Quality Attributes for Web Services: A Model-based Approach for Policy Creation

Alexander Wahl, Bernhard Hollunder, and Varun Sud
Department of Computer Science
Furtwangen University of Applied Science
Furtwangen, Germany
alexander.wahl@hs-furtwangen.de, bernhard.hollunder@hs-furtwangen.de, varun.sud@hs-furtwangen.de

Ahmed Al-Moayed
BI/ HANA Deparment
Adweko GmbH
Walldorf, Germany
ahmed.al-moayed@adweko.com

***Abstract*—Service-oriented architectures (SOA) define a conceptual framework for the creation and integration of enterprise applications. Within an SOA, the core functionality is realized by distributed services, which are typically composed to support the required business processes. Today, Web services are the predominant technology to implement and deploy services in heterogeneous environments. In many business domains, Web services must exhibit quality of service (QoS) attributes such as security, performance, scalability, and accounting. Currently, there is only limited support for the assignment of QoS attributes to Web services, though. In this paper, we present a model-based approach for deriving policies from a QoS model. Our solution covers the modeling of QoS attributes based on a meta-model for quality attributes, the generation of a graphical user interface to configure the modeled QoS attributes, and the transformation into policy descriptions. Finally, these policies will be assigned to the target Web services. To highly automate our approach, we apply techniques from model-driven development such as model-to-model and model-to-code transformations. As a consequence, our solution reduces the cost and effort when creating QoS-aware Web services.**

***Keywords*-Service-oriented architecture; Web services; QoS meta-model; model-to-model transformation; model-to-code transformation; WS-Policy.**

I. Introduction

Service-oriented architectures (SOA) refer to a system architecture that provides applications and software components as reusable and interoperable services with well-defined business functionalities. In most deployment settings, the services must also address non-functional requirements such as security, performance, and accounting in order to guarantee predefined quality of service (QoS) attributes the overall business applications must fulfill. As an enterprise typically refers to a variety of internal and external service providers, it is crucial to explicitly assign QoS attributes to the underlying Web services.

In the literature, several policy languages have been proposed to formally specify QoS attributes for particular technical or business domains. With the WS-Policy specification [2], there exists a well-known and widely used framework for defining QoS attributes for Web services. Basic building blocks in WS-Policy are so-called assertions, where a single assertion may represent a domain-specific capability, constraint or requirement. In order to create valid WS-Policy descriptions for non-trivial scenarios, technical knowledge regarding the design of WS-Policy assertions and the underlying policy grammar is required (see, e.g., [3], [4]).

However, a developer may not necessarily acquire this knowledge, but should be enabled to easily assign QoS attributes to the Web service under development.

In this paper, we present a model-based approach to specify, configure, and assign QoS attributes to Web services. Given a QoS meta-model, the "QoS profile developer" creates instances of the meta-model, which formalize QoS attributes for dedicated domains such as security or performance. Such models, also called QoS profiles, are reusable assets and can be applied to different Web services by the "Web service developer". By means of model transformations, a graphical user interface (GUI) is generated, which is used by the developer to adjust the predefined QoS attributes for the specific deployment context of the Web service. Eventually, the refined QoS profiles are automatically translated into corresponding descriptions of well-known policy languages such as WS-Policy.

It should be noted that Integrated Development Environments (IDE) such as Eclipse, NetBeans, and Visual Studio provide specific project types for the construction of Web services. These environments automate many activities such as code compiling, WSDL interface generation, creation of proxy objects and code deployment. However, currently

This is a revisited and substantially augmented version of "An Approach to Model, Configure and Apply QoS Attributes to Web Services", which appeared in the Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011) [1].

there is only limited support for developing QoS-aware Web services, i.e., Web services with well-defined QoS attributes. In particular, these environments are either hard to extend or are restricted to certain policy domains such as WS-SecurityPolicy [5] and WS-ReliableMessaging [6]. To our best knowledge, there is no framework covering the following features:

- A simple, but powerful QoS meta-model for formalizing arbitrary QoS attributes.
- An easy way to create QoS profiles (i.e., collections of QoS attributes) for Web services.
- The automatic creation of a graphical user interface, which allows the developer to configure the modeled QoS attributes for each designated Web service.
- Automatic transformation of the configured QoS profile into equivalent policy descriptions.
- Assignment of the created policy description to the Web service under development.

In this work, we elaborate the conceptual and technical foundations of such a framework. We also describe our proof of concept implementation, which demonstrates the feasibility of the approach. Our solution is a further step to reduce the developing effort and the costs of creating QoS-aware Web services.

This paper is structured as follows. The next section describes the problem addressed in this paper in more detail and sketches our solution strategy. After introducing the solution architecture in Section III, the successive sections focus on particular elements: Section IV introduces the QoS meta-model followed by the QoS profile (Section V) and the graphical user interface for configuring QoS attributes (Section VI and VII). In Section VIII, the generation of policy descriptions is elaborated. A description of the proof of concept implementation is given in Section IX. Then, a discussion on related work (Section X) and future work (Section XI) is given, followed by a conclusion.

II. Problem Description and Solution Strategy

Developing QoS-aware Web services is a strenuous task for Web service developers. Typically, QoS attributes are hardcoded into the Web service business logic increasing code complexity. Implementing QoS attributes in the source code also decrease the degree of reusability of Web service for different deployment settings and flexibility to react on changing QoS requirements. Web service developers explicitly require knowledge of policy languages, e.g., WS-Policy, to create and apply policies to Web services. They also require knowledge of associated policy grammars such as policy-domain specific tags, elements, nesting rules, rules of operations and operators and policy expression creation rules. Such knowledge is not always available to the Web service developers. Currently available solutions and support to create and apply QoS attributes to the Web services are either limited to certain domains or specific to development environments.

In this paper, we present an approach to answer the following questions and challenges that emerge with such QoS-aware Web services:

- Is it possible to model, configure and apply not only standardized QoS attributes but also project specific QoS attributes to Web services in an easy, extendable and flexible manner?
- Is there a solution to enable quick development of QoS-aware Web services irrespective of the Web service implementation language, business logic and policy domains?
- Moreover, is there a solution to handle and enable frequent changes in business requirements with respect to non-functional requirements?
- Can the time, complexity and effort in designing, modeling, creating and applying QoS attributes to the Web services be reduced?

This paper offers a solution, a tool chain, which automates and simplifies modeling, configuring and applying of QoS attributes to Web services. Figure 1 describes the working of our proposed solution with the actors, components and processes involved.

