax diagram of an operation:

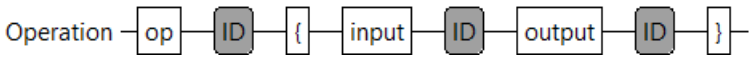

Figure 5
Operation syntax with SWSM

Each input and output of an operation is of the type *message*. The keywords *input* and *output* make the semantics of an operation signature easy to comprehend by describing the parameters for the operation with its returning type. The *message* element defines the data elements of an operation. Each message can consist of one or more fields (parts). These fields play the role of the parameters of a function call as in a traditional programming language. All modeled fields form the method signature for each operation.

Given a collection of operations $O_1 \ldots O_n$ with associated input and output messages, we define the mapping to web services and ports:

- One or more operations ($O_1 \ldots O_n$) are mapped to a port P1 to describe one function of a web service.
- P1 defines the connection point to a web service, one or more web services ($W_1 \ldots W_n$) are modeled within an SWSM file.

The message format and protocol details for a web service are modeled via binding. A *binding* is identified by its name (ID on the diagram) and the mapping to a port is described by *port* attribute. The binding style is represented by *bindingStyle* attribute. Value of *transport* attribute has the direct semantics of defining which transportation protocol to use. For example, in the case of HTTP, we can simply assign "*http*" to transport. This is more convenient than the approach currently used in WSDL where "http://schemas.xmlsoap.org/soap/http" is assigned. To define the operations that the port exposes, the mapping *operationBinding* is used. For each operation binding, the corresponding SOAP action is described with its encoding type of input and output.

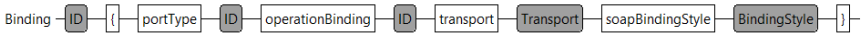

Figure 6
Binding syntax with SWSM

The best way of illustrating the syntax is to start modeling web services in a case-study. The first step in model-driven development of web services is designing the models. The output of this phase are models that conform to a web service meta-model, which can be represented in a textual format complying to the grammar of a DSL. Model artifacts are later used as input for the generation process. One of the important influencing factors is that any changes in the models will propagate changes in other stages. SWSM has a mechanism to support change propagation. To start modeling web services with SWSM, the process begins with representing the principal elements of a web service in the modeling language:

- Types: used to define the abstract elements in the description of the web service. They can be of a simple or complex type. They are identified by the keyword *type*.
- Messages: are units of information exchanged between the web service and the customer application (logically they are input/output messages and sometimes also fault messages). Each operation provided by a web service is described by at most, one input message and one output message. These messages relate to the parameters of the operation. In SWSM messages are identified by the *message* keyword.
- Interfaces (or portTypes in WSDL1.0): they constitute aggregations of operations provided by the service. In SWSM interfaces start with the keyword *interface*.
- Bindings: they specify in particular the protocol used to invoke the methods of an interface. In SWSM bindings start with the keyword *binding*.
- Services and ports: the service can constitute an aggregation of ports. A port is an endpoint enabling access to an interface through an URI address. Services are identified by the keyword *webservice* in SWSM. We can define multiple web services within one single design [18].

Utilizing MDD principles, web service development using SWSM can be decomposed into four steps:

1. Modeling the web service using SWSM language.
2. Enhancement and automatic validation of web service models.
3. Generating Java code using built-in code generation feature of SWSM.
4. Code refinement, refactoring and testing.

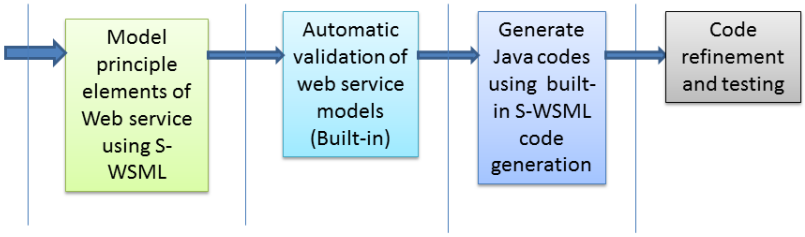

Figure 7
Stages in development of web services using SWSM language

To demonstrate the modeling syntaxes, we can see how SWSM is used to represent various essential elements of web services in an example. To model the service as an aggregation of several ports, the keyword *webservice* is used. The code bellow shows how the keyword *webservice* is used to define a service called DictionaryService:

```
webservice DictionaryService {
        port LookUpPort
        targetNameSpace "http://ws.mydictionary.net/lookUp"
  }
end
```

