An Approach to Model, Configure and Apply QoS Attributes to Web Services

Ahmed Al-Moayed
Department of Computer Science
Furtwangen University of Applied Science
Furtwangen, Germany
ahmed.almoayed@hs-furtwangen.de

Bernhard Hollunder
Department of Computer Science
Furtwangen University of Applied Science
Furtwangen, Germany
bernhard.hollunder@hs-furtwangen.de

***Abstract*—Service-oriented architectures has become a commonly accepted solution for integrating enterprise applications around the globe. As the SOA environment grows, its complexity and the complexity of modelling, configuring and applying quality of service attributes to it increases. Though there are some tools supporting these activities, they are either limited to certain quality of service domains or dependent to specific development environments. In this paper, we elaborate a concept for managing quality of service attributes for Web services. In particular, our approach covers the modelling of arbitrary quality attributes based on a meta-model for quality attributes. It also covers the generation of a graphical user interface to configure the modelled quality of service attributes and the transformation of the modelled QoS attribute into policy descriptions. Finally, our approach outlines the assignment of the policies to the target Web services. This approach offers a solution, which reduces the cost and effort by the creation of QoS-aware Web services.**

***Keywords*-Service-oriented architecture, meta-model, model-model transformation, QoS-aware Web services**

I. Introduction

Service-oriented architectures (SOA) refers to a system architecture that provides a variety of different and possibly incompatible methods and applications as reusable services. As an enterprise may have a huge variety of service providers, which offer the same functionality, the services may differ in the non-functional requirements. A well designed application should have a precise functional goal and a set of non-functional requirements such as security and performance, which must be full-filled during execution time.

Applying quality of service (QoS) to distributed Web services is an important process as the demand for high quality Web services in terms of non-functional attributes raises. One way to apply QoS to Web services is by associating it with the Web Service Definition Language (WSDL). The WS-Policy Framework [7] is an OASIS standard, which allows Web services to express their capabilities, requirements and general characteristics in an XML form. However, in order to create such policies, a certain policy grammar knowledge is needed, which is not always acquired by Web service developers. In this paper, we present an approach, which offers an easy way to model quality of services for Web services and applies them to any Web service without being dependent on any kind of IDE (Integrated Development Environment), implementation language or previous knowledge of the implementation source code of a Web service.

We distinguish two kinds of developers: a QoS developer and Web service developer. The first one is responsible for modelling and implementing new QoS attributes; the latter one is responsible for applying the required QoS attributes to Web service once they have been developed.

There are a few tools, which allow the Web service developer to select certain QoS and apply them to a Web service. The problem with existing tools is, they are either hard to extend, mostly restricted to a certain policy domain, such as WS-SecurityPolicy [9] and WS-ReliableMessaging [8], or bundled with a specific IDE. To our best knowledge, there is no tool support, which offers both QoS developers and Web service developers the following features:

- A flexible, extended and simple meta-model for QoS modelling.
- An easy way to model and create QoS attributes for Web services and place them under a Web service developer's disposal.
- A dynamic graphical user interface, which allows the developer to easily and separately configure the modelled QoS attributes for each designated Web service.
- A QoS editor, which is not platform-, IDE- or language-specific. An editor, which supports SOA as an architecture and not Web service as a language-specific implementation.
- Automatic transformation of the configured QoS model into an adequate policy and automatically associate the created WS-Policy with the Web service.
- Association of the created policy description with the Web service.

While developing a Web service, an IDE is more than just a source code editor, which helps the Web service developer to write source code and to offer the developer code suggestions. It also automates many processes during development such as code compiling, generation of proxies and stubs and code deployment. However, there is only limited IDE support for developing QoS-aware Web services. This paper is one step to improve this support. It offers a solution, which automate and simplify the modelling, configuring and applying of QoS attributes to Web services. The Web service developer will no longer be required to directly configure the required set of low-level policy assertions and manually configure steps in order to apply the modelled QoS to Web services. This will reduce the developing effort and the costs of creating a QoS-aware Web service.