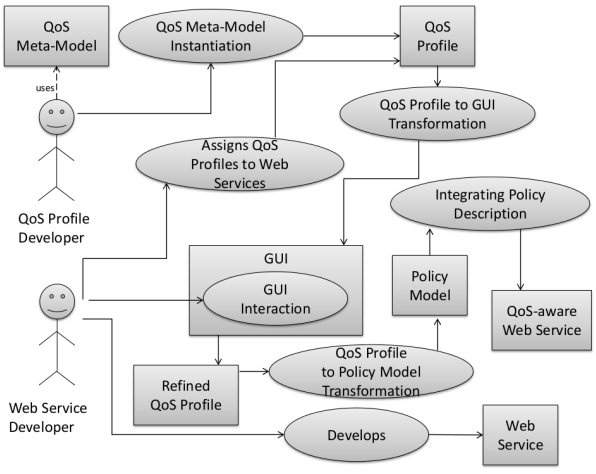

Figure 1. Use case diagram describing our solution strategy.

As shown in the figure, there are two distinctive roles: the QoS profile developer and Web service developer performing their concerned tasks. QoS profile developer uses the QoS meta-model to create a so-called QoS profile, which forms a reference between the Web services and the QoS attributes. Examples of such QoS profiles are security profile and performance profile for a Web service.

By undergoing certain set of transformations, a GUI is generated from a QoS profile. Web service developer now interacts with the generated GUI to create a refined QoS profile based on certain business requirements. The Web

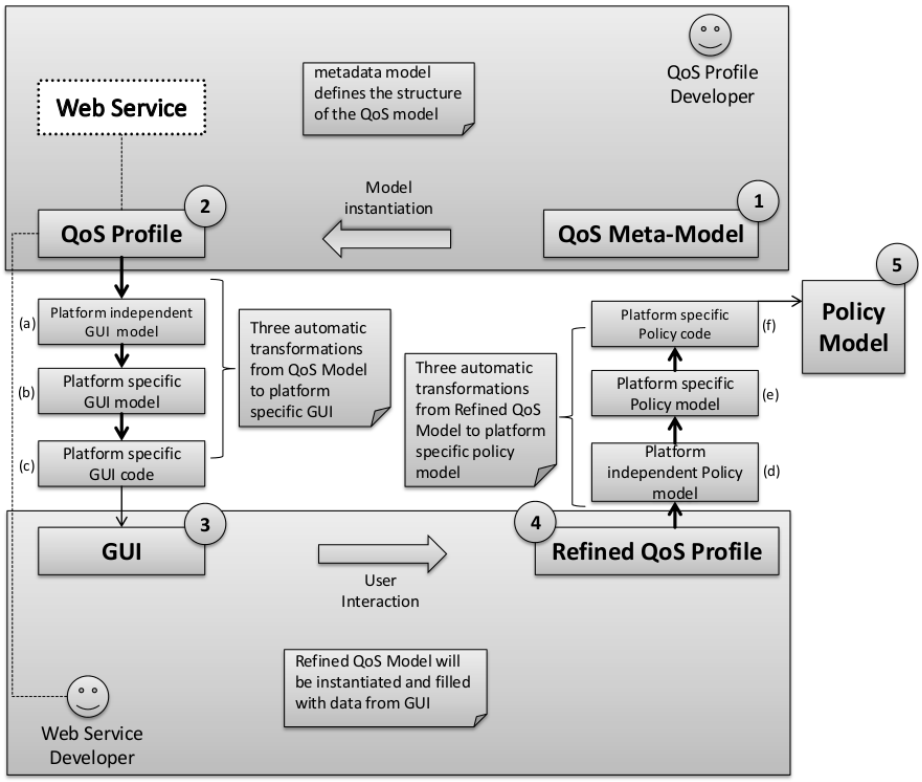

Figure 2. An approach to model, configure and apply QoS attributes to Web services.

service developer later assigns the refined QoS profile to the concerned Web service.

Refined QoS profile follows a set of transformations to create a policy model. Policy model is the starting point to produce policy descriptions to be integrated into the Web services to finally generate QoS-aware Web services. By introducing this solution strategy, we can answer the questions described in the problem description in the following manner:

- QoS meta-model component allows modeling, configuring and applying QoS attributes of not only standardized but also project specific QoS attributes in an easy extendable and flexible manner.
- QoS meta-model, QoS profile, refined QoS profile and policy model are components of the solution strategy. QoS profile, refined QoS profile and policy model undergo transformations to generate business specific solutions.
- The mentioned transformations also allow reprocessing of new technical and business specifications at different components of the solution strategy.
- Separating the tasks of QoS profile developer and Web service developer reduces the effort, time and complexity of designing, modeling, creating and applying QoS attributes to Web services.

III. APPROACH

Figure 2 shows the solution architecture in more details describing our approach. The first component is the QoS meta-model (see (1) in Figure 2). It describes exactly how the QoS profiles are created. It is simple, extensible, easy to understand and expressive enough to model arbitrary QoS attributes.

The second component is the QoS profile (see (2) in Figure 2), which is an instance of QoS meta-model. It offers the QoS profile developer a way to model QoS attributes within QoS categories that are already defined in the QoS meta-model. As we will see in Section VII, with this meta-model, we will be able to model different QoS attributes including QoS attributes such as reliable messaging and performance.

Once a QoS profile has been instantiated, the third component of the solution, a GUI is generated (see (3) in Figure 2) based on certain transformation rules. The transformations process essential information from the QoS profile and depending on the elements in the QoS profile generate the GUI. The main purpose of the GUI is to provide Web service developer an interface to refine and configure available QoS profiles. For example, a Web service developer can choose and specify specific encryption algorithm for a QoS security profile.

After a successful user interaction with the GUI, the forth component of the solution, i.e., a refined QoS profile (see (4) in Figure 2) is obtained. The refined QoS profile is the actualized version of the QoS profile, which is again transformed into the policy representation, i.e., the fifth component of the solution (see (5) in Figure 2). The policy model is the final step towards applying QoS attributes as policies to Web services.

In general, there are six transformations taking place from QoS profile to policy model generation:

(a) An automated transformation from QoS profile to platform independent GUI model (GUI PIM). This enables easy extension of the solution to different GUI technologies such as Swing, SWT and WPF.
(b) An automated transformation from GUI PIM to platform specific GUI model (GUI PSM). GUI PSM is specific to modelled QoS attributes.
(c) An automated transformation from GUI PSM to platform specific GUI code.
(d) An automated transformation from refined QoS profile to platform independent policy model (Policy PIM). This enables easy extension of the solution to different policy formalisms such as WS-Policy [2] and XACML [7].
(e) An automated transformation from Policy PIM to platform specific policy model (Policy PSM). Policy PSM is specific to a policy domain selection of which is based on project or business requirements.
(f) An automated transformation from platform specific Policy model to platform specific policy description. The transformation reduces complexity, time and effort to generate policy descriptions.

