Enhanced Deployment Process for QoS-aware Services

Alexander Wahl and Bernhard Hollunder
Department of Computer Science
Furtwangen University of Applied Science
Robert-Gerwing-Platz 1, D-78120 Furtwangen, Germany
alexander.wahl@hs-furtwangen.de, bernhard.hollunder@hs-furtwangen.de

***Abstract*—Service-oriented architectures (SOA) are a widely used design paradigm for the creation and integration of distributed enterprise applications. The predominant technology used for implementation are Web services. In the business domain, Web Services must be equipped with quality of service (QoS) attributes, like, e.g., security, performance, etc. WS-Policy standard provides a generic framework to formally describe QoS. Verification and enforcement of such formally described QoS require corresponding QoS modules, e.g., handlers, to be installed at the runtime environment. In this work, we provide an approach that enhances the current deployment process for Web services. The aim is to provide a sophisticated process that not only deploys Web Services and formal QoS descriptions, but to guarantee that the desired QoS are verified and enforced in the runtime environment. We therefore introduce additional steps for the analysis of WS-Policy descriptions, the identification of corresponding handlers, and their installation at the runtime environment. As a result, a comprehensive, automated deployment process is created, and the fulfillment of an overall WS-Policy description is ensured.**

***Keywords*-Service-oriented architecture; QoS-aware service; Deployment.**

I. INTRODUCTION

The usage of distributed systems is nowadays widespread. With the increased availability of networks, especially the internet, distributed systems find application in business domains as well as in private environments. Typically, some desired distant functionality is used following the client-server model. The Service-oriented Architecture (SOA) paradigma is a well-known paradigma to form such a system: business functionality is implemented as a service and offered to service consumers. The communication takes place over a, possibly public, network.

Beside the offering of pure functionality, the fulfillment of non-functional aspects that relate to Quality of Service (QoS), is an important factor. For example, for e-payment security and transactional behavior aspects are obviously crucial. Beside these exemplary QoS several others exist, that apply to the domain of distributed systems, respectively SOA, such as response time, availability, cost, usage control, roles and rights, etc.

In a technical environment, like a SOA infrastructure, it is worthwhile that desired QoS are analyzed and enforced in an automated manner by the infrastructure itself. In order to achieve such an automatism these QoS first have to be formally described. Next, the formal description is related to the desired services, following the separation of concerns (SoC) principle [1]. In other words, the functionality implementation is separated from the QoS implementation. The functionality is implemented using some programming language, like Java, C++ or C#. For the QoS implementation usually declarative languages, so-called policy languages, are used. The resulting service is then called a QoS-aware service.

A well-known and widely used language to formally describe QoS is WS-Policy [2]. In the context of WS-Policy a policy is a collection of policy alternatives. Each policy alternative is a set of policy assertions. The policy assertions are used to express QoS. It should be noted, that WS-Policy does not provide concrete assertions, but is a more general framework. Concrete assertions are introduced in related specification, like, e.g., WS-SecurityPolicy [3]. With WS-Policy, other domain-specific assertions can be defined.

Creating appropriate formal descriptions of a QoS is in the repsonsibility of the service developer. This is no easy task, and a deep knowledge on several different models, languages and technologies is a must. This is especially true with QoS for whom no standardized specifications, frameworks and approaches are available.

The formal description of a QoS is just half the way. The validation and enforcement of such a QoS description is in the responsibility of the service runtime environment. However, it has to be ensured, that the specified QoS of a service is understood and enforced by that runtime environment. One can think of different approaches to achieve that. In this work, we will enforce QoS by using specialized service agents, so-called handlers. These handlers are installed, configured and invoked at the runtime environment. It has to be mentioned, that this approach does not apply to all QoS, but to a significant set of of QoS. Again, a comprehensive knowledge on the infrastructure used, the service provider and different used technologies, e.g. frameworks, is necessary.