This paper is structured as follows: Section II gives an overview on our approach. Section III describes the QoS meta-model used in this paper. The QoS model will be explained in Section IV. Section V describes the dynamic QoS graphical user interface, which is derived from the QoS model. Section VI will shed a light on QoS model to policy transformation. Related work will be discussed in Section VII. Section VIII discusses future works. In the final section, we will conclude this paper.

II. APPROACH

The first component is the meta-model. It describes exactly how the QoS model is created or defined. There are many QoS meta-model proposals, which can be used to define and apply QoS models for Web services. Malfatti [6] introduced a suitable meta-model for our approach. It is simple, extensible, easy to understand and expressive enough to model arbitrary QoS attributes. The meta-model was created in Eclipse Modelling Framework (EMF) (Core) meta-model, a powerful tool for designing models and their runtime support.

The QoS model offers the QoS developer a way to model QoS attributes within certain QoS categories. These categories are already defined in the QoS meta-model. As we will see in Section IV, with this meta-model, we will be able to model different QoS attributes including some standardized QoS attributes such as reliable messaging and security. The QoS model is expressed in XML format. EMF, however, provides Java interfaces, Java implementation classes and a factory and package (meta data) implementation class, which provides support for building and modifying EMF models. The QoS developer will be able to use an EMF editor to manually add or remove QoS attributes.

Once a QoS model has been created, a graphical user interface will be automatically generated. The QoS model includes essential information on how the graphical user interface (GUI) should look like. Depending on certain elements in the QoS model, the GUI will be able to adapt to new changes. For example, depending on how many QoS categories are modelled, the GUI will generate a tab for each category. Within the same category, the QoS attribute will be presented. The GUI will also collect additional data from the Web service, such as the service endpoint interface methods, which will be needed for later steps in order to generate QoS policies. The main purpose of the GUI is to enable the Web service developer to easily configure the modelled QoS attributes and apply them to a Web service. Once this step has been done, the GUI will write the configured QoS attributes back to the QoS model in order to be transformed into a policy representation. For the moment, the GUI is implemented in Java as an Eclipse plug-in. It is, however, our intention to support other GUI frameworks in the future.

Once all data for the generation of the QoS policies has been configured in the GUI, component four in Figure 1 will automatically transform the QoS model into a user defined QoS policy. This component will be able to transfer the QoS model to different QoS policies. In this approach, we have used the WS-Policy to demonstrate this work. Figure 1 summarizes our approach.

III. QOS META-MODEL

The QoS meta-model in Figure 1 has two purposes; saving monitoring data of existing SOA environments with existing QoS attributes and the modelling of QoS. QoS monitoring is not the focus of this work. However, it has a great importance for future works. The meta-model for QoS modelling was slightly modified for better flexibility. The following changes were made in the meta-model:

- The CATEGORY attribute in the QOSPARAMETER was modified to include only predefined values specified in the enumeration class QOSCATAGORY.

The following elements were added:

- The QOSVALUE element was expanded to include a new attribute DATATYPE. The new attribute is necessary for the creating of the QoS graphical user interface described in Section V.
- A new enumeration class QOSDATATYPE was added. A predefined values, which are needed for defining the DATATYPE attribute.
- A new enumeration class QOSCATAGORY was added. A list of pre-defined categories the QoS model supports.
- A new QOSPROPERTIES element was added. A list of properties, which could be used to add more information to either the QOSPARAMETER or QOSVALUE.

The following element was not considered in the modified meta-model:

- The QOSLEVEL was not considered in this work since the modelled QoS is always fulfilled.

The meta-model enables the QoS developer to model QoS attributes for Web services for different business domains. A main characteristic of the QoS model is its simplicity and the ability to model QoS attributes. The QoS model has the following relationships:

- Every Web service or Web service method has 0..* QoS parameters.
- Every QOSPARAMETER has exactly one QoS metric. As described in [6], a metric specifies a measurement unit used for describing the QOSVALUE.