In the next sections, we will describe the components shown in Figure 2 in more details.

IV. QoS Meta-Model

There are several QoS meta-model proposals which can be used to define and apply QoS profiles for Web services. Malfatti [8] introduced a suitable meta-model for our approach. Figure 3 shows the QoS meta-model used in our solution which is a slightly modified version of Malfatti. It is extensible and expressive enough to model standardized and arbitrary QoS attributes. The meta-model is created in Eclipse Modeling Framework (EMF-core), a powerful tool for designing models and their runtime support [9]. QoS profile developer uses this QoS meta-model to instantiate QoS profiles with QoS attributes. The basic elements of the QoS meta-model are:

- `Service`: Name of the Web service to apply policies. The Web service can have zero or more `QoSCategory` elements.
- `QoSCategory`: Defines categories with which quality criterions are grouped. Examples of a `QoSCategory` could be security, reliability and performance. Each `QoSCategory` has one or more `QoSParameters`.
- `QoSParameter`: It describes the quality criterions, e.g., `Inactivitytimeout` is a `QoSParameter` for reliability category. Each `QoSParameter` has exactly one `QoSAgreedValue` and a `QoSMetric` associated with it.
- `QoSAgreedValue`: The value of the criterion is defined in this element, e.g., the value 20 for `Inactivitytimeout`. This element can also be extended with `QoSProperty` elements.
- `QoSMetric`: This element specifies a unit with which the value of `QoSAgreedValue` element is measured e.g., "seconds" for the `QoSAgreedValue` 20 which is associated with the `QoSParameter Inactivitytimeout`.

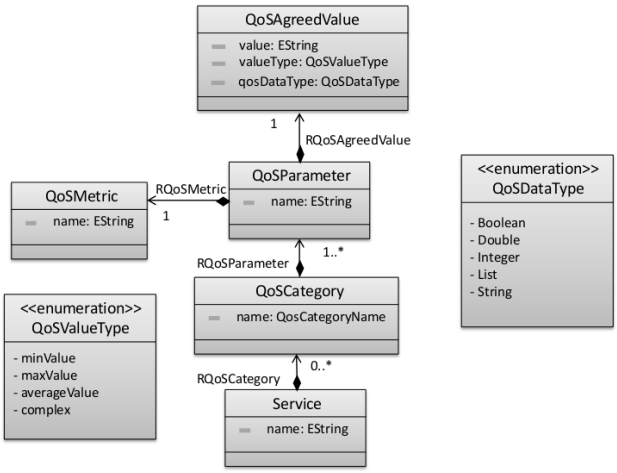

Figure 3. QoS meta-model.

The meta-model enables the QoS profile developer to model QoS attributes for Web services for different business domains. The following changes were made in the meta-model to the meta-model proposed in [8]:

- The `Category` attribute in the `QoSParameter` was modified to include only predefined values specified in the enumeration class `QoSCategoryName`.
- The `QoSLevel` was not considered in this work since the modeled QoS is always fulfilled.

Our meta-model is built using EMF (Core) [9], a modeling framework and code generation facility, which is used to build tools and applications based on a structured data model.

V. QoS Profile

QoS profile developer uses QoS meta-model to define and create QoS profiles. A QoS profile is an instance of the meta-model for a specific non-functional business requirement coupled with corresponding QoS criterions in default state or value. All the values of QoS attributes defined by QoS profile developer in QoS profiles are default values that

could be used by Web service developer while applying QoS attributes to the Web service. Web service developer can also change or configure the QoS attributes' values to fit business or project requirements while applying them to the Web service. Hence, QoS profiles are used by Web service developers during development to provide concrete QoS attribute values and apply them to their Web services as policies. Following, we will model three QoS profiles to demonstrate the flexibility of the meta-model. We will present a standardized QoS attribute from the WS-* family and introduce two non-standardized QoS profiles.

The first QoS attribute is from WS-ReliableMessaging. Figure 4 models QoS attributes described as a RM policy assertion example in [6], Section 2.4. In this example, `RQoSAgreedValue` "60000" for the `RQoSParameter InactivityTimeout` indicates that if the idle time exceeds 60000 milliseconds, the sequence will be considered as terminated by the service endpoint. `RQoSAgreedValue` "3000" for `RQosParameter BaseRetransmissionInterval` expresses that an unacknowledged message will be transmitted after 3000 milliseconds. `RQoSAgreedValue` "200" for `RQoSParameter AcknowledgementInterval` indicates that an acknowledgement could be buffered up to two-tenths of a second by the RM destination.

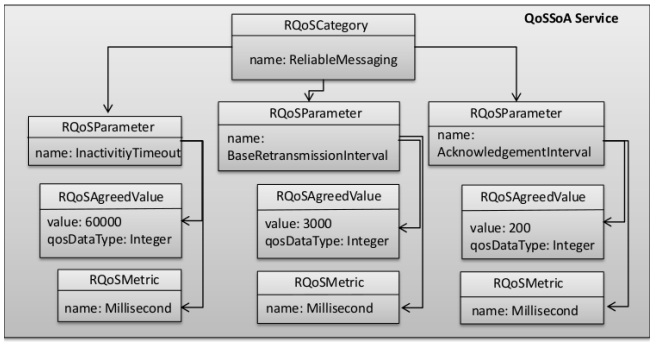

Figure 4. QoS profile for reliable messaging.

The second example models a QoS profile, which is not standardized. A calculator Web service performs arithmetic operations by accepting the operands and the operators. Figure 5 describes calculator Web service constraints for arithmetic operations such as addition and multiplication whose QoS implementation may differ with respect to number overflow. The calculator constraints set `minInt` and `maxInt RQoSParameters` for the Web service class or Web service methods. The `minInt` and `maxInt RQoSParameters` indicate that all the input and output numbers fall within the range of "0" and "65535". `qosDataType` in RQoSAgreedValue is set to `Integer` indicating the data type criterion of the value of `RQoSAgreedValue` for the calculator constraint.

