A Methodology for Constructing WS-Policy Assertions

Bernhard HOLLUNDER, Matthias HÜLLER, Andreas SCHÄFER
Department of Computer Science
Furtwangen University of Applied Sciences
Robert-Gerwig-Platz 1, D-78120 Furtwangen, Germany
Email: hollunder@hs-furtwangen.de

ABSTRACT

Today, Web services play a significant role in communication between heterogeneous systems. In order to describe Quality of Service (QoS) properties for Web services, the WS-Policy specification has been introduced by the W3C. WS-Policy is widely accepted in the Web services space and is supported by almost all Web services platforms and application servers. Other well-known specifications such as WS-SecurityPolicy apply WS-Policy to formalize domain-specific QoS properties with the help of so-called assertions. Although there has been much research on WS-Policy, there does not exist a definition of a comprehensive process for designing, creating, implementing and deploying WS-Policy assertions. In this paper, we elaborate a methodology comprising the necessary steps required to construct custom-designed assertions for WS-Policy. We not only describe for each step the respective activities, results, and recommendations, but also distinguish the different stakeholders. The methodology presented strongly facilitates the usage of WS-Policy and is a further important step towards improving the developing of Web services with well-defined QoS properties.

Keywords: Methodology, Web Services, WS-Policy, WS-Policy Assertions, Quality of Service (QoS)

1. INTRODUCTION

In the course of the last few years, service-oriented architectures (SOA) are a widely used paradigm for structuring distributed enterprise applications. In this context, services are often characterized as reusable units of functionality, which are aggregated to automate a particular task or business process. Based on well-known protocols and standards such as HTTP, SOAP, and WSDL, Web services are the prevailing technology for implementing services. In fact, almost all application servers from technology providers such as IBM, Microsoft, and Oracle/Sun include development and runtime support for Web services.

When employing Web services in enterprise business applications, often "pure" Web services are not sufficient. This is because at the business level Quality of Service (QoS) properties such as message reliability, availability, scalability, security, pricing and accounting are needed. The importance of QoS-aware Web services has been stressed elsewhere (see e.g. [1], [2]).

With the WS-Policy specification [3], application providers can formally express QoS properties for the services to be developed and deployed. Basically, QoS properties are formalized with so-called assertions. A set of assertions forms a policy, which is part of the interface contract of the Web service. While it is the responsibility of the service developer to specify a suitable WS-Policy description, it is up to the Web service runtime environment to validate the policy, i.e. to check whether the policy assertions are really satisfied during service processing. This strategy strongly supports the well-known *separation of concerns* principle (see e.g. [4]): The Web service implementation encodes the core business functionality, whereas QoS properties are configured in a declarative manner with WS-Policy.

For example, suppose the service provider defines a (security) policy which requires the service parameters to be encrypted during transmission. The runtime environment will validate if the service invoker has really encrypted the parameter values. If not, the service request will be rejected; otherwise the request will be forwarded to the service implementation (after decrypting the parameter values).

WS-Policy itself does not come with concrete assertions. Instead, related specifications (such as WS-SecurityPolicy [5]) introduce specific assertions covering domains such as security and reliable messaging. The high acceptance of e.g. WS-SecurityPolicy especially demonstrates the applicability and usefulness of WS-Policy.

From an application development point of view it would be quite helpful to have a wider repertoire of assertions. In fact, it should be possible to construct further, custom-designed assertions covering project-, business-, or technology-specific phenomena. Thus, the separation of concerns principle for QoS properties should not be restricted to the already addressed domains.

Observe that WS-Policy has been designed in such a way that it allows the creation of arbitrary WS-Policy assertions. However, the construction of new assertions turns out to be difficult, mainly for the following two reasons:

- The creation and usage of custom-designed assertions is of high complexity covering different activities, results, and stakeholders.

- There is a lack of definition of an adequate process for constructing custom-designed assertions.

The target of this paper is to reduce the complexity of constructing WS-Policy assertions by clearly defining a comprehensive process. To get closer to this target, we will identify the steps required for the overall process. For each step, we will describe the respective activities and results. Moreover, the elaborated process comprises guidelines with concrete instructions for the stakeholders involved.