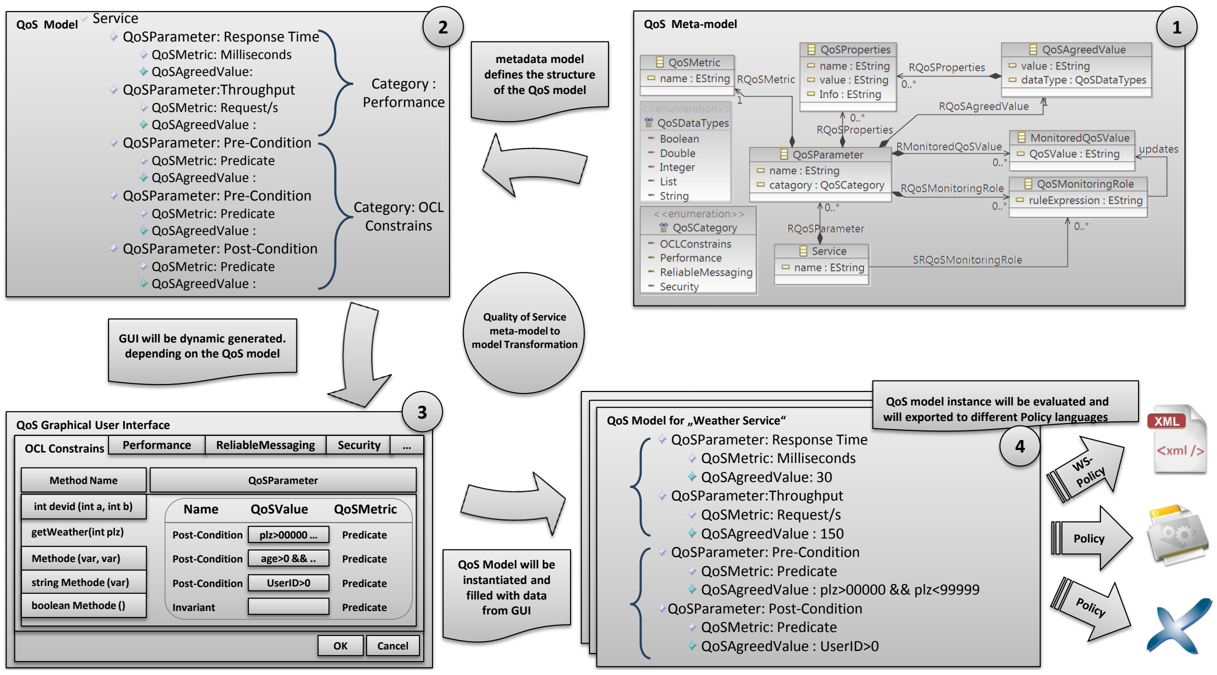

Figure 1. An approach to model, configure and apply QoS attributes to Web services

- Every QoSPARAMETER has exactly one QoS value.
- Every QoSPARAMETER and QoSVALUE have 0..* QoS properties. It is used to add extra information about the QoSPARAMETER or QoSVALUE.
- Every QoSPARAMETER has 0..* monitored QoS values. The monitored QoS values are series of values taken by the monitoring system in order to be either compared to the QoSAGREEDVALUE or to be used to compute instant or average values.
- Every QoSPARAMETER has 0..* QoS monitoring rules, a rule defines how the QoS attributes in SOA should be monitored.

This meta-model is based on EMF (Core), a modelling framework and code generation facility, which is used to building tools and applications based on a structured data model.