Figure 6 shows the third example. It is a performance QoS

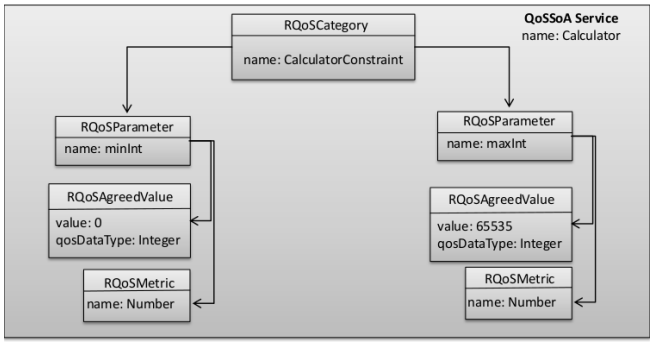

Figure 5. QoS profile for calculator example.

profile where `ResponseTime` and `Throughput` are QoS attributes, which are two of the most common used attributes in order to measure performance. Response time refers to the duration, which starts from the moment a request is sent to the time a response is received. Throughput is 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].

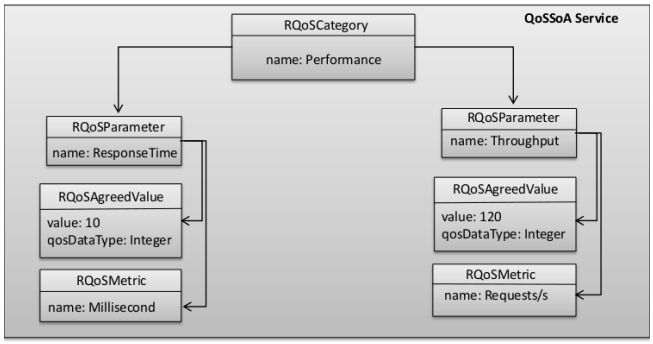

Figure 6. QoS profile for performance.

`RQoSAgreedValue` "10" for `RQoSParameter ResponseTime` indicates that the Web service shall guarantee a response within 10 milliseconds where `Millisecond` is defined as `RQoSMetric`. Similarly, RQoSParameter `Throughput` with RQoSAgreedValue "120" indicates that the Web service will be able to handle up to 120 request/second without having any change on the Web service performance. RQoSMetric defines the unit of measure for RQoSAgreedValue.

VI. Graphical User Interface

The graphical user interface enables Web service developers to configure or refine QoS profiles with new QoS attributes' values according to the business requirements. It is a graphical tool to associate the QoS values of the modeled QoS attributes.

There are two factors, which decide how the GUI should look like; the first factor is the QoS profile. The QoS profile

specifies the number of categories and the associated QoS attributes with their default values. In our approach, every QoS category is represented by a GUI tab and each tab shows the QoS attributes of the respective QoS category. Such a design ensures a user friendly management and division of QoS attributes based on their categories. For example, if the QoS profile includes three QoS categories performance, reliable messaging and calculator constraints, the QoS profile will be transformed into a GUI, which has three tabs. Each tab will represent a category. If the QoS category, e.g., calculator constraint has two QoS attributes, the QoS category tab on the GUI will represent these two QoS attributes, i.e., `minInt` and `maxInt` with their default values as shown in Figure 7.

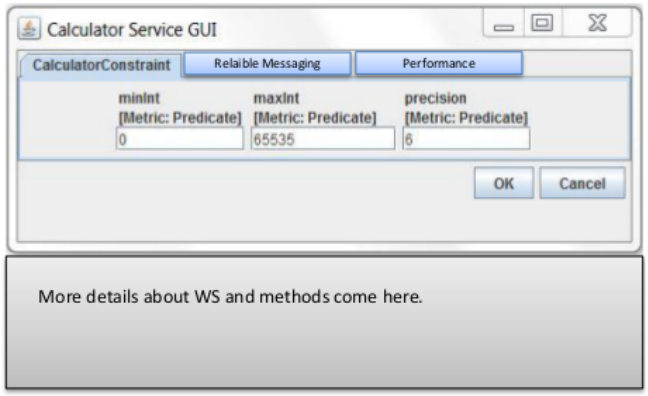

Figure 7. Generated GUI from the QoS profile.

The element `QoSMetric` helps the GUI engine to determine, how the `QoSAgreedValue` shall be presented. For example, if the `QoSMetric` indicates that the QoS attribute is a `Number`, the GUI engine will use a text field 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. A QoS profile can be associated either to a simple Web service or a set of Web services, i.e., all Web services contained in a WSDL description. If, for example, two Web service methods have two different two different values for "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 also be applied Web service wide. These possibilities give the Web service more flexibility and dynamics.

The QoS profile is transformed into the GUI code using a set of three separate transformation processes. A model-to-model transformation from QoS profile to GUI platform independent model (GUI PIM), which introduces an additional layer of abstraction supporting multiple GUI platforms, a model-to-model transformation from platform independent GUI model to platform specific GUI model (GUI PSM), for a specific GUI technology such as Swing and SWT and finally a model-to-code transformation from GUI platform specific model to GUI platform specific code, which on execution generates the GUI output. Every transformation is performed based on transformation rules that are defined using transformation languages. The two model-to-model transformations described above are based on the language "Operational Query View Transformation" (QVTO) [11] while the model-to-code transformation is based on "Xpand2" [12].

The transformation from QoS profile to GUI PIM is based on the mappings between QoS profile elements and the GUI elements. The GUI elements are defined in a GUI PIM meta-model, which is used to generate GUI PIM ensuring the extendibility of the solution to other GUI platforms. The GUI PIM meta-model is an abstract GUI model consisting of basic GUI elements. Figure 8 outlines the GUI PIM meta-model. At the root of the GUI PIM meta-model, is a `GUI` element associated with exactly one `Frame` element in which all the GUI elements are contained. The next level of abstraction of the GUI elements is the `Container` element which can contain `GUIElement`. The `GUIElement` is another abstract form of basic GUI elements divided under four categories, i.e., `ActionElement` (e.g., buttons), `OutputElement` (e.g., labels), `InputElement` (e.g., text fields) and `ChoiceElement` (e.g., list box) [13]. The architecture of the GUI meta-model is similar to the schematics of the QoS meta-model.