The methodology presented in this paper strongly facilitates the usage of the WS-Policy technology. As a consequence, QoS properties for the Web services under development need not be hard-coded, but can be declaratively specified with suitable WS-Policy assertions. The main advantages of this approach are:

- Reduction of the Web services source code complexity due to *separation of concerns*.
- Higher degree of flexibility to react on changing QoS requirements.
- Increased reusability of Web services with QoS requirements for different deployment settings.

To sum up, the methodology described facilitates the development of robust Web services by properly applying WS-Policy.

The paper is organized as follows: we will start by introducing fundamental technologies, terms and definitions required for a basic understanding of this paper. Then, in Section 3 we will identify the phases and stakeholders of the WS-Policy assertion construction process. Related work that has inspired the methodology elaborated is presented in Section 4, followed by a conclusion in Section 5.

Due to limited space, the scope of this paper does not give an introduction to WS-Policy. Hence, we assume an understanding of the basic concepts of WS-Policy (see e.g. [6], [7]).

2. BASIC CONCEPTS

Web Services Description Language (WSDL) is the standard formalism for expressing interfaces of Web services. Part of the interface description is the name of the service, its parameter types and faults. This information is utilized by a service consumer to construct SOAP messages that are exchanged with the service implementation. A SOAP message consists of a body, containing the payload of the message (including the current parameter values of the request), and an optional header, containing additional information such as addressing or security data.

WSDL itself does not cover QoS properties for services. However, a WSDL description can refer to WS-Policy descriptions that formalize the QoS properties such as performance, availability, security, and usage costs.

Figure 1 illustrates the structure of a Web service communication scenario with a WS-Policy description attached to a WSDL file.

In order to take into account the requirements imposed by WS-Policy descriptions during service invocation, typically the content of a SOAP request will be manipulated. Web service runtime environments provide so-called (policy) handlers, which have access to and can modify SOAP requests. On both service consumer and service provider side handlers can be installed.

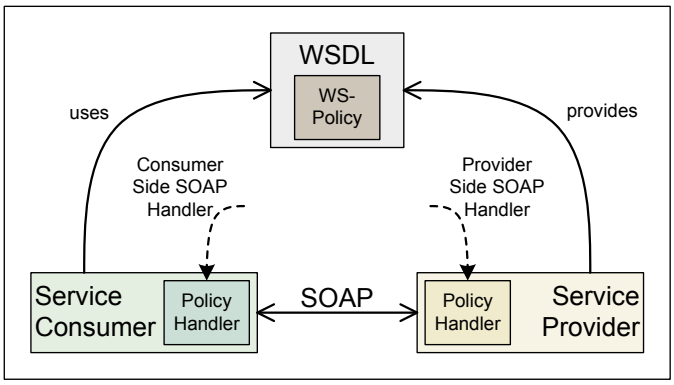

Fig. 1. Structure of Web service communication.

To give an example, suppose the service will only accept a request if the service consumer also transmits a time stamp in the SOAP message. To formalize this requirement, the service provider inserts an `IncludeTimeStamp` assertion into the WS-Policy description. A potential service consumer has access not only to the WSDL, but also to the attached policy. In order to fulfill the `IncludeTimeStamp` requirement, the service consumer must extend the SOAP request with a time stamp. More precisely, a client policy handler puts a current time stamp into the SOAP header before transmitting the SOAP message. On server side, a corresponding handler checks whether a time stamp has really been included or not. In case of the assertion being satisfied, the request will be handed over to the service implementation, otherwise a fault message is created and is sent back to the consumer by the policy handler.

3. ASSERTION CONSTRUCTION PROCESS

Figure 2 visualizes the main result of this paper: a comprehensive process for the construction and usage of WS-Policy assertions. As we will argue in Section 4, only some of the activities have already been identified by other authors. In addition, a comparable composition of all relevant activities has not yet been elaborated.

Basically, the overall process can be divided into three separate phases:

A) *Assertion creation*: This phase covers the design and implementation of assertions. Roles involved: *assertion designer* and *assertion developer*.

B) *Service provider usage*: In this phase, the deployment and usage of assertions on the service provider side is addressed. Roles involved: *provider administrator* and *service developer*.