Figure 1 gives an overview on a typical process currently used to create a QoS-aware service. The process used so far attaches a QoS description (using, e.g., WS-Policy) to a service implementation. The resulting artifact is then

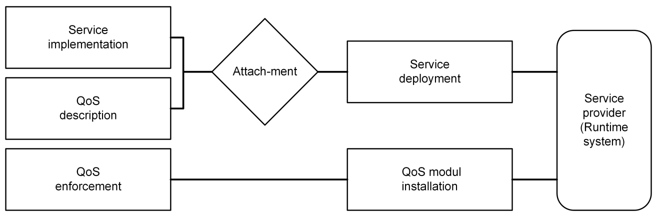

Figure 1. QoS-aware service development, deployment and installation process.

transferred to the service provider. Todays process does not check the availability of corresponding QoS modules at the runtime environment, i.e., the service provider. It does not include an analysis of the QoS description to identify, install and configure the QoS modules, i.e., handlers, that are necessary to enforce the QoS. The solution of this deficit is enhance the current deployment process by an automated module installation, which is based on the analysis of a QoS description. By that means it can be guaranteed, that a QoS description are verified and enforced during runtime.

In this work, we integrate additional steps to the deployment process for policy analysis, and QoS module identification, installation and configuration. The aim is to provide a comprehensive, streamlined and significantly eased deployment process for QoS-aware services. As benefit

1) the development process of QoS-aware WS is improved by additional steps that identify and install necessary QoS modules, and
2) the availability and usage of uniform (and thereby more interoperable) QoS descriptions and corresponding handlers is increased.

This paper is organized as following: In Section II the problem is described in more detail, and a solution strategy is introduced. Section III provides a general overview on our approach, which is explained more precisely in Section IV. Section V presents a prototypic implementation of our solution. A discussion on related work is given in Section VI, followed by a conclusion in Section VII.

II. Problem Description and Solution Strategy

The development of QoS-aware services is a demanding task. Beside the implementation of the service, the desired QoS need to be formally described. Both, service and QoS description need to be bound, and afterwards be deployed to the runtime system. Except for a few QoS, that are supported by current frameworks, enforcement modules for the desired QoS also need to be implemented, and afterwards installed at the runtime system. Overall, a developer needs to have extensive knowledge of available languages and applicable technologies in order to realize a QoS-aware service.

A more sophisticated approach is desirable. Such an approach first enables to describe QoS, including non-standardized ones, at a higher level of abstraction, and to generate appropriate formal representation in an automated manner. Next, it applies the generated QoS representation to the services under development, and identifies appropriate QoS modules. Finally, it gathers and installs – if necessary – the required QoS modules at the infrastructure.

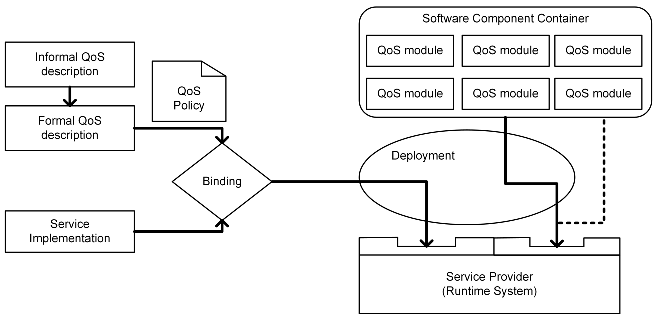

Figure 2. Enhanced service deployment overview.

This approach can be divided into two parts. The former part is already covered by Al-Moayed, Hollunder and Wahl [5], who provide a solution to specify QoS at an higher level of abstraction and to generate corresponding WS-Policies descriptions. In this work, we enhance this approach by the latter steps described before: The generated QoS description is analyzed, and corresponding QoS modules are identified. Afterwards these QoS modules are retrieved from a dedicated container component (i.e. the Software Component Container), configured, and finally installed at the service provider. Figure 2 visualizes this approach.

III. Approach

In this section, we describe the deployment process steps in more detail: the analysis of a formal QoS description with regard to QoS module identification, the QoS module identification itself, and the QoS module installation (see Figure 1).

The introduced approach is part of a more general approach described in Hollunder, Al-Moayed and Wahl [6]: A Tool Chain for Constructing QoS-aware Web Services.

The formal description of a desired QoS, i.e., QoS description, is realized with WS-Policy. In Al-Moayed, Hollunder and Wahl [5] a QoS is specified using a model-based approach. On that QoS model model-to-model and model-to-code transformations are performed that finally create a formal QoS description based on WS-Policy.

In a first step, the generated WS-Policy description is processed. The aim is to identify all occuring assertions. Assertions are the basic building blocks of a WS-Policy. These assertions reflect QoS. For each assertion a QoS module must be available, and the QoS module verifies and enforces the QoS. The identification of such a specific module requires knowledge on which assertions a QoS module is capable of. In our approach, we introduce the Software Component Management Unit, whose responsibility is to i) store and manage all available QoS modules, and ii) to store the information how assertions are implemented by each QoS module.

