Automating Web Service Development Using a Unified Model

Ngoc Bao Bui, Liming Zhu, Yan (Jenny) Liu, Vladimir Tosic, Ross Jeffery
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
Managing Complexity Research Group - Sydney (ATP), NICTA, Sydney, Australia

Abstract

Web service standards are being developed in a loosely coordinated and constantly evolving manner and there is a lack of Web service modeling approaches that efficiently reflect the status of the standardization. Consequently the development and deployment of Web services tend to be ad-hoc and platform-oriented. This introduces potential interoperability issues and maintenance overhead. This paper proposes a model-driven framework that includes a Web service modeling language describing functionality and non-functional properties of service-oriented applications in unified models. This Web service modeling language is based on a Web service meta-model extracted directly from the WS-* standards. We developed a corresponding tool that generates code stubs, configurations and deployment heuristics, along with standard-based artifacts from models. We conducted a real-world case study to validate our approach.

1. Introduction and motivation

Service Oriented Architecture (SOA) has drawn much interest in both the research community and the industry. It is an architectural style that instructs to build distributed systems as a collection of interacting and loosely coupled services. Over the last few years, Web service technologies became the main choice for realizing SOA. They are based on a stack of industrial standards, among which SOAP, WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery and Integration) are usually considered the core. These three standards enable to build a service that can be described, discovered and accessed using XML artifacts over the Internet. To support the development of Web services that can address specific non-functional requirements, the second generation of WS-* standards mainly concern non-functional system properties (e.g., security, reliability), as well as orchestrations and collaborations of existing Web services. Along with Web service standards, various service middleware platforms—such as Axis, Axis2, ASP.NET with Web Service Enhancement, Window Communication Framework (WCF)—are introduced to provide a development and deployment environment for Web services with various degrees of standard conformance.

This trend of Web service technologies leads to the increasing demand on an efficient model-driven Web service development approach, which models services at different levels, conforms to different standards and effectively transforms the models to infrastructure-specific artifacts during implementation and deployment. However, the current set of Web service standards is not developed in an explicit and consistent manner for such a model-driven paradigm, due to many limitations:

- Various Web service standards are developed only in a loosely coordinated fashion, under different sub-committees within a standardization body or even different standardization bodies, at different times. While this separation of concerns does serve well for certain goals, the connections and consistencies among elements of different standards are often lacking precision and clarity for effective model-driven development.
- The relative rigor of these standards often relies on syntax and semantics of XML schemas. While XML is a universal format for representation and exchange, it is not ideal for comprehension in both modeling and development aspects. The semantics is often given in textual description parts of the standards, which are open to informal interpretations. This limitation also prevents highly effective model-driven automation.
- Most Web service standards go directly into the details of a specific version of a standard format without a proper meta-level presentation of the core concepts. With rapid, continuous and separate evolution of these individual standards, adaptability and portability of services built on top of these standards can suffer from lacking of a more
consistent regulating meta-model. Although both OASIS (Organization for the Advancement of Structured Information Standards) and W3C (World Wide Web Consortium) developed Web service reference architectures [1, 2], they only provide some controlled vocabularies and general concepts rather than explicit meta-models.

This problem is further exacerbated by the fact that standard conformance among Web service platforms varies significantly. It is common to see Web service standards being supported in heuristic ways that defeat the purpose of standard-based development. Standard-based artifacts are not equal to standard-based development. To address the above problems, we propose an integrated software engineering solution:

1. A standard-based, consistent and unified Web service meta-model. It captures the core concepts of Web service modeling and development, which we faithfully extracted from a collection of the existing WS-* standards. For example, it reflects concepts such as Policy, Policy Assertion, Subject and Domain, which are essential elements of various WS-* standards, but usually not reflected in ad-hoc service modeling languages. Our meta-model provides a consistent basis for modeling language derivation and evolution. This is different from current model driven service development approaches that have been done without a clear meta-model foundation.

2. An evolvable Web service modeling language derived from the meta-model. It directly reflect the concepts we extracted from the WS-* standards and can be used for modeling both functional and non-functional aspects of Web services. For example, Policy and Policy Assertion can be used to specify Service Level Agreements (SLA). The evolution of the language constructs can be managed more easily, due to their relationships with the unified meta-model. This is different from the current approaches, which simplify standard compliance into format level compliance in generated artifacts rather than explicitly reflecting standard-based constructs on the modeling level.