The semantics of the *webservice* block indicates a web service, which can consist of one or many ports. This can be seen from the syntax diagram described above. However, on this dictionary look-up example, there is only one port named *LookUpPort* declared. In the next step, the ports associated to the service also need to be defined:

```
port
 LookUpPort {
        op LookUp
        }
end
```

In a port, there are operations involved, *LookUp* operation is the one in this case. A port is associated with an *interface* by the association *binding*. The meta-class *interface* constitutes an aggregation of several operations.

```
binding LookupBinding {
        portType  LookUpPort
        operationBinding  OpBinding
        transport http
        soapBindingStyle rpc
}
end
```

Each operation (identified by the keyword *op*) consists of *message*(s) which define(s) its inputs and outputs. A *message* naturally refers to a meta-class indicating its simple or complex type:

```
op LookUp {
    input Input
    output Output
}
message Input {
    username : String
    word: String
}

message Output {
    meaning : String
    wordType : String
    related : Integer
}
```

In addition, we can also define the action associated with an operation by using the keyword *opBinding*:

```
opBinding OpBinding {
    soapAction "http://ws.mydictionary.net/lookUpAction"
    inputSoapBody literal
    outputSoapBody literal
}
```

For the HTTP protocol binding of SOAP, the value of *soapAction* is required. For other SOAP protocol bindings, this value could be omitted. The *inputSoapBody* and *outputSoapBody* indicate whether the message parts are encoded using some encoding rules.

Putting all the pieces together gives us the information needed to model a simple web service. These models are later used as input for code generation. SWSM makes it possible to design web services by using simple and fast syntaxes. In contrast to other approaches, SWSM is uncomplicated, rapid and easy to adapt. The syntax used in SWSM is simple and more intuitive in comparison to the complex structure of UML. The order in which aspects of a web service are defined is the same as the logical order, when we design a web service. This makes the designing process more natural and perceptive. Using SWSM enables us to focus only on the essential aspects of the web service. This approach promotes model-driven development principles and makes the web service development process more efficient.

All of the SWSM language infrastructures can be packed as a plug-in for the Eclipse integrated development environment. This includes a text editor with autosuggestion and validation capabilities. This enables the development phase to be carried out seamlessly. We also built a code generation feature (Java language) based on a code template engine and embedded it into SWSM. Textual models created using SWSM are used as input for the generation of Java web services. Code generation can be executed right within the editor [18].

The role of MDA in this development process is to raise the level of abstraction in which we develop systems. This is aimed to improve productivity similarly as when we moved from assembly language to third-generation languages. At first, third-generation language compilers did not produce code as optimal as hand-crafted machine code. Over time, however, the productivity increase justified the changeover, especially as computers speeded up and compiler technology improved [7]. SWSM is similarly used at a different level of abstraction to third-generation programming languages, to tackle overall productivity.

6 Related Work

Model-driven development of web services is still evolving to address the problem of increasing complexity and fast-changing technologies in the software industry. Model-driven development of web-services is discussed in the work of Benguria et al. [1]. This approach focuses on building platform independent models for service oriented architectures. The solution provides a platform independent meta-model and a set of transformations that link the meta-model with specific platforms following the MDA approach. There are also existing UML-based approaches to modeling services. UML collaboration diagrams have been used extensively to model behavioral aspects, such as service collaboration and compositions in the work of Bezivin et al. [2]. In this approach, the Platform-Independent Model is created using UML. This PIM is transformed using Atlas Transformation Language (ATL) to generate the Platform-Specific Model based on three target platforms: Java, Web Service and Java Web Service Developer Pack (JWSDP). This approach showed that UML profiles allow the extension of the UML meta-model. However, UML profiles make the creation of transformation rules difficult.