C) *Service consumer usage*: This phase describes the activities of the service consumer. Roles involved: *consumer administrator* and *service consumer*.

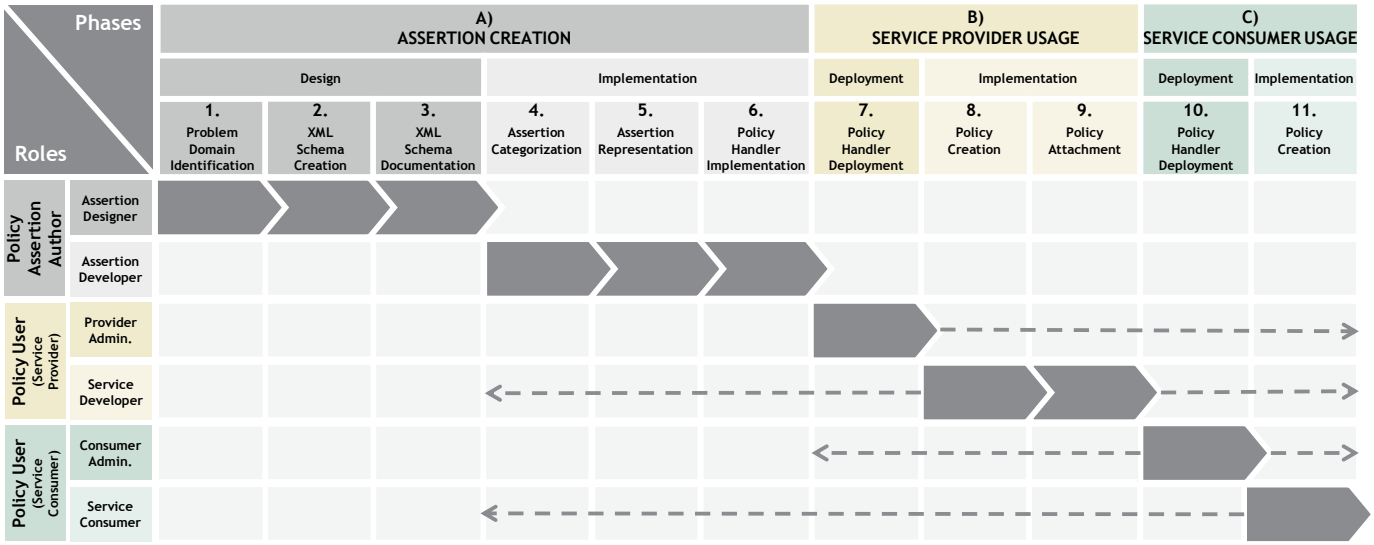

Fig. 2. Visualization of the assertion construction process.

The single steps in these phases usually proceed sequentially, even though some of the steps could also, in principal, be performed in a parallel manner. The steps that can be executed simultaneously are marked with a dashed line. This means these steps can be shifted on the timeline during the process. Next, all the single process steps are described.

We will illustrate the steps by means of a *calculator service* example. Due to space restrictions, this example must be kept simple. However, the presented methodology is also applicable to complex problem domains.

Assertion Creation

This phase is concerned with the design and the creation of assertions. After a need for a specific, not yet defined assertion has been recognized, an appropriate solution can be designed by an assertion designer who specifies and documents the assertion. These steps, which are independent from concrete Web service infrastructure products, are described in steps 1 to 3.

After the design of the assertion has been accomplished, an implementation in a concrete programming language and for a particular Web service runtime environment is needed. Hence, in steps 4 to 6 the assertion developer implements both the assertion class and a specific (policy) handler in some programming language.

1. Problem domain identification (assertion designer): The first task in the process is to structure the problem domain and to identify the elements required for the creation of the assertion's XML schema in step 2. The assertion designer should be very familiar with the problem domain in order to find suitable abstractions. In general, the result of this step should be an object model of the problem domain comprising possible assertion types. The abstractions can be visualized e.g. in a model of the Unified Modeling Language (UML) that introduces the key terminology. Such a model is quite helpful to derive the XML schema and assertion classes in the steps which follow.