3. A language workbench that facilitates and demonstrates uses of our modeling framework.

This solution can be realized by Model Driven Development (MDD) techniques. MDD is a software development paradigm in which applications are specified and generated from models. Currently, there are two general MDD approaches, Model Driven Architecture (MDA) [3] and Domain Specific Modeling (DSM) [4]. MDA was proposed by the Object Management Group (OMG) and is based on a set of existing OMG standards such as the Meta-Object Facility (MOF) and the Unified Modeling Language (UML). Applying MDA requires extending the current UML or developing a new modeling language based on MOF, which may inherit some unnecessary constructs of UML or MOF. Different from MDA, DSM does not conform to any standard. Applying DSM requires the use of one or more Domain Specific Languages (DSL) that are designed specifically for a particular domain, usually from scratch. The design of a DSL is based on a language workbench environment such as Microsoft DSL tool [5] or MetaEdit+ [6]. In this work, we choose Microsoft DSL as the language workbench environment for specifying a unified meta-model and a set of modeling language constructs for Web service development.

We applied our approach to a real-world e-business service scenario involving a reference implementation for Australian lending industry standards. This case can particularly benefit from our approach because there is a need to: 1) maintain the reference implementation to continuously demonstrate the benefits of different WS-* standards; 2) clearly communicate the design of the reference implementation to the standardization body through architectures manifested in models; and 3) map the reference design into different platforms. All this has to be done effectively, with certain degree of model-driven automation. This case demonstrated that our approach effectively meets the above requirements. The case details are presented later in this paper.

This paper is organized as follows. Section 2 reviews some past model-driven service development approaches. Section 3 outlines our general approach. The details of our Web service meta-model, the corresponding modeling language, and the language workbench are discussed in sections 4 and 5. Section 6 presents our real-world case study. The last two sections evaluate our approach and conclude the paper.

2. Related work

There were many attempts to apply MDD to service oriented development and most of them favour MDA-based approaches.

Johnston and Brown [7] proposed a UML profile as a language for modeling services, service oriented architectures and service oriented solutions. The language allows specifying functions offered by a service and interaction protocols between service providers and consumers. Also based on UML, Grønmo et al. in [8] recommends a model-driven Web service development approach that leverages general UML constructs instead of designing a new language. Another approach is shown in [9], which introduces a WSDL-dependent UML profile for modeling individual services. The UML profile elements and their relations directly correspond to WSDL components such as service, port type, port, and message. These approaches allow models to be transformed into WSDL documents. One common limitation of these approaches is that they focus on modeling and producing individual services rather than service oriented applications.

Similar works have been done for modeling non-functional properties (NFP) of services. Grundy et al. [10]
use an aspect-oriented approach to model cross-cutting and quality aspects of a service. Ortiz and Hernández [11, 12] recommend two different profiles for modeling services and their non-functional properties. The NFP profile includes the abstract stereotype Extra-Functional Property that can be specialized to represent various non-functional properties. Although allowing modeling non-functional properties in a loosely coupled manner, the profile serves as an indicator for code generation process rather than for design. Wada et al. [13] also propose a UML-based service profile in which non-functional properties are modeled as tagged values of profile elements such as service or connector. However, it mainly focuses on aspects related to message transmission and message semantics, such as synchrony, timeout or queuing. Generally, the works on modeling service non-functional properties are limited.

In general, the above modeling approaches try to simplify web service development by means of models and automation. However, there are some aspects that are not addressed adequately in these approaches:

- Faithful reflection of WS- standards.
- Unification of functional and non-functional aspects in a meta-model and a corresponding modeling language.
- Use of a unified model for design, implementation and deployment (rather than separate models).

The approach proposed in this paper covers all above issues and aims to provide uniform guidelines for Web service development.

3. Our model-driven framework

The proposed model-driven Web service development is based on DSM rather than MDA. The central part of the approach is a meta-model and a corresponding Web service modeling language that expresses functionality of services and quality (i.e., non-functional) attributes, such as security. Although the existing approaches model similar concepts, they do not extract concepts directly from Web Service standards and, most importantly, do not link the WS-* standards. Our approach determines the terminology and concepts for different aspect of Web services. Extracting from Web service standards not only ensures adequacy of the modeling language, but also promotes standardization on the modeling level rather than the artifact level. The specified models can be transformed into any vendor-specific service middleware platform.

Our approach is summarized as follows:

- A Web service meta-model is abstracted from the main existing Web service standards. Connections and reconciliations are made among core concepts in a standard or across different standards and reflected in the meta-model.
- A Web service modeling language is then derived from the model considering a number of factors, including design, implementation and deployment needs, and language evolution needs and usability issues.
- The language then allows developers to specify structural specification, behavioral specification and non-functional attributes through policies. The information is general and not bound to any platform. It is possible to generate WSDL files and policy documents directly from these models. Developers can implement the business logic for the service.
- The generic model can also be transformed into target platform models and analysis models.