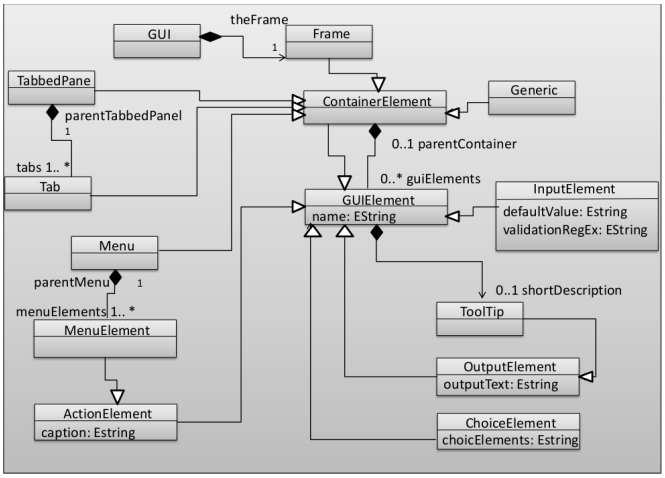

Figure 8. GUI PIM meta-model.

The transformation rules ensure a consistent mapping from different elements of the QoS profile to GUI PIM. Each of the QoS categories from the QoS profile is defined as different tabs in GUI. The quality criterion itself is described with a name, value and a metric representation. Depending on the complexity of the value element, respective mapping is performed, for example, on text field or text area.

The transformation from GUI PIM to GUI PSM is also performed through transformation rules. Essentially, the GUI PSM is platform specific extension of properties of the GUI PIM that contains technology specific entities. The GUI PSM, thus, inherits the elements of GUI PIM and refines them.

The working example shown in Figure 7 is specific to Java Swing framework. The mappings of the elements are performed by simply associating the corresponding elements from the two models. For example, Java Swing element JLabel inherits from the label element of the GUI PIM called Label. Finally from the GUI PSM, the code is generated through a model-to-code transformation. The mappings of each element of the QoS model to GUI PIM and further to Swing specific types are shown in the Figure 9.

QoS Model	GUI PIM Model	Swing Type
Service	Frame	JFrame
QoSCategory	Tab	JTabbedPane
QoSParameter	Generic	JPanel
QoSParameter	Label	JLabel
QoSMetric	Label	JLabel
QoSAgreedValue	TextField or TextArea	JTextField or JTextArea

Figure 9. Mappings from QoS profile to GUI PIM model and corresponding Swing specific types.

The mapping of GUI PSM (Java Swing) into the Java Swing code is straight forward since the elements of GUI PSM bear the same Swing names.

VII. Refined QoS Profile

Web service developer interacts with the generated GUI to refine the QoS attributes' values further according to the business requirements. Hence, the refined QoS profile (see (4) in Figure 2) is an extended version of the QoS profile (see (2) in Figure 2) with well-defined values of the QoS attributes. This component in the solution concept is the first step in building the policy descriptions for the Web services. After entering the values to the QoS attributes in the GUI (see (3) in Figure 2), the refined QoS profile is produced with the set values to QoS attributes. During this process, GUI reads the entered values of QoS attributes. QoS elements in the GUI are searched and matched with the corresponding attributes modeled in the QoS profile. Finally, a refined QoS profile is generated with the entered QoS values assigned to the respective QoS attributes.

Figure 10 shows the refined QoS profile of the calculator service example mentioned earlier in Figure 5 with new values to RQoSParameters `minInt` and `maxInt` as "-32768" and "32767" respectively. The new values to RQoSParameters represent the non-functional requirements set by the Web service developer over the calculator service.

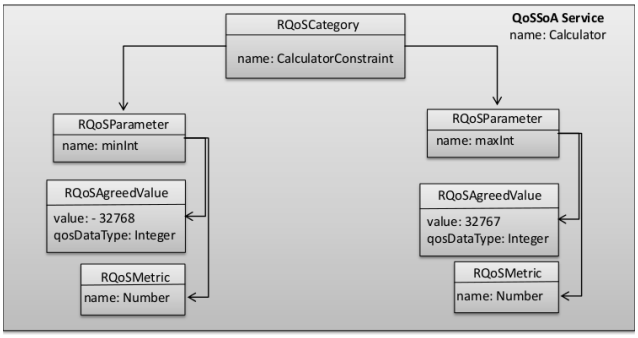

Figure 10. Refined QoS profile for Calculator.

Refined QoS profile is generated by iterating through the elements of GUI PSM, extracting the corresponding elements of QoS profile model with the refined values of QoS attributes (set via the GUI), reframing the quality criterions and finally storing the quality criterions in the refined QoS profile model.

It is of great interest to realize the importance of generating a refined QoS profile component than directly generating the policy model. The refined QoS profile provides the extensibility and flexibility to the generation of policies and applying them to the Web services. It provides an additional layer of abstraction to generate the policy descriptions through policy PIM thereby allowing the solution to extend to multiple policy languages through policy PSM (see Figure 2). Each QoS profile can then extend the Web service with profile specific non-functional quality criterions. This also reduces complexity of integrating multiple profiles into the Web services.

After successful generation of refined QoS profile from the GUI, further transformations to policy model take place. The QoS policy model and the transformations are discussed in the next section.

VIII. QoS Policy

As a Web service developer refines the QoS attributes on the GUI, all QoS values are assigned to the `QoSAgreedValue` element in the refined QoS profile. Once the QoS values have been assigned and refined QoS profile is created, a QoS policy model is 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 description may include the specifications of more than one QoS attribute depending on the user input in the GUI.

The transformation of the refined QoS profile instance to policy model can also be divided into three different transformation processes. A model-to-model transformation

from refined QoS profile to platform independent policy model (Policy PIM), which introduces an additional layer of abstraction supporting multiple policy languages. Another model-to-model transformation from policy PIM to platform specific policy model (Policy PSM) supporting specific policy languages such as WS-Policy and XACML. And finally a model-to-code transformation from policy PSM to platform specific policy description code. These transformations are performed based on certain transformation rules that are also defined using the transformation languages QVTO and XPand2.

The policy description elements are defined in a policy PIM meta-model which is similar to the GUI PIM meta-model described in Section VI. It is used to generate policy PIM ensuring the extensibility of the solution to other policy formalisms. The policy PIM meta-model is an abstract policy model consisting of four basic policy elements outlined as `ServicePolicy`, `AssertionGroup`, `Assertion` and `Property` [14]. Figure 11 describes the policy PIM meta-model. `ServicePolicy` is the root element which can encapsulate any number of `AssertionGroup` elements as child elements. `AssertionGroup`s can have any number of `Assertion`s as child elements. A `Property` can be assigned to any of the mentioned elements. A `Property` extends an element with additional information.