IV. QoS MODEL

In this section, we will model three QoS attributes to demonstrate the flexibility of the model. We will present a standardized QoS attribute from the WS-* family and introduce two non-standardized QoS attributes. The first QoS attribute is from WS-ReliableMessaging. Listing 1 models QoS attributes described as a RM policy assertion example in [8], Section 2.4. In the following example, lines (2) - (6) indicate that if the idle time exceeds ten minutes, the sequence will be considered as terminated by the Service Endpoint. lines (7) - (11) express that an unacknowledged message will be transmitted after three seconds. Lines (12) - (16) express that the exponential backoff algorithm will be

```
1  <qossoa:Service  xmi:version="2.0"
2  <RQoSParameter  name="InactivityTimeout"
3      catagory="ReliableMessaging">
4      <RQoSMetric  name="Millisecond"/>
5      <RQoSAgreedValue  value="600000"  dataType="Integer"/>
6  </RQoSParameter>
7  <RQoSParameter  name="BaseRetransmissionInterval"
8      catagory="ReliableMessaging">
9      <RQoSMetric  name="Millisecond"/>
10     <RQoSAgreedValue  value="3000"  dataType="Integer"/>
11 </RQoSParameter>
12 <RQoSParameter  name="ExponentialBackoff"
13     catagory="ReliableMessaging">
14     <RQoSMetric  />
15     <RQoSAgreedValue  />
16 </RQoSParameter>
17 <RQoSParameter  name="AcknowledgementInterval"
18     catagory="ReliableMessaging">
19     <RQoSMetric  name="Milliseconds"/>
20     <RQoSAgreedValue  value="200"  dataType="Integer"/>
21 </RQoSParameter>
22     ...
23 </qossoa:Service>
```

Listing 1. QoS model for reliable messaging

used to retransmitted the message if the message was not acknowledged. Lines (17) - (21) indicate that an acknowledgement could be buffered up to two-tenths of a second by the RM destination.

The following example models a QoS attribute, which is not standardized. Listing 2 describes OCL constraints, a well-known formalism for expressing constraints on classes variables, methods parameters or methods return values. The OCL constraints set preconditions, postconditions and invariants for the Web service class or Web service methods. The pre-condition in line (4), for example, indicate that the given age must be within a range, a minimal age of 18

```
 1 <qossoa:Service xmi:version="2.0"
 2   xmlns:xmi="http://www.omg.org/XMI"
 3   xmlns:qossoa="http://qossoa/1.0">
 4  <RQoSParameter name="preConditions"
 5       category="OCLConstrains">
 6      <RQoSMetric name="predicate"/>
 7      <RQoSAgreedValue value="age>=18 && age<=120"
 8          dataType="String"/>
 9  </RQoSParameter>
10  <RQoSParameter name="preConditions"
11       category="OCLConstrains">
12      <RQoSMetric name="predicate" />
13      <RQoSAgreedValue value="zipCode >01000 &&
14          zipCode<99999 && zipCode.size==5"
15          dataType="String"/>
16  </RQoSParameter>
17   ...
18  <RQoSParameter name="postCondition"
19       category="OCLConstrains">
20      <RQoSMetric name="predicate"/>
21      <RQoSAgreedValue value="userID>0"
22          dataType="String"/>
23  </RQoSParameter>
24  <RQoSParameter name="Invariant"
25       category="OCLConstrains">
26      <RQoSMetric />
27      <RQoSAgreedValue />
28  </RQoSParameter>
29 </qossoa:Service>
```

Listing 2. QoS model for OCL constraints

years and maximum of 120 years. The pre-condition in line (10) indicates that the given zip code must be within rang between 01000 and 99999. It also implies that the zip code must be of five digits since these pre-conditions must comply with the zip code rules in Germany. The post-condition in line (18) defines that the return value USERID shall not be a negative number. The model offers also the possibility to specify an OCL invariant condition as shown in line (24).

Another example of modeling QoS attributes is performance. Response time and throughput are QoS attributes, which are two of the most common used attributes in order to measure performance. As response time refers to the duration, which starts from the moment a request is sent to the time a response is received, throughput refers to the maximum amount of requests that the service provider can process in a given period of time without having effect on the performance of the Web service endpoint [10]. Listing 3 shows an example of how performance can be modelled. Line (4) defines the QoS attribute "ResponseTime", which indicated that the Web service shall guarantee a response time within 10 milliseconds. Line (9) indicated that the Web service will be able to handle up to 120 request/second without having any change on the Web service performance.