This meta-model is fairly stable – it has to be updated only when the Web service standards evolve. This enables iterative development of Web services from definition to implementation and deployment.

The following sections will discuss the Web service meta-model and the Web service modelling language and related issues.
4. The Web service meta-model

Our meta-model is comprised of concepts faithfully extracted from the main Web service metadata specifications in terms of acceptance in practice WSDL, WS-Policy [14], WS-PolicyAttachment [15] and domain specific policy specifications such as WS-SecurityPolicy [16]. These concepts provide a unified framework to describe and specify functional and non-functional requirements of Web services in a standard and consistent way. Figure 1 presents a part of the overall unified meta-model to illustrate our modeling approach. The key elements and their relationships are:

- **Service**: represents a Web service. A Web service includes a set of endpoints (see **Endpoint** in Figure 1) for interaction with its consumers. Each endpoint provides the exact location and the permitted protocol and encoding to access the service.
- **Interface**: denotes a group of logically related operations (see **Operation** in Figure 1) for interaction with its consumers. To access the service, a service may implement more than one interface and an interface can be implemented by many services.
- **Message**: denotes a data container that carries essential information in a specific interaction between service providers and consumers. A message can be a request, a response or a fault message in an operation.
- **PolicyAssertion**: a policy assertion expresses a domain-specific requirement (such as security, reliability) or a middleware-specific requirement. All policy-related concepts in our meta-model are derived from WS-Policy and associated standards. The policy assertions are taken directly from the domain specific policy languages, such as WS-SecurityPolicy, and integrated into the meta-model.
- **PolicyAlternative**: denotes a logical grouping of non-functional requirements and a policy is comprised of policy alternatives. The **Policy** and **PolicyAlternative** concepts help specify non-functional properties and attach them to a Web service in an organized and consistent way.
- **Policy**: represents a set of non-functional requirements applied to a particular policy subject.
- **Subject**: denotes an entity that a policy can be attached to. It relates policies with the other concepts in our meta-model.

5. The Web service modeling language

Our Web service modeling language organizes the set of meta-model concepts into a set of modeling constructs. It enables developers to utilize the formal concepts specified in the meta-model to design Web service oriented solutions. The language design focuses on means to manifest service concepts effectively and flexibly. An individual service concept represented as an element or an association in the meta-model is now
modeled as a standalone language element or an element associated with others, depending on its semantics and relationships with other concepts. The alignment between the meta-model and the modeling language is:

- **Service, Interface** and **Policy** are represented as standalone modeling elements since they are the first-level concepts that other concepts depends on.
- **Message**, which may be referenced by different operations, and **PolicyAlternative**, which may be reused in different policies, are designed as a stand alone modeling constructs.
- **Operations** are grouped into an **Interface** and they are modeled as attributes inside **Interface** elements. This is shown in Figure 2. Similarly, **PolicyAssertions** are modeled as attributes of **PolicyAlternative** elements.

**Figure 2. Representation of operations**

- An Endpoint is modeled as a **Service** port instead of a standalone element. This is due to its dependence on the element **Service** at the conceptual level. Modeling **Endpoint** as a port rather than an attribute inside the **Service** element actually reflects the role of an **Endpoint** as an access point to a **Service** in the model.
- **InputMessage, OutputMessage** and **FaultMessage** denote the roles of messages referenced by operations. Therefore, they are designed as properties inside an **Operation**. Examples are shown in Figure 3.

**Figure 3. Representation of input, output, and fault messages**

- There is no explicit modeling construct for **Subject** since this concept indicates the roles other concepts play in relationships with **Policy**.

The Web service meta-model and the Web service modeling language are implemented in the language workbench developed using the Microsoft DSL toolkit. It also provides tool support to generate implementation and deployment artifacts. The detailed example in the next section illustrates the use of the language workbench.

6. Example: LIXI business scenario

We have been working with a leading Australian e-business industry standardization body, Lending Industry XML Initiative (LIXI), which serves the lending industry in Australia. During a loan application process, a lending organization has to determine the market value of the loan-related property. This process is called “valuation” and is usually conducted by a valuation firm. Many such firms have an e-business interface that is a company-specific Web portal for uploading documents related to valuation requests. In order to improve interoperability across the Australian landing industry, we were asked to supply a Web service reference implementation based on the current LIXI standard and Web service standards.