For each identified assertion of the WS-Policy description, a corresponding QoS module is searched within the Software Component Management Module. Once the assertions are related to a corresponding QoS module, the required QoS modules to enforce the given WS-Policy are known. With that information, the runtime environment can be equipped with these QoS modules. This will ensure, that the complete WS-Policy description is verified and enforced at the runtime environment.

IV. Enhanced Deployment Process

In this section we will describe the individual steps and components of the approach presented in the previous section.

A. *Analysis of a formal QoS description*

Consider a formal QoS description, e.g., based on WS-Policy. Listing 1 shows an example, which was initially described in [5]. Line 1 defines a policy description with Id `CalculatorConstraintPolicy`. Line 2 specifies, that the following are policy alternatives, which is equivalent to OR. Line 3 introduces a set of policy assertion, which equals AND. Lines 4 and 5 are assertions that specify a QoS constraint – a range of numbers – with two QoS parameter `wscal:minInt` and `wscal:maxInt`. The remaining lines are closing tags for lines 1-3.

Upon closer examination, `wscal:minInt` and `wscal:maxInt` in lines 3 and 4 concrete assertions. Both have to be interpreted by an appropriate handler (e.g., http handler or SOAP handler). Since these assertions do not belong to any known WS-Policy related specification, the handler have to be developed from scratch. Same is true for any other custom assertion.

```
1 <wsp:Policy wsu:Id="CalculatorConstraintPolicy">
2  <wsp:ExactlyOne>
3   <wsp:All>
4    <wscal:minInt Number="-32768"> </wscal:minInt>
5    <wscal:maxInt Number="32767"> </wscal:maxInt>
6   </wsp:All>
7  </wsp:ExactlyOne>
8 </wsp:Policy>
```

Listing 1. Calculator service WS-Policy description.

As described before, the initial step is to identify all assertions within the WS Policy. In our example, these are the two assertions `wscal:minInt` and `wscal:maxInt` in line 4 and 5. Therefore, in our example the result of the analysis step is a list of two elements.

B. *Software Component Container*

The Software Component Container is a central component of the approach. Its main purpose is to contain all available QoS modules. For each such QoS module there is a relation to the WS-Policy assertions it implements. This relation is also stored in the container.

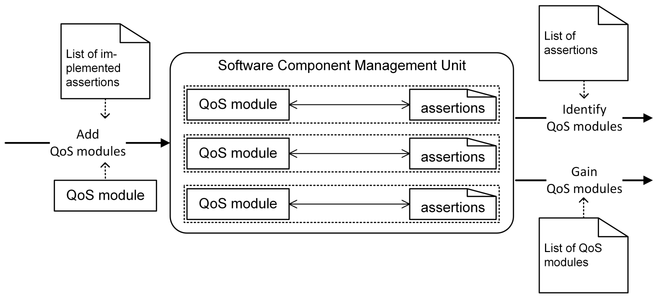

Figure 3. Software Component Container.

There are three interfaces at the Software Component Container (see Figure 3). The first interface provides a means to add QoS modules to the container. This interface requires information on the implemented assertions and the QoS module. Via a second interface a list of corresponding QoS modules for a given list of assertions can be retrieved. Finally, QoS modules can be gained from the Software Component by a third interface.

C. *QoS module identification*

QoS modules are well-defined software components that enforce a desired QoS. In this work, we focus on handlers, which are one category of QoS modules. The term QoS module therefore is equivalent handler. However, this approach is not limited to the handler approach.

This step identifies the QoS modules needed to verify and enforce the formalized QoS described in the WS-Policy file. Input for this step is the list of QoS assertions described before. For each entry of that list the Software Component Container is inquired. Remind that the Software Component Container is aware of all assertions implemented by any of its registered QoS modules. If an assertion is found at the Software Component Container, a corresponding QoS module is available, and the module is stored in a list of necessary QoS modules. Otherwise, no appropriate QoS module is available, and the assertion cannot be verified and enforced. In that case the assertion is added to a list of unresolved assertions. With unresolved assertions there are different options, ranging from canceling the enhanced deployment process up to inform the developer at the end of the deployment process.