V. Graphical User Interface

The graphical user interface is a component, which uses the QoS model to create its representation. Its main purpose is to offer the Web service developer a graphical tool to configure the QoS values of the modelled QoS attributes and associate them with the Web service.

There are two factors, which decide how the GUI should look like; the first factor is the QoS model. The QoS model specifies how many categories shall be represented. Every QoS category is represented by a GUI tab, where all QoS attributes under the represented category will be represented. For example, if the QoS model includes four QoS categories; performance, OCL constraints, reliable messaging and security. The QoS model will be transformed into a GUI, which has four tabs. Each tab will represent a category. If the QoS constraints, for example, has four QoS attributes, the QoS constraints tab on the GUI will represent these four QoS attributes as shown in Figure 1.

The element QoSMetric helps the GUI engine to determin, how the QoSAgreedValue shall be presented. For example, if the QoSMetric indicates that the QoS attribute is a string, the GUI engine will use a text field. If the QoSMetric is a predicate, then the GUI engine will use a check box for the presentation of this attribute.

The second factor is the Web service endpoint. A list of the Web service methods will be extracted either directly from the Web service endpoint interface (SEI) or from the WSDL. Each extracted method has its own list of QoS attributes. If, for example, two Web service methods have two different "ResponseTime" values, a policy for each method will be created. This will result in creating a separate policy for each selected method. The created policy could be also applied Web service wide. These possibilities give the Web service more flexibility and dynamic.

VI. QoS Policy

As a Web service developer configured the QoS attributes on the graphical user interface, all QoS values will be assigned to the QoSAgreedValue element in the QoS model. Once the QoS values has been assigned, a QoS policy will be generated. If, for example, every Web service method has different QoS attributes, a separate policy will be created for every Web service method. A WS-Policy may include the description of more than one QoS attribute depending on the user input in the graphical user interface.

Listing 4 shows the modelled QoS in Listing 1 after being transformed into reliable messaging policy assertion (also described in [8]). The following transformation rules are applied:

```
 1 <qossoa:Service xmi:version="2.0"
 2       xmlns:xmi="http://www.omg.org/XMI"
 3       xmlns:qossoa="http://qossoa/1.0">
 4  <RQoSParameter name="ResponseTime"
 5          category="Performance">
 6     <RQoSMetric name="Millisecond"/>
 7     <RQoSAgreedValue value="10" dataType="Double"/>
 8  </RQoSParameter>
 9  <RQoSParameter name="Troughput" category="Performance">
10     <RQoSMetric name="Requests/s"/>
11     <RQoSAgreedValue value="120" dataType="Double"/>
12  </RQoSParameter>
13 </qossoa:Service>
```

Listing 3. QoS model for performance

```
<wsp:Policy  wsu:Id="MyPolicy">
 <wsrm:RMAssertion>
    <wsrm:InactivityTimeout  Milliseconds="600000"  />
    <wsrm:BaseRetransmissionInterval
         Milliseconds="3000"  />
    <wsrm:ExponentialBackoff  />
    <wsrm:AcknowledgementInterval
         Milliseconds="200"  />
 </wsrm:RMAssertion>
</wsp:Policy>
```

Listing 4. Reliable messaging policy assertion

- The CATEGORY attribute in the QoS model declares the name of the policy. For example, the category “ReliableMessaging” is transformed into a *wsrm:RMAssertion* element declaring a ReliableMessaging policy.
- The element RQOSPARAMETER in the QoS model declares a QoS attribute. The RQOSPARAMETER element indicates the reliable messaging quality attribute “InactivityTimeout” and is therefore transformed into “wsrm:InactivityTimeout” element.
- The element RQOSMETRIC in the QoS model declares a property or how the QoS should be measured. The “Milliseconds” is transformed into “Milliseconds” attribute within the “*wsrm:InactivityTimeout*” element.
- The element RQOSAGREEDVALUE in the QoS model declares a QoS value. The value “600000” will be mapped as a value for the QoS attribute.