A valuation service usually has a number of policies at different levels. At the transportation level, reliable messaging and security policies can be associated. At the messaging level, we exposed coordination logic through abstract BPEL (Web Services Business Process Execution Language). At the application level, LIXI specific policies are exposed. Some policies indicate requirements, such as revision numbers and pre-negotiated fees. Some other policies indicate capabilities, such as delivery priorities.

Figure 4 shows the model of a Web service based implementation for the property valuation business process and its attached policies. In the model, the valuation process is exposed as “ValuationA” service that can be accessed via two endpoints. The functionality of the “ValuationA” service is expressed as operations in interfaces “ValuationService” and “ValuationServiceCallback”. The quality attributes of the Web service are defined by the “ValuationPolicies” element comprised of **PolicyAlternatives** (Figure 5). Policy assertions provide the details of each policy (security, reliability or platform specific policy) and can be dragged and dropped from the toolbox (Figure 6).
Figure 4. Mapping between the model and XML based documents
Figure 5: Policy Alternatives

Figure 4 demonstrates the mapping from a graphical model into a WSDL with embedded policies. The alternatives grouped into “ValuationPolicies” are transformed into different policy expressions. The connection between the service “ValuationA” and “ValuationPolicies” is translated into the three policy references inside the WSDL definition. The user-defined properties of each policy assertion provide sufficient information to generate WS-Policy documents.

7. Evaluation

We considered three types of evaluation methodologies for our approach: a real-world case study, a modeling language usability evaluation, and a systematic comparative case study with other modeling approaches. We have so far conducted evaluations from the first two types, while evaluations from the third type are part of our ongoing work.

As shown in the previous section, we applied our approach to our Web service reference implementation for the valuation sub-process in LIXI standards. The benefits of our approach align well with the goals of the LIXI reference implementation in terms of coping with standard evolution, demonstrating usage of Web service technologies, exposing policies at all levels, improving design communication and reducing development effort. We observed a number of benefits and differences from the other approaches:

- The usage of standard-based concepts through modeling constructs significantly improves communication since it leverages developers’ familiarity with existing standards.
- Using policies as a way to cover both requirements and capabilities in the same model is well-received compared to using aspects and ad-hoc non-functional requirements representations from the academic community. Policies also resonate with business analysts (in addition to developers).
- Decoupling from platforms and products in the model-driven approach has been perceived as a good way of reducing vendor lock-in and standard evolution risks. Many existing MDD approaches pollute high level models with product and platform specific concepts. Standard-based modeling constructs are considered a good solution.

We also evaluated usability of the modeling language through expert opinions. This evaluation is based on the following criteria:

- Expressiveness: How much does our modeling language support describing required features of a service? Since the concepts in the meta-model were derived directly from the Web Service standards, we were not surprised to find that our language can cover the essential details of a service.
- Readability and scalability: Is a model easy to read and comprehend, even when it is large? The most challenging issue people encountered before is the modeling of cross-cutting non-functional requirements. They often make elements on a model cluttered. The introduction of levels of abstraction through policy containers, policy assertions and multiple ways they can be attached to a
service significantly relieves the cluttering symptoms. Readability and comprehension were considered good by participants in our evaluation.

Overall, the language usability evaluation has further confirmed the general ability of our approach, beyond the findings of our real-world case study.

8. Conclusion

This paper presented a standard-based model-driven approach for Web service development. It consists of a meta-model and corresponding modeling languages. Model transformation facilities are also provided for generating artifacts and other models for further analysis. The approach has two unique contributions:

1) A unified meta-model directly reflecting the current Web service standards provides a consistent basis for modeling language derivation and evolution. This is critical for standard-based development at all levels (rather than only the XML format level).

2) A service modeling language that can be used to model functional and non-functional aspects of Web services in a unified, standard-based and platform-independent manner. It further explicitly reflects standard-based constructs on the modeling level.

The approach and the language were validated through a real-world case study and expert opinion based language evaluation. Both evaluations have received positive results, but also identified some issues in the language design. For example, even though the language tries to faithfully reflect standard-based concepts, the standards leave certain interpretation space. On the other hand, there are multiple types of users for the language. Some language design trade-offs arise due to this flexibility and potentially conflicting requirements. In our ongoing work, we are resolving these trade-offs with the language users. Further evaluation is also an ongoing activity.

Our service modeling framework is still under development and should be considered as a research prototype. However, we work towards that in the near future it is mature enough for distribution to the broader community and for wide use in practice.

References