In summary, the QoS module identification step is able to identify QoS modules, that are needed to enforce an QoS, based on the assertions used within the WS-Policy. It further enables to identify assertions, where do not exist corresponding QoS modules. With that, the developer of a QoS-aware service is able to recognize in a proactive manner, which assertions can or cannot be handled by the runtime environment, reflected by i) a list of QoS modules to be installed, and ii) a list of unresolved assertions.

D. QoS module installation

Once the QoS modules are identified, the WS runtime environment is checked, if these modules are already installed. If there are modules missing, they are collected from the Software Component Management Unit, configured and installed. Using WS technology, the runtime environment is typically an application server. Such an application server usually provides a management API, that can be accessed to install additional modules, like the QoS modules. Once all QoS modules are installed, it is assured that the overall WS-Policy description given can be fulfilled.

E. Automation

Up to today the deployment process of a QoS-aware service mainly includes steps for buidling, packaging and installing in some container, as described elsewhere, e.g., for WSIT [7]. As visualized in Figure 1, identification, configuration and installation of necessary QoS modules is not included in the deployment process. In the preceding paragraphs we identified the steps that are to be performed in order to be sure that a given WS-Policy can be completely handled. We state that each of these steps can be automated. There is no need for user interaction at each of these steps – even with unresolved assertions. We therefore argue to enhance the current deployment process with the described steps.

V. Proof of Concept

For a proof of concept, we focused on SOA using Java-based infrastructure and technologies. In detail:

- NetBeans IDE [8]
- Java API for XML Web Services (JAX-WS) [9]
- Glassfish application server [10]
- Eclipse IDE [11]
- Apache Ant [12]
- Apache Neethi [13]
- Apache Subversion [14]
- MySQL Community Edition [15]

NetBeans IDE is used to create Web Services based on JAX-WS technology. We further use the Glassfish application server. It can be registered to the NetBeans IDE as deployment destination. NetBeans IDE uses Apache Ant to implement the deployment process, as described in the WSIT Tutorial [7]. The corresponding Apache Ant build files, for building the Web Archive (WAR) and for deployment, are generated by NetBeans.

Eclipse IDE is used to create a formal QoS description based on WS-Policy. We use Eclipse IDE due to the fact, that the prototypic tool of Al-Moayeds approach is a Eclipse plugin.

The description of the WS interface is realized using the Web Service Description Language (WSDL). WSDL may refer to a WS-Policy description. When a WS is invoked, the WS runtime environment recognizes the existence of a policy and delegates the request to the installed handlers. A handler then processes the request according to the policy assertions it is responsible for.

```
<target description="Build Web Archive (WAR)."
        name="dist">
  <jar jarfile="dist/CalculatorService.war">
      <fileset dir="web" />
  </jar>
</target>
```

Listing 2. Building a Web Archive (WAR) using Apache Ant.

Apache Ant is used to automate the creation of the WAR and the deployment. Listing 2 displays the corresponding target `dist`, which uses the task `jar` to create a WAR named `CalculatorService.war` in folder `dist`.

```
<target description="Deploy Web Archive (WAR)."
        name="deploy">
  <get src="http://localhost:4848/__asadmin/deploy?
             path=dist/CalculatorService.war"/>
</target>
```

Listing 3. Deployment of a Web Archive (WAR) using Apache Ant.

Listing 3 shows the Ant target `deploy` to deploy the WAR file to a Glassfish application servers. It uses the Administration Console running on `localhost:4848`, and invokes the `asadmin` command with parameter `deploy`. Afterwards the `CalculatorService.war` is available.

```
<target name="identify-handler">
  <java jar="qos-module-identification.jar">
    <arg value="in=policy.xml"/>
    <arg value="out=handler.xml"/>
    <arg value="out=unresolved-assertions.xml"/>
  </java>
</target>

<target name="install-handler">
  <java jar="qos-module-installation.jar"/>
    <arg value="in=handler.xml"/>
  </java>
</target>
```

Listing 4. Steps introduced in Enhanced Deployment Process.

Between these two steps, `dist` and `deploy`, we introduce further steps that enhance the deployment process by the step for WS-Policy analysis, handler identification and installation, as described in Section IV. In Listing 4 the two introduced targets are shown. The first target, `identify-handler`, invokes `qos-module-identification.jar`, which implements the WS-Policy analysis and handler identification steps; `qos-module-installation.jar` implements the handler installation, which is invoked by `install-handler`.