The support for SaaS and Services Modeling has also been addressed by providing lightweight extensions to UML through Profiles. These approaches can be seen in the work of Fensel and Frankel et al. [6, 7, 3]. UML-profiles for services and SOA are proposed by Heckel et al. [8]. This effort developed a suitable syntax for this domain by sketching a UML profile for SOA based on UML 1.x standards with a direct mapping between WSDL 1.1 elements and their model elements. Once the profile is properly defined, its semantics can be given in terms of a graph

transformation. This approach has an advantage of UML generality, it can be used to model just about any type of application, running on any type and combination of hardware, operating system, programming language, and network. However, since UML is large and complex, using multiple models/diagrams makes it difficult to keep them consistent with each other and more code has to be added manually.

There are also other efforts to provide domain specific languages for modeling of web services and service-oriented architecture. A qualitative study that provides some analysis of a number of such approaches through a series of three prototyping experiments, in which each experiment has developed, analyzed, and compared a set of DSLs for process-driven SOAs, can be seen in the work of Oberortner et al. [13].

Maximilien et al. [11] developed a DSL for Web APIs and Services Mashups. This effort describes a domain specific language that unifies the most common service models and facilitates service composition and integration into end-user-oriented Web applications. A number of interesting design issues for DSLs are discussed including analysis on levels of abstraction, the need for simple and natural syntax as well as code generation.

On the track of non-UML-based modeling approaches, there are efforts also supporting modeling of services. The Web Services Modeling Framework (WSMF) [6] defines conceptual entities for service modeling. It is an effort to build the Web Service Modeling Framework (WSMF) that provides the appropriate conceptual model for developing and describing web services and their composition. Its philosophy is based on the following principle: maximal de-coupling complemented by a scalable mediation service. Web-Service Modeling Ontology (WSMO) [10] provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automation of discovering, combining and invoking electronic services over the Web. It has its foundations in WSMF but it defines a formal ontology to semantically describe web services. The Web Services Modeling Language (WSML) [4] provides a formal syntax and semantics for the WSMO based on different logical formalisms.

In general, approaches utilizing UML such as in [1], [2] and [8, 7, 3] make the creation of transformation rules challenging. On the other hand, non-UML-based modeling approaches as in [10, 6, 4] provide the conceptual models for web services but have limitation for code generation support. Existing DSL approaches such as [11] is rather complex. For model-driven development of web services, there is the need for simple and natural syntax that provide the support for code generation. Language such as SWSM is an effort aiming at making this possible.

Conclusions

MDD approach can be applied to web services in order to increase the resilience of implementations, as web services technologies change and evolve. The result of this research brings up the design theory and methodology for implementing and utilizing a domain specific language for model-driven development of web services. Adopting domain specific languages, such as the one we introduce, can increase productivity and ease the burden of development of web services as the backbone on SOA systems. SWSM also reduces the cost implied in maintaining the systems and provides a solution to software reuse.

SWSM was written at a good abstraction level. This improves code readability and makes program integration easier. SWSM enables users without experience in programming at a higher level to focus only on knowledge of their concerned domain. Hence under this approach, it is possible for different stakeholders such as business experts and IT experts to model web services during early stages of web services design. Another advantage of SWSM for modeling is the ability to generate more verification on the syntax and semantics than a general modeling language. This can reduce errors on the testing or debugging process. However, we also need to point out that this approach has several drawbacks. There is an extended learning curve for a new language, even though SWSM as a domain specific language proves to be a lot easier to learn than a general programming language. Additionally, as a general language is adopted by more people, it could be more feasible to find staff capable of solving the problem using their language knowledge. There are also spaces for improvement in the syntaxes of SWSM.

In practice, approach using SWSM can be applied to the web service development process in various environments. As members of our team are working with companies in the top global Fortune 500 dealing with large-scale web services for financial services and telecommunication industries, the outlined approach has started to gain adoption and initially has been applied successfully. Our future work will continue on the enhancement of SWSM and introduce a suitable model transformation for both SOAP and Representational State Transfer (REST) web services.