Calculator example: Basic concepts in this scenario are (among others): *calculator* with operations (e.g. add, multiply) and attributes (e.g. registers), *numbers*, *precision*, and *overflow*.

2. XML schema creation (assertion designer): The terminology of the previous step is now refined to create an XML schema, that specifies the syntax, structure and data types of the assertion. The XML schema is used by the service developer in step 8 and by the service consumer in step 11 to compose policies.

Calculator example: We introduce two types for defining the range of numbers that can be processed by a particular calculator implementation. The following fragment is part of the XML schema for the sample calculator policy assertions.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace= ...
        xmlns:tns= ...>
    <element name="MinNumber" type="double" />
    <element name="MaxNumber" type="double" />
</schema>
```

XML schema CalculatorPolicy.xsd

3. XML schema documentation (assertion designer): Since the assertions are typically not defined by the actual users (e.g. service developers), proper documentation is required. This documentation should describe the problem domain, the semantics of the assertion, and the relevant technical details. In additon to the XML schema, it can also be helpful to provide a more reader-friendly description, such as an XML outline (see [8], section 5.3).

Calculator example: Depending on the concrete implementation of a calculator service, the sets of representable numbers may differ. For instance, a simple calculator may represent numbers with 2 bytes, while a scientific version is able

to deal with numbers ranging from -2^{31} to $2^{31}-1$. A calculator policy defines the range by means of `MinNumber` and `MaxNumber` assertions. If a client (i.e., a service consumer) wants to add two numbers whose sum is greater than the maximum representable number, the provider policy handler should reject the request.

4. Assertion categorization (assertion developer): WS-Policy assertions can be divided into two categories. As elaborated in [9], the background for this categorization is of technical nature. Some assertions can only be evaluated properly, if on service consumer side a policy handler is installed. Such assertions are said to be of type 2, otherwise they are called type 1 assertions. An example for a type 2 assertion is `IncludeTimeStamp` of Section 2, because a client handler has to include a time stamp into the SOAP message header by modifying the outgoing SOAP request.
Calculator example: Both `MinNumber` and `MaxNumber` are type 1 assertions. This is due to the fact that the service provider policy handler can determine—by inspecting the incoming parameter values—whether an overflow error will occur. No additional information must be provided by the client.

5. Assertion representation (assertion developer): The WS-Policy specification defines the structure of assertions and policies, but it does not prescribe how to represent assertions in a specific programming language. Typically, assertions are implemented as Java or C# classes, which must be derived from an abstract base type or interface. As this type is not standardized by the WS-Policy specification, but provided by WS-Policy implementations, there is a dependency on a specific WS-Policy implementation and its programming language. It is good practice to derive the stubs for the assertion classes from the UML model or the XML schema created in the first two steps.
Calculator example: When using the WS-Policy implementation Apache Neethi [10] the assertion classes `MinNumber` and `MaxNumber` have to implement the `org.apache.neethi.Assertion` interface.

6. Policy handler implementation (assertion developer): This step comprises the implementation of a policy handler. As shown in Figure 1, policy handlers can be deployed on service consumer as well as on service provider side. Although every Web services platform supports a handler mechanism, no standards exist. In other words, each Web services implementation provides its own way how to implement and configure handlers. As a consequence, this step is dependent on the Web services product to be employed. The handlers have the following responsibilities: A policy handler on provider side has to inspect the incoming SOAP message to verify whether or not the defined policy is satisfied. If it happens to be satisfied, the request is passed to the service implementation; otherwise it generates a fault message and sends it back to the consumer. The main task of a consumer side policy handler is to extend the outgoing SOAP request by additional information according to the semantics of the type 2 assertion.

Calculator example: Suppose we want to implement a calculator policy handler for Axis2 [11]. We create a Java class which is derived from `AbstractHandler`. Its `invoke` method is called, when a service request is received at service side. The implementation of `invoke` inspects the service parameter values and checks that the numbers of the request do not produce an overflow error.

Together with step 5 the result of this step is a program library, which can be deployed into a specific WS-Policy platform to extend it with the required assertion logic.

Service Provider Usage

Now that the policy assertion design as well as the implementation in form of assertion classes and handlers are available, the Web service provider can proceed with the following tasks:

7. Policy handler deployment (provider administrator): The custom policy handler has to be deployed on service provider side to allow compliance verification of the Web service's policy. In contrast to policy handlers already delivered by most Web services infrastructures (WS-Security, for example), the policy handler of step 6 needs to be deployed manually to the infrastructure used on provider side. This step is performed by the administrator who knows the specialty of the server side infrastructure. Another point is that special requirements and additional hints should be given in the documentation.

8. Policy creation (service developer): In this step, the service developer creates a concrete policy containing custom-designed as well as predefined assertions. The syntactic representation of the assertions must conform to the respective XML schemata (see step 2). According to the WS-Policy specification, assertions can be composed with the help of the `Policy`, `All` and `ExactlyOne` operators to construct complex policies. If a parametrized assertion is used (i.e., an assertion with XML attributes and child elements), the specific values for the properties are also added. The result of this step is a WS-Policy description that defines the QoS properties of the service.
Calculator example: In the case of the concrete calculator policy, the assertion parameters are set such that 4 bytes numbers are supported.