Listing 5 shows the modelled QoS in Listing 2 after the transformation into an OCL policy assertion. The same transformation rules apply as already described above.

Once the policies have been created, component four in Figure 1 will assign the created policies to the Web service endpoint interface. In our proof of concept, we use the CXF policy engine to attach the corresponding policy to either the selected Web service methods or the Web service endpoint interface. CXF uses the @POLICY annotation to signal the compiler that there are policies, which should be considered and assigned to the correspond at Web service while creating the Web service WSDL.

VII. RELATED WORK

In our research for related work, a recent approach, which nearly investigates our approach or even a part of it was not found. Most of the recent works on QoS-aware Web services focus on QoS-aware Web services compositions. They investigate methods, algorithm or frameworks in order to better compose Web services according to their QoS attribute. Such works could be found in [1] [2] [5]. In this section, we will describe papers, which propose either QoS meta-models or policy editors.

```
<OCLConstrains  context="RegistrationService">
 <Precondition  predicate="zipCode>01000 &&
     zipCode<99999 && zipCode.size==5"/>
 <Precondition  predicate="age>=18 && age<=120"/>
  ...
 <Postcondition  predicate="UserID>0">
 <Invariant/>
</OCLConstrains>
```

Listing 5. OCL policy assertion

Tondello et al. [12] proposes a QoS-Modelling Ontology, which allows QoS requirements to be specified in order to fully describe a Web service in terms of quality. However, this proposal focuses on using QoS specification for semantic Web services description and Web service search. This approach, however, contains many variables and many characteristics in ontology for semantic Web services, which does not flow in the same direction as this work intends to.

Suleiman1 et al. [11] addresses the problem with Web service management policies during design. The authors presented a solution, which uses a novel mechanism. It generates W-Policy4MASC policies from corresponding UML profiles semi-automatically and feedback information monitored by the MASC middleware into a set of UML diagram annotations.

D’Ambrogio [3] introduced a WSDL extension for describing the QoS of a Web service. It uses a meta-model transformation according to MDA standards. The WSDL meta-model is extended and transformed into a new WSDL model called Q-WSDL, which supports QoS description. As D’Ambrogio favour an approach, which does not support introducing a new additional language on top of WSDL, our approach uses standards for the description of QoS attribute in Web services.

WSO2 WS-Policy editor [14] offers an integrated WS-Policy editor with the WSO2 application server. The editor offer two policy views; a source view and a design view. The source view shows the policy in its XML format and the design view shows the policy as a tree view. The user will be able to add and remove element to and from the policy. However, this policy editor only offers support for WS-Security and WS-ReliableMessaging. A support for new QoS attributes is not mentioned.

NetBeans offers a graphical tool, which allows users to graphically configure security and reliable messaging to a Web service. Extending this tool, however, is complex due to the lack of documentation and its dependability to NetBeans API and Glassfish.

All the these works discuss QoS attributes after the Web services is developed. Our approach offers a solution to develop a QoS-aware Web service.

VIII. FUTURE WORK

In [4], we presented the design of a comprehensive tool chain that facilitates development, deployment and testing of QoS-aware Web services. This paper is a part of the work presented in the tool chain, which elaborates a concept for managing quality of service attributes for Web services. Future works will include different tasks, which will be individually explained in this section.

In Section V, we introduced a GUI, which is dynamically generated depending on the QoS attributes modelled in the QoS model described in Section IV. However, the generation of the GUI is platform-specific. This GUI is only a proof of concept in order to demonstrate the feasibility of this approach. Our goal is to create a GUI using MDA as a base for our approach, which will allow the dynamic GUI generation to different platform.

Section IV indicates that the QoS model will be transformed to a QoS policy. In this paper, we have only considered WS-Policy as a policy language in order to prove that the concept really works. It is the intention of this approach to offer QoS model transformation support to more than one policy language. This will increase the flexibility of our approach.