Acknowledgement

This work has been supported by the Department of Computer Science and Engineering, Faculty of Electrical Engineering and by the grant of Czech Technical University in Prague number SGS14/078/OHK3/1T/13.

References

[1] Benguria, G., Larrucea, X., Elvesæter, B., Neple, T., Beardsmore, A., Friess, M.: A Platform Independent Model for Service-oriented Architectures. Enterprise Interoperability, pp. 23-32, 2007

[2] Bezivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA Approach for Web Service Platform. In Proceedings of Enterprise Distributed Object Computing Conference, pp. 8-70, 2004

[3] Bordbar, B., Staikopoulos, A.: Automated Generation of Metamodels for Web Service Languages. In Proceedings of Second European Workshop on Model-driven Architecture, 2004

[4] De Bruijn, J., Lausen, H. (Ed.): The Web Service Modeling Language WSML. W3C Member Submission. Retrieved December 14, 2012, http://www.w3.org/Submission/WSML/, 2005

[5] Deursen, A., Klint, P.: Little languages: Little maintenance. Journal of Software Maintenance, pp. 75-93, 1998

[6] Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF, Electronic Commerce. Research and Applications, pp. 113-137, 2002

[7] Frankel, D., Parodi, D.: Using Model-driven Architecture to Develop Web Services, IONA Technologies (2nd Ed.), 2002

[8] Heckel, R., Lohmann, M, Thöne, S.: Towards a UML Profile for Service-Oriented Architectures. MDAFA, 2003

[9] Krueger, C. W.: Software Reuse. ACM computing Surveys, pp. 131-183, 1992

[10] Lausen, H., Polleres, A., Roman, D. (Ed.): Web Service Modeling Ontology (WSMO) W3C Member Submission. Retrieved December 14, 2012, http://www.w3.org/Submission/WSMO/, 2005

[11] Maximilien, E. M., Wilkinson, H., Desai, N., Tai, S.: A Domain Specific-Language for Web APIs and Services Mashups, In Proceedings of 5th International Conference on Service Oriented Computing (ICSOC), LNCS 4749, Springer-Verlag, pp. 13-26, 2007

[12] Mernik, M., Heering, J., Sloane, A. M.: When and How to Develop Domain Specific Languages. ACM Computing Survey, Vol. 37 No. 4, pp. 316-344, 1999

[13] Oberortner, E., Zdun, U., Dustdar, S.: Domain specific Languages for Service-oriented Architectures: An Explorative Study. Towards a Service-Based Internet. Springer-Verlag Berlin, Heidelberg, pp. 159-170, 2008

[14] Object Management Group (OMG): Meta Object Facility (MOF) Core. Retrieved March 20, 2012, http://www.omg.org/spec/MOF/2.4.1/, 2012

[15] Object Management Group (OMG): The Architecture of Choice for a Changing World. Retrieved March 20, 2012, http://www.omg.org/mda, 2012

[16] Qafmolla, X., Nguyen, V.: Automation of Web Services Development Using Model-driven Techniques. In Institute of Electronics Engineers, The

2nd International Conference on Computer and Automation Engineering (ICCAE 2010), pp. 190-194, 2010

[17] The Enterprise Architect: Building an Agile Enterprise. Retrieved November 18, 2012, http://www.theenterprisearchitect.eu, 2012

[18] Web Modeling Group - Czech Technical University in Prague (WMG): SWSM Language. Retrieved March 15 2013, from http://webmodeling.net, 2013

[19] Wikipedia: Domain Specific Language. Retrieved January 12, 2012, from http://en.wikipedia.org/wiki/Domain specific_language, 2012

[20] Wile, D. S.: Lessons Learned from Real DSL Experiments. Science of Computer Programming, 51, pp. 265-290, 2002

[21] World Wide Web Consortium (W3C): Web Services Architecture. Retrieved December 20, 2009, http://www.w3.org/TR/ws-arch, 2013

[22] Yu, X., Zhang, Y., Zhang, T., Wang, L., Hu, J., Zhao, J., Li, X.: A Model-driven Development Framework for Enterprise Web Services. Information Systems Frontiers, pp. 391-409, 2007