```
<wsp:Policy
    xmlns:wsp="http://schemas.xmlsoap.org/ws
        /2004/09/policy"
  xmlns:cp=...
  xmlns:xsi=...>
  <wsp:ExactlyOne>
    <wsp:All>
      <cp:MinNumber>-2.147.483.648</cp:MinNumber>
      <cp:MaxNumber> 2.147.483.647</cp:MaxNumber>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Concrete sample calculator policy.

9. Policy attachment (service developer): Finally, the policy of the previous step is attached to the WSDL of the service.

The principles of this attachment are specified in [12]; however, no concrete API functions are defined. Instead, each Web services infrastructure provides a specific mechanism so that a service developer can programmatically extend a WSDL to include a WS-Policy description.

Calculator example: The following example shows a policy attached to a WSDL file. Through the `PolicyReference` element the policy is accessible by other parts of the WSDL as shown for the `binding` element. It would also be possible to have several policies in a single WSDL file.

```
<definitions ...>
  <wsp:Policy wsu:Id="CalculatorPolicy">
    <wsp:ExactlyOne>
      <wsp:All>
        <cp:MinNumber>-2.147.483.648</cp:MinNumber>
        <cp:MaxNumber> 2.147.483.647</cp:MaxNumber>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  ...
  <binding ...>
    <wsp:PolicyReference
          URI="#CalculatorPolicy"
          wsdl:required="true" />
    <operation ...> ...
    </operation>
  </binding>
  ...
  <service name="CalculatorService"> ...
  </service>
</definitions>
```

WSDL with embedded WS-Policy description.

The steps on service provider side do not have to run sequentially. Step 7 requires the implementation of the policy handler, so it has to be taken after step 6. Since steps 8 and 9 do not depend on step 7, they could also be performed parallel to these steps (see Figure 2).

Consumer Side Usage

On service consumer side we have identified two steps.

10. Policy handler deployment (consumer administrator): As argued in step 4 (assertion categorization), a client side policy handler needs to be installed only for type 2 assertions. Hence, this step does not apply for type 1 assertions. The deployment process on consumer side is similar to the handler deployment on provider side (cf. step 7). The handler usually consists of classes that include additional information to the SOAP request, according to the semantics of the assertion.

11. Policy creation (service consumer): The creation of a client side policy will only be required, if so-called policy intersection is applied. According to the WS-Policy specification ([3], p. 29) "policy intersection is optional but a useful tool when two or more parties express policies and want to limit the policy alternatives to those that are mutually compatible." Typically, policy intersection is applied on client side during service selection to identify the most appropriate service based on a given policy.

Calculator example: A client requests a calculator service that is able to deal with positive numbers from 0 to 65.535. To express this requirement the client policy is similar to the policy as described in step 8, except that the `MinNumber` (resp. `MaxNumber`) assertion has the value 0 (resp. 65535). Suppose there are two service implementations: the first one is able to deal with numbers from -32.768 to 32.767, while the second one can process numbers from -2^{31} to $2^{31} - 1$. Obviously, only the second service fulfills the requirements of the client. Hence, policy intersection should select the second service.