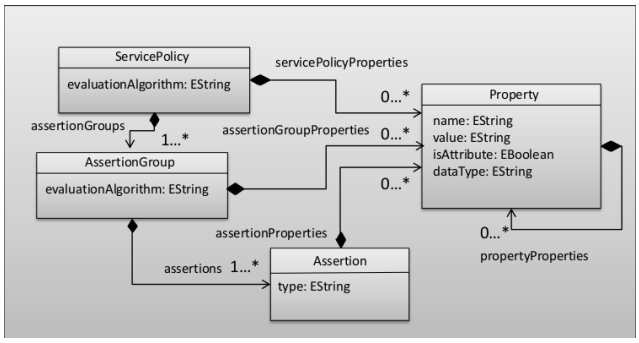

Figure 11. Policy PIM meta-model.

Once the Web service developer refines the QoS profile instance with new QoS values using the GUI, the refined QoS profile to policy PIM transformation process gets executed. This model-to-model transformation is based on transformation rules defining the logical mappings between the elements of refined QoS profile and rolicy PIM. Figure 12 outlines the mappings.

The transformation from policy PIM to policy PSM is also a model-to-model transformation process. This transformation process can yield multiple policy language specific policy PSMs. Once the policy PSM is created, the policy code is generated via model-to-code transformation XPand2 process.

The following transformation rules are applied to generate policy PIM from refined QoS profile:

Refined QoS Model	Policy PIM Model
Service	ServicePolicy
QoSCategory	-
QoSParameter	Assertion

Figure 12. Refined QoS profile to policy PIM mapping.

- The `Category` attribute in the refined QoS profile 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 refined QoS profile 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 refined QoS profile 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 refined QoS profile declares a QoS value. The value “60000” will be mapped as a value for the QoS attribute.

Listing 1 shows the modeled QoS in Figure 4 after the transformation into a reliable messaging WS-Policy assertion.

```
1 <wsp:Policy wsu:Id="ReliableMessagingPolicy">
2 <wsp:ExactlyOne>
3 <wsp:All>
4  <wsrm:InactivityTimeout Milliseconds="60000" />
5  <wsrm:BaseRetransmissionInterval
6      Milliseconds="3000" />
7  <wsrm:AcknowledgementInterval
8      Milliseconds="200" />
9 </wsp:All>
10 </wsp:ExactlyOne>
11 </wsp:Policy>
```

Listing 1. Reliable messaging WS-Policy description.

Similarly, the WS-Policy descriptions of calculator constraint and performance are generated after their profile transformation into policy assertions. Listing 2 shows the modeled and refined QoS in Figure 10 after the transformation into a calculator constraint WS-Policy assertion.

Listing 3 shows the modeled QoS in Figure 6 after transformation into WS-Policy performance assertion.

Once the policies have been created, policy model (see (4) in Figure 2) will assign the created policies to the Web service endpoint interface. There are several ways to associate a WS-Policy description to a Web service. In our

```
<wsp:Policy wsu:Id="CalculatorConstraintPolicy">
<wsp:ExactlyOne>
<wsp:All>
    <wscal:minInt Number="-32768"> </wscal:minInt>
    <wscal:maxInt Number="32767"> </wscal:maxInt>
 </wsp:All>
 </wsp:ExactlyOne>
 </wsp:Policy>
```

Listing 2. Calculator service WS-Policy description.

```
<wsp:Policy wsu:Id="PerformancePolicy">
<wsp:ExactlyOne>
<wsp:All>
 <wsrm:ResponseTime Milliseconds="10" />
 <wsrm:Throughput Requests/s="120" />
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Listing 3. Performance WS-Policy description.

proof of concept, we use the CXF policy [15] 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 corresponding Web service while creating its WSDL.

IX. Proof Of Concept

The solution architecture (Figure 2) discussed in the previous sections can be instantiated in several ways. For our proof of concept we have used a Java based infrastructure. To be precise, the following technologies are used:

- Eclipse IDE - Eclipse Modeling Tools [9].
- QVTO [11] and XPand2 [12] transformation frameworks.
- Java Swing GUI implementation.
- Apache Tomcat application server [16].
- Apache CXF services framework with WS-Policy support [15].

Eclipse Modeling Tools facility is used to build tools and applications based on a structured data model. It provides a pluggable framework to store the model information and by default uses XML Metadata Interchange (XMI) format to preserve the model definition. QoS meta-model in our proof of concept is based on Eclipse Modeling Tools.

WS-Policy [2] provides a policy language to formally describe properties of a behavior of Web services. WS-Policy itself does not come with concrete assertions. Related specifications introduce domain specific assertions, e.g., WS-Security for the security domain. The respective specifications do not only define the syntax, but also the meaning of the assertions and their impact on the Web services runtime behavior.

Operational Query View Transformation (QVTO) is used to create transformation rules for model-to-model transformations. It is the sublanguage of Query View Transformation (QVT) [17].

XPand2 is used to perform model-to-code transformations. XPand2 is administered in Eclipse Modeling Project (EMP) but has its roots from OpenArchitectureWare Framework. In our proof of concept, XPand2 is used to generate a Swing based GUI.

Apache CXF is an open source services framework. Apache CXF helps in building and developing services using frontend programming APIs, like JAX-WS and JAX-RS. Apache CXF includes a broad feature set, but it is primarily focused on supporting Web service standards including WS-Policy and frontends.

The main purpose of Apache CXF in our example is WS-Policy support it offers and its possibility to associate WSDLs with existing WS-Policies through annotations.

This section uses the calculator example mentioned in this paper to attach quality attributes and to demonstrate the overall working of the solution.

We begin with Eclipse Modeling Tools and define the QoS meta-model. QoS profile developer uses and instantiates QoS meta-model to provide QoS profile. QoS profile for a simple calculator is generated with two quality attributes, i.e., `minInt` and `maxInt`. The QoS profile, then, undergoes two automatic QVTO transformations, i.e., QoS profile to GUI PIM, and GUI PIM to GUI PSM. Listing 4 shows a sample transformation rule of QoS profile to GUI PIM.