Within `qos-module-identification.jar` Apache Neethi, an open-source implementation of WS-Policy, is used to parse a WS-Policy description and to identify its assertion. Afterwards, the Software Component Container (described later) is invoked for each assertion to identify the implementing QoS module. To compare the assertions the policy intersection algorithm of WS-Policy is used. If a match of assertions is found, the corresponding QoS module is stored. Otherwise the assertion is added to the list of unresolved assertions.

The Software Component Container responsibility is to administer the individual QoS modules, and to track the assertions implemented by a QoS module. We use a software versioning and revision control system, Apache Subversion, to version each QoS module. For each QoS module metadata is stored. Beside others, this metadata mainly consists information on the assertions implemented with a QoS module, author, version, etc. These data are saved using a MySQL database. Both, Apache Subversion and MySQL come with APIs for Java, which enables to implement a dedicated component for QoS module identification.

A further Java-based component, implemented within `qos-module-installation.jar`, is used to gather and install the identified handler within the Glassfish application server. This component checks out the handler from the repository using the Subversion API. Afterwards, the QOS modules are installed. We use the Applicationserver Management eXtensions (AMX) and Java Management Extensions (JMX) to perform this step.

The proof of concept showed that an automated identification of handler based on assertions in a WS-Policy description, and an installation of these handler is feasible. The approach ensures that an overall WS-Policy description given can be fulfilled. But it showed that the execution of the Enhanced Deployment Process requires administrative access to the application server.

VI. Related Work

There are books and papers that describe QoS-aware WS, the deployment process, and the use of QoS components, i.e., handlers. However, the identification and installation of QoS modules is so far a manual step.

In Erl's book [4] service deployment is a phase of the SOA delivery lifecycle. In this stage distributed components, service components, service interfaces, and any associated middleware products are installed and configured on the production server. In another book [16], he describes how to add WS-Policy descriptions to a WSDL.

WS-Policy is widely used to formalize QoS. Hollunder [17] discusses the introduction of an operator in WS-Policy for conditional assertions. He further describes the implementation of a corresponding policy handler based on the Apache Axis framework. Mezni, Chainbi and Ghedira [18] extend WS-Policy to specify services related data in order to enable for policy-based self-management and to describe autonomic Web services. Mathes, Heinzl and Freisleben [19] extend WS-Policy to introduce time-dependant policy descriptions, which allows to specify time constraints on the validity of a policy description.

In this work assertions are matched using the policy intersection algorithm. Hollunder [20] presents a new approach to determine the compatibility of policies that operates not only syntactically but also takes into account the semantics of assertions and policies. Brahim, Chaari, Jemaa and Jmaiel [21] present a sematic approach to match WS-SecurityPolicy assertions.

Al-Moayed, Hollunder and Wahl [5] introduce a model-based approach to create a policy description based on WS-Policy. They use a meta-model introduced by Malfatti [22].

A description of the deployment process using Apache Ant is provided in the WSIT Tutorial [7]. Another framework, Apache CXF [23], also uses Apache Ant to implement the deployment process.

VII. Conclusion

Designing and implementing QoS for a SOA is a demanding task. Up to now just a few QoS are supported by IDEs, tools and frameworks. But there are several QoS that are still implemented in a proprietary manner. Also, for the verification and enforcement corresponding QoS modules need to be developed. As a result numerous different implementations for each QoS exists, which are usually not interoperable.

Automated deployment processes for Web Service that use WS-Policy to describe QoS are established and widely-used. However, identification, configuration and installation of QoS modules, e.g., handlers, at the runtime environment is still performed manually.

In this work, we proposed a way to identify the QoS modules needed to enforce a WS-Policy description. We further introduced a component i) to manage available QoS modules including different versions, ii) to store the implemented constraints of each QoS module, and to iii) identify the QoS modules necessary to verify and enforce an overall WS-Policy. We further introduced a component that is able to gather the desired QoS modules and to install them at the runtime environment.

Finally, we showed that the deployment process can be enhanced to identify and install missing QoS modules automatically, to identify assertions that cannot be implemented at all due to unavailable QoS modules.

Further, by using a central, dedicated Software Component Management we expect to improve the availability of QoS modules and corresponding QoS descriptions over time by publicly offering such a unit.

The benefit of this work is the consolidation of two separate processes that are both necessary to successfully implement QoS-aware Web Services. One is the deployment process of Web Services and WS-Policy descriptions. The other is the installation of handler at the infrastructure, which also has been automated. The introduction of a dedicated unit that manages QoS modules, relates them to the implemented assertions is a valuable progress.