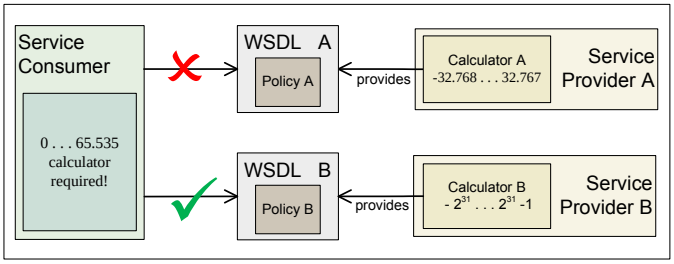

Fig. 3. Policy intersection and service selection.

As shown in Figure 2, step 10 depends on step 6, and the creation of the consumer policy in step 11 requires the XML schema of step 3. Both steps 10 and 11 could also be performed in a parallel manner.

4. RELATED WORK

There are books and papers which describe some steps towards constructing policy assertions, but so far a comprehensive process which includes all the tasks required has not been defined. Figure 4 illustrates tasks described in the literature. In the following publications, these approaches are described in more detail.

Erl, Thomas. Web Service contract Design & Versioning for SOA: Erl's book about contract design for SOA and Web services [13] includes a whole chapter about "Custom Policy Assertion Design". As shown in Figure 4, the description of custom policy assertions focuses on the design of the XML schema for WS-Policy assertions and the impact of parameters and attributes to policy intersection. The book also includes some considerations about the identification of the problem domain and how to add WS-Policy descriptions to a WSDL service contract.

Hollunder, Bernhard. WS-Policy: On Conditional and Custom Assertions: This paper [9] introduces a new WS-Policy operator for conditional assertions. While defining this operator the author implicitly describes some steps towards creating policy assertions. The concept of type 1 and type 2 assertions is introduced, which we used in our process in step "assertion categorization." Furthermore, the author describes the implementation of a policy handler with the Apache Axis2 framework (through extending Apache Neethi).

Mathes, Markus. WS-TemporalPolicy: A WS-Policy Extension for Describing Service Properties with Time Constraints: This paper [14] introduces WS-TemporalPolicy, a concept for describing time constraints to attach a validity period to policy assertions. It specifies the development process of

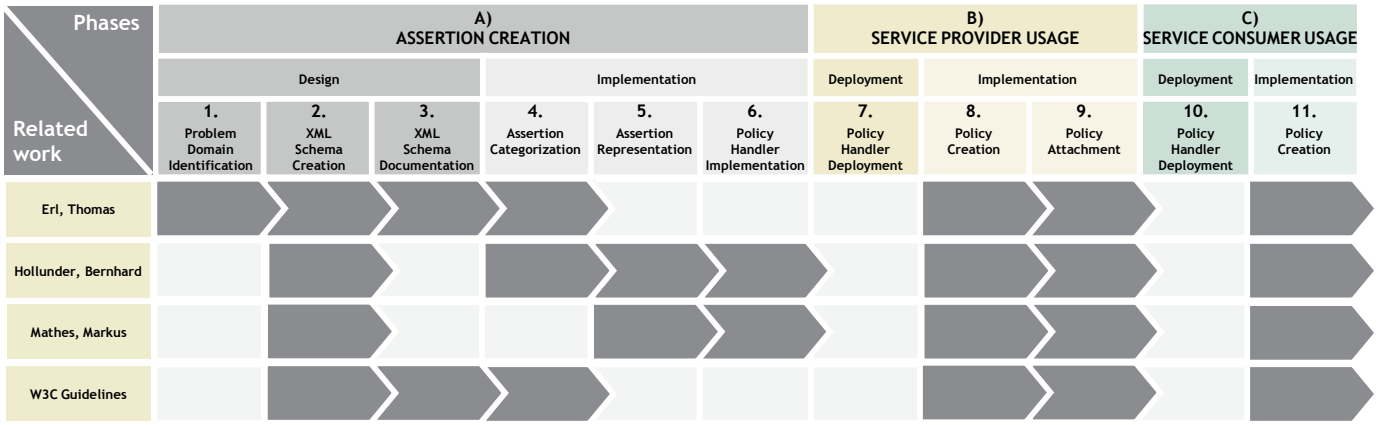

Fig. 4. Related work.

these temporal policies and thus describes some tasks for the creation of the XML schema, the concrete implementation with Axis2 and the implementation of policy assertion on service provider and service consumer side.