```
1  modeltype QOS 'strict' uses
2        QoSSOAMetaModel('http://qosmetamodel/1.0');
3  modeltype GUI_PIM 'strict' uses
4        guipimmetamodel('http://guipimmetamodel/1.0');
5  transformation QoS2GUIPIMTransformation
6        (in qos : QOS, out guiPim : GUI_PIM);
7
8  property validationRegExDouble = '/\\d+[\\.|,]\\d+/';
9  property validationRegExInteger = '/\\d+/';
10
11 main() {
12    qos.objects()[Service]->xmap serviceToGUI();
13 }
14
15 mapping Service::serviceToGUI() : GUI
16 when {
17    self.RQoSCategory->size() > 0;
18 }
19 {
20    theFrame := object Frame {
21       name := self.name + ' GUI';
22    };
23    ...
24    ...
25    ...
```

Listing 4. Sample .qvto code for QoS profile to GUI PIM transformation.

The participating models are defined by using the keyword `modeltype`. Keyword `transformation` describes the source and the target model. The mapping functions are defined and called from within the `main()` function, which is called on the execution of the transformation file. Lines 20-21, for example, add `Frame` to the GUI. GUI PSM in our proof of concept is a GUI implementation, which is generated via transformation rules specific to the Java Swing environment from the abstract GUI PIM. The transformation

follows the similar syntax as described for QoS profile to GUI PIM transformation. Listing 5 shows the sample transformation.

Once the GUI PSM is generated, the code is created automatically via model-to-code Xpand2 transformation process. XPand2 template file reads the elements of the GUI PSM and translates them into Java Swing elements' code. On execution, it presents the Web service developer with the GUI. Figure 7 shows the generated GUI for our proof of concept calculator example with default values. Web service developer can interact with the GUI and insert new values to the quality criterion of calculator example.

```
1 modeltype GUI_PIM 'strict' uses
2  guipimmetamodel('http://guipimmetamodel/1.0');
3 modeltype GUI_PSM 'strict' uses
4 guipsmswingmetamodel('http://guipsmswingmetamodel/1.0');
5 transformation GUIPIM2GUIPSMTransformation
6  (in pim : GUI_PIM, out psm : GUI_PSM);
7
8 main() {
9  pim.objects()[GUI]->map guiToGUI();
10 }
11
12 mapping GUI::guiToGUI() : GUI {
13  theFrame := object JFrame {
14      name := self.theFrame.name;
15  };
16  theFrame.guiElements +=
17  self.theFrame.guiElements->select
18  (elem | elem.oclIsTypeOf(TabbedPanel))
19  .oclAsType(TabbedPanel)
20 ->map tabbedPanelToJTabbedPane();
21
22  theFrame.guiElements +=
23  self.theFrame.guiElements->
24  select(oclIsTypeOf(Generic)).oclAsType(Generic)->
25  map genericToJPanel();
26  ...
27  ...
28  ...
```

Listing 5. Sample .qvto code for GUI PIM to Java Swing transformation.

Each generic element from the GUI PIM is mapped with respective Java Swing GUI element. Lines 16-20 show the mappings of GUI PIM element `TabbedPanel` to the corresponding GUI PSM (Java Swing) element `JTabbedPane`. Figure 9 shows the mappings of the elements.

The process of generating refined QoS profile is similar to the process of instantiation of QoS meta-model described before to generate QoS profile.

The refined QoS profile undergoes two model-to-model QVTO transformations, i.e., refined QoS profile to policy PIM and policy PIM to policy PSM as well as a model-to-code XPand2 transformation from policy PSM to get WS-Policy code.

Refined QoS profile to policy PIM transformation is similar to the transformation described above, i.e., QoS profile to GUI PIM. Listing 6 shows the sample transformation. Lines 19-21 maps the `QoSParamaters` of a `QoSCategory` to respective assertions.

Policy PSM in our proof of concept is WS-Policy specification. It is generated from the abstract Policy PIM via

```
1 modeltype QOS 'strict' uses
2  QoSSOAMetaModel('http://qosmetamodel/1.0');
3 modeltype PIM "strict" uses
4  "http://webuser.hs-furtwangen.de/~passfall/PIM";
5
6 transformation QoS2PolicyModellTransformation
7     (in qos : QOS, out policy : PIM);
8
9 main() {
10  qos.objects()[Service]->
11          xmap Service2ServicePolicy();
12 }
13
14 mapping Service::Service2ServicePolicy() : ServicePolicy
15 {
16  evaluationAlgorithm := self.name + ' Policy';
17  var assertionGroupElement :=
18  object AssertionGroup {};
19  assertionGroupElement.assertions +=
20  self.RQoSCategory.RQoSParameter ->
21  xmap Parameter2Assertion();
22  assertionGroups += assertionGroupElement;
23  ...
24  ...
25  ...
```

Listing 6. Sample .qvto code for refined QoS profile to policy PIM transformation.