But there are still challenges. We plan to investigate further QoS with regard to implementation strategies, and on new technologies to support this implementation. Further, we want to improve the availability of QoS. Finally the comprehensive tool chain for QoS-aware Web Services is further improved.

ACKNOWLEDGMENT

We would like to thank the anonymous 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] S. Robertson and J. Robertson, *Mastering the requirements process*, 3rd ed., P. Education, Ed. Addison-Wesley, 2012.

[2] A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and U. Yalcinalp, “Web services policy 1.5 - framework,” World Wide Web Consortium, Tech. Rep., 2007, last access: 15. Jan. 2013. [Online]. Available: http://www.w3.org/TR/ws-policy/

[3] K. Lawrence and C. Kaler, “WS-SecurityPolicy 1.2,” OASIS, Tech. Rep., Feb. 2009, last access: 15. Jan. 2013. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy.html

[4] T. Erl, *Service-oriented architecture*, 9th ed. Upper Saddle River, NJ [u.a.]: Prentice-Hall, 2005.

[5] A. Al-Moayed, B. Hollunder, A. Wahl, and V.Sud, “Quality attributes for web services: A model-based approach for policy creation,” *International Journal on Advances in Software*, vol. 5, no. 3&4, pp. 166–178, Dec. 2012. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=soft_v5_n34_2012_2

[6] B. Hollunder, A. Al-Moayed, and A. Wahl, “A tool chain for constructing QoS-aware web services,” in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. IGI Global, 2012, pp. 189–211. [Online]. Available: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-60960-794-4.ch009

[7] Oracle, “The WSIT tutorial,” last access: 15. Jan. 2013. [Online]. Available: http://docs.oracle.com/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/index.html

[8] ——, “Netbeans IDE.” [Online]. Available: http://netbeans.org/

[9] java.net, “JAX-WS,” last access: 15. Jan. 2013. [Online]. Available: http://jax-ws.java.net/

[10] ——, “Glassfish,” last access: 15. Jan. 2013. [Online]. Available: http://glassfish.java.net/de/

[11] The Eclipse Foundation, “Eclipse IDE,” last access: 15. Jan. 2013. [Online]. Available: http://www.eclipse.org/downloads/moreinfo/jee.php

[12] The Apache Software Foundation, “The apache ant project,” last access: 15. Jan. 2013. [Online]. Available: http://ant.apache.org/

[13] ——, “The apache neethi project,” last access: 15. Jan. 2013. [Online]. Available: http://ws.apache.org/neethi/

[14] ——, “The apache subversion project,” last access: 15. Jan. 2013. [Online]. Available: http://subversion.apache.org/

[15] Oracle, “Mysql community edition.” [Online]. Available: http://www.mysql.com/

[16] T. Erl, Ed., *Web service contract design and versioning for SOA*, ser. The @Prentice Hall service-oriented computing series from Thomas Erl. Upper Saddle River, NJ [u.a.]: Prentice Hall, 2009.

[17] B. Hollunder, “Ws-policy: On conditional and custom assertions,” in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, july 2009, pp. 936 –943.

[18] H. Mezni, W. Chainbi, and K. Ghedira, “Aws-policy: An extension for autonomic web service description,” *Procedia Computer Science*, vol. 10, no. 0, pp. 915 – 920, 2012, ¡ce:title¿ANT 2012 and MobiWIS 2012¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050912004796

[19] M. Mathes, S. Heinzl, and B. Freisleben, “Ws-temporalpolicy: A ws-policy extension for describing service properties with time constraints,” in *Computer Software and Applications, 2008. COMPSAC ’08. 32nd Annual IEEE International*, 28 2008-aug. 1 2008, pp. 1180 –1186.

[20] B. Hollunder, “Domain-specific processing of policies or: Ws-policy intersection revisited,” in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, July, pp. 246–253.

[21] M. B. Brahim, T. Chaari, M. B. Jemaa, and M. Jmaiel, “Semantic matching of ws-securitypolicy assertions,” in *Service-Oriented Computing - ICSOC 2011 Workshops*. Springer Berlin Heidelberg, 2012, pp. 114–130. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31875-7_13

[22] D. Malfatti, “A Meta-Model for QoS-Aware Service Compositions,” Master’s thesis, University of Trento, Italy, 2007.

[23] The Apache Software Foundation, “Apache CXF,” last access: 15. Jan. 2013. [Online]. Available: http://cxf.apache.org/