W3C. Guidelines for Policy Assertion Authors: This document by the W3C Working Group [8] is a complementary guide to using the WS-Policy specification. It aims at providing the best practices and patterns to follow when creating custom policy assertions; the impact on interoperability and policy intersection is also stressed.

5. CONCLUSION

Typically, the implementation of a Web service not only covers the core business functionality, but also realizes (explicitly or implicitly) QoS properties. Instead of hard-coding the latter features and hence violating the *separation of concerns* principle, it is of advantage to "externalize" the required QoS properties with the help of specific WS-Policy assertions.

In this paper we have presented a methodology which covers the necessary steps required to design, create, implement, and deploy custom-designed WS-Policy assertions. It clearly defines the responsibilities for the various activities and helps developers to apply WS-Policy in their Web services projects and applications – thus reducing code complexity and increasing reusability as well as flexibility of the solution. Though some of the activities have been described elsewhere, a comprehensive representation of the overall process has not been defined so far.

Due to lack of space we did not address interoperability issues in this paper. In order to apply WS-Policy, both the Web service provider and consumer must agree on a common set of assertions. To achieve this, the applicable assertions (together with artefacts such XML schema, handler implementation and documentation) may be stored in an assertion repository, which is populated by the assertion developers. The design and implementation of such an assertion repository is part of future work.

Another interoperability issue must be taken into account, if handlers are required on client side (which is the case for type 2 assertions). The assertion developer must provide specific handlers for those Web services runtime environments which are deployed on client side. Therefore, a client side handler should be designed in such a way that its core functionality has a high degree of portability.

The process as defined is generic and does not have any dependency on specific Web services platforms and WS-Policy implementations. Hence, an interesting extension of this work would be to adapt the process to specific infrastructures such as Windows Communication Foundations (WCF) [15].

Acknowledgments

We would like to thank Aileen C. S. Craig and the reviewers for giving us helpful comments. This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

REFERENCES

[1] Web Services Quality Factors. http://www.oasis-open.org/committees/wss.
[2] L. O´Brien, P. Merson, and L. Bass, "Quality attributes for service-oriented architectures," in *Proc. of the Intern. Workshop on Systems Development in SOA Environments*. IEEE Computer Society, 2007.
[3] Web Services Policy 1.5 - Framework. http://www.w3.org/TR/ws-policy.
[4] I. Sommerville, *Software Engineering*. Pearson Education, 2004.
[5] WS-SecurityPolicy 1.3. http://docs.oasis-open.org/ws-sx/wssecurity-policy/v1.3.
[6] J. Rosenberg and D. Remy, *Securing Web Services with WS-Policy*. Sams Publishing, 2004.
[7] Web Services Policy 1.5 - Primer. http://www.w3.org/TR/ws-policy-primer/.
[8] Web Services Policy 1.5 - Guidelines for Policy Assertion Authors. http://www.w3.org/TR/ws-policy-guidelines/.
[9] B. Hollunder, "WS-Policy: On conditional and custom assertions," in *IEEE International Conference on Web Services (ICWS'09)*. IEEE Computer Society, 2009.
[10] Apache Neethi. http://ws.apache.org/commons/neethi/.
[11] Apache Axis2. http://ws.apache.org/axis2/.
[12] Web Services Policy 1.5 - Attachment. http://www.w3.org/TR/ws-policy-attach/.
[13] T. Erl, A. Karmarka, P. Walmsley, H. Haas, U. Yalcinalp, C. K. Liu, D. Orchard, A. Tost, and J. Pasley, *Web Service contract Design & Versioning for SOA*. Prentice Hall, 2009.
[14] M. Mathes, S. Heinzl, and B. Freisleben, "WS-TemporalPolicy: A WS-Policy extension for describing service properties with time constraints," in *Proc. of the 32nd Annual IEEE International Computer Software and Applications Conference*. IEEE Computer Society, 2008.
[15] J. Löwy, *Programming WCF Services*. O'Reilly, 2007.