```
1 modeltype PIM "strict" uses
2  "http://webuser.hs-furtwangen.de/~passfall/PIM";
3 modeltype WSP "strict" uses
4  "http://webuser.hs-furtwangen.de/~passfall/PSMWSPolicy";
5 transformation PIM2WSP(in Source: PIM, out Target: WSP);
6
7 helper findNamespaces
8  (param : Set(PIM::Assertion)) : List(WSP::Namespace) {
9
10 var nss : List(WSP::Namespace);
11 param->switch(s) {
12  case (s.type = "SignedElementsIndicator") {
13      var ns := object Namespace {
14      prefix := "sp";
15      url :=
16      "http://docs.oasis-open.org/ws-sx/
17              ws-securitypolicy/200702";
18      };
19  ...
20  ...
21  ...
22 }
23
24 main() {
25  Source.objects()
26      [ServicePolicy]->xmap ServicePolicyToPolicy();
27 }
28
29 mapping ServicePolicy::ServicePolicyToPolicy(): Policy {
30  var assertions := Source.objectsOfType(PIM::Assertion);
31  namespaces := findNamespaces(assertions);
32  alternatives+=self.assertionGroups->
33  xmap AssertionGroupToAlternative();
34  ...
35  ...
36  ...
```

Listing 7. Sample .qvto code for policy PIM to WS-Policy transformation.

transformation rules specific to the WS-Policy specifications. Listing 7 shows the sample transformation. Each generic element from the Policy PIM is mapped with a respective WS-Policy specific element. Line 26 is the function call to ServicePolicytoPolicy function. Line 32 performs the mapping of `AssertionGroup` to WS-Policy `Alternative`.

Figure 13 shows the mappings of the elements.

Policy PIM Model	Policy PSM Model (WS-Policy)
ServicePolicy	Policy
AssertionGroup	Alternative
Assertion	Assertion
Property	AssertionProperty

Figure 13. Policy PIM to WS-Policy mappings.

Once the policy model for WS-Policy is generated, the application of the policies is attached to the Web service. Now, Apache CXF policy engine attaches the corresponding policy to either the selected Web service method or the Web service endpoint interface. Apache CXF uses the @POLICY annotation to signal the compiler that there are policies, which should be considered and assigned to the corresponding Web service while creating the Web service WSDL.

X. Related Work

In our research for related work, an 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 [18]–[20]. In this section, we will describe papers, which propose either QoS meta-models or policy editors.

Tondello et al. [21] proposes a QoS-Modeling 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 service descriptions 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.

Suleiman et al. [22] addresses the problem with Web service management policies during design. The authors presented a solution, which uses a novel mechanism. It generates WS-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 [23] introduced a WSDL extension for describing the QoS of a Web service. It uses a meta-model transformation according to the MDA standard. The WSDL meta-model is extended and transformed into a new WSDL model called Q-WSDL, which supports QoS descriptions. 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 attributes in Web services.

WSO2 WS-Policy editor [24] 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 [25] 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 [26].

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

XI. Future Work

In [27], 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 VI, we introduced a GUI, which is dynamically generated depending on the QoS attributes modeled in the QoS profile described in Section V. 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 VII indicates that the QoS profile 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 [28], 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 modeled 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 handler implementation, handler configurations, and test cases.

XII. Conclusion

The design and implementation of an SOA that contains QoS attributes is difficult. There are tools and IDEs, which help developers to ease the process of creating programs, minimizes their error rates, designing and implementing such complex systems. But, to create a QoS policy and conjugate it with a Web service still requires a good knowledge of its grammar and its mechanism. It is highly desirable to have tools, which help developers to model QoS attributes, simplify the configuration and automate applying QoS attributes to Web services. First steps towards such tools have been made, but the overall support for developers needs to be highly improved.

In this paper we presented a tool chain, which increases the support for two developer roles: QoS profile developer and Web service developer. The former, by the use of a generic QoS meta-model, defines QoS profiles for targeted QoS attributes. The approach thereby support the modeling of new QoS attributes. From a QoS profile a corresponding GUI is generated, which supports the Web service developer to refine the QoS profile, to generate a policy model, and to apply the corresponding policies to Web services. Implementation details, like the usage of WS-Policy and adding necessary annotation in source code, are hidden from the Web service developer.

Throughout the paper we described the necessary modifications, user interactions and transformations step by step. At the end, the feasibility of the approach was shown by a proof of concept.

In summary, the approach is a major step towards an increased support for constructing QoS-aware Web services. It eases and unifies the development process and helps to reduce the error rate, development effort and the overall costs.

XIII. Acknowledgment

We would like to thank the anonymous reviewers for giving us helpful comments.

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

References

[1] A. Al-Moayed and B. Hollunder, "An approach to model, configure and apply QoS attributes to web services," in *Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011)*. Xpert Publishing Services, 2011, pp. 405–410.

[2] W3C, "Web Services Policy Framework - Version 1.5," September 2007, last access: 20.12.2012. [Online]. Available: http://www.w3.org/TR/ws-policy/.

[3] 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.

[4] B. Hollunder, M. Hüller, and A. Schäfer, "A methodology for constructing ws-policy assertions," in *Proceedings of the 2nd International Conference on Engineering and Meta-Engineering (ICEME 2011)*, 2011, pp. 112–117.

[5] OASIS, "Web Services Security Policy - Version 1.3," April 2009, last access: 20.12.2012. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/.

[6] OASIS, "Web Services Reliable Messaging Policy - Version 1.2," February 2009, last access: 20.12.2012. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.pdf.

[7] OASIS, "eXtensible Access Control Markup Language (XACML) Version 2.0," OASIS, February 2005, last access: 20.12.2012. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

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

[9] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, E. Gamma, L. Nackman, and J. Wiegand, Eds. Addison-Wesley Professional, 2008.

[10] E. Kim, Y. Lee, Y. Kim, H. Park, J. Kim, B. Moon, J. Yun, and G. Kang, "Web service quality factors version 1.0," OASIS, Tech. Rep., 2011, last access: 20.12.2012. [Online]. Available: http://docs.oasis-open.org/wsqm/WS-Quality-Factors/v1.0/WS-Quality-Factors-v1.0.pdf.

[11] R. Dvorak, "Model Transformation with Operational QVT," Borland Software Corporation, 2008, http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf, last access: 20.12.2012. [Online]. Available: http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf.

[12] E. Galileo, "Xpand/ Xtend/ Check Reference," The Eclipse Foundation, last access: 20.12.2012. [Online]. Available: http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.xpand.doc/help/ch01s06.html.

[13] M. Hermann and A. Hülzenbecher, "M2M-Transformation zur Generierung einer grafischen Benutzeroberfläche in einem QoS-SOA Kontext," Hochschule Furtwangen University, 2011, informatik Journal, Faculty of Informatics.

[14] A. Passfall, T. Rübsamen, and R. Teckelmann, "Modellbasierte Erzeugung von Policy-Dokumenten," Hochschule Furtwangen University, 2011, informatik Journal, Faculty of Informatics.

[15] "Apache CXF: An Open-Source Services Framework," The Apache Software Foundation, last access: 20.12.2012. [Online]. Available: http://cxf.apache.org/.

[16] "Apache Tomcat," The Apache Software Foundation, last access: 20.12.2012. [Online]. Available: http://tomcat.apache.org/.

[17] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," last access: 20.12.2012. [Online]. Available: http://www.omg.org/spec/QVT/1.0/.

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

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

[20] H. Kil and W. Nam, "Anytime Algorithm for QoS Web Service Composition," in *Proceedings of the 20th international conference companion on World wide web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 71–72, last access: 20.12.2012. [Online]. Available: http://doi.acm.org/10.1145/1963192.1963229.

[21] G. Tondello and F. Siqueira, "The QoS-MO Ontology For Semantic QoS Modeling," in *Proceedings of the 2008 ACM symposium on Applied computing*, ser. SAC '08. New York, NY, USA: ACM, 2008, pp. 2336–2340, last access: 20.12.2012. [Online]. Available: http://doi.acm.org/10.1145/1363686.1364239.

[22] 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*, 2009, pp. 75–82.

[23] A. D'Ambrogio, "A Model-driven WSDL Extension for Describing the QoS of Web Services," in *Web Services, 2006. ICWS '06. International Conference on*, sept. 2006, pp. 789–796.

[24] WSO2, "WSO2 WSAS: The WS-Policy Editor 3.2.0 - User Guide," WSO2, April 2010, last access: 20.12.2012.

[25] "NetBeans IDE," Oracle Corporation, last access: 20.12.2012. [Online]. Available: http://netbeans.org.

[26] "Glassfish Application Server," last access: 20.12.2012. [Online]. Available: http://glassfish.java.net.

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

[28] 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*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 27–33.