In [13], we offered a solution architecture, which collects real time data about applied QoS attributes from the SOA environment: The purpose of this architecture is to evaluate the compliance of the entire SOA with the QoS attributes described in the SOA QoS policy. It is our intention to use the meta-model mentioned in Section IV for the evaluation and monitoring of the SOA environment.

This paper presents an approach of how QoS attributes could be easily modelled and transformed into an adequate policy language. However, a policy without a handler, which enforces the policy on the Web service is only half the solution. Future works include a repository component, which is designed to store QoS handlers. This repository will include everything a Web service developer needs to implement a Web service Handler. This includes handler implementation, handler configurations and test cases.

IX. Conclusion

There are tools and IDEs, which help developers to ease the process of creating programs and minimizes their error rates. Nowadays, it is hardly imaginable to start designing and implementing complex systems without them. To create a QoS policy and conjugate it with a Web service requires a good knowledge of its grammar and its mechanism. Tools, which help developers to model QoS attributes, simplify the configuration and automate applying QoS attributes to Web services still has a long way to completion. In this paper, an approach, which relieve a Web service developer with this burden, was presented. It offers an easy way to model QoS attributes. It also supports the modelling of new QoS attributes, simplifies the configuration and automatize applying QoS attributes to Web services. It is a step forward to completing a tool chain for constructing QoS-aware Web services and reducing a lot of development effort and cost.

X. Acknowledgment

This work has been supported by the German Ministry of Education and Research (BMBF) under research contract 017N0709.

References

[1] M. H. Agdam and S. Yousefi. A Flexible and Scalable Framework For QoS-aware Web Services Composition. In *Proc. 5th Int Telecommunications (IST) Symp*, pages 521–526, 2010.

[2] P. Bartalos and M. Bielikova. QoS Aware Semantic Web Service Composition Approach Considering Pre/Postconditions. In *Proc. IEEE Int Web Services (ICWS) Conf*, pages 345–352, 2010.

[3] A. D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS ofWeb Services. In *Web Services, 2006. ICWS '06. International Conference on*, pages 789–796, sept. 2006.

[4] B. Hollunder, A. Al-Moayed, and A. Wahl. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, chapter A Tool Chain for Constructing QoS-aware Web Services, pages 172–188. IGI Global, 2011.

[5] H. Kil and W. Nam. Anytime Algorithm for QoS Web Service Composition. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 71–72, New York, NY, USA, 2011. ACM.

[6] D. Malfatti. A Meta-Model for QoS-Aware Service Compositions. Master's thesis, University of Trento, Italy, 2007.

[7] OASIS. Web Services Policy Framework - Version 1.5, September 2007. http://www.w3.org/TR/ws-policy/. Last Access: 17.04.2011.

[8] OASIS. Web Services Reliable Messaging Policy - Version 1.2, February 2009. http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.pdf. Last Access: 23.04.2011.

[9] OASIS. Web Services Security Policy - Version 1.3, April 2009. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/. Last Access: 17.05.2011.

[10] OASIS. Web Services Quality Factors Version 1.0, July 2010. http://docs.oasis-open.org/wsqm/wsqf/v1.0/WS-Quality-Factors.html. Last Access: 17.05.2011.

[11] B. Suleiman and V. Tosic. Integration of UML Modeling and Policy-Driven Management of Web Service Systems. In *Proc. ICSE Workshop Principles of Engineering Service Oriented Systems PESOS 2009*, pages 75–82, 2009.

[12] G Tondello and F. Siqueira. The QoS-MO Ontology For Semantic QoS Modeling. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 2336–2340, New York, NY, USA, 2008. ACM.

[13] A. Wahl, A. Al-Moayed, and B. Hollunder. An Architecture to Measure QoS Compliance in SOA Infrastructures. In *Proceedings of the Second International Conferences on Advanced Service*, pages 27–33, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[14] WSO2. WSO2 WSAS: The WS-Policy Editor 3.2.0 - User Guide, April 2010. http://www.wso2.org/project/wsas/java/3.2.0/docs/policyeditor/docs/userguide.html. Last Access: 17.05.2011.