Volume 51, Number 4, 2010

ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

A CONCEPTUAL FRAMEWORK AND IMPLEMENTATION FOR DEVELOPING WS - QoS AWARE ARCHITECTURES

Cosmina IVAN, Vasile DADARLAT
Faculty of Automation and Computer Science, Technical University of Cluj Napoca
26-28, Baritiu,,Cluj-Napoca , Romania,
Email: cosmina.ivan@cs.utcluj.ro, vasile.dadarlat@cs.utcluj.ro

Abstract: Web Services and SOA are becoming more and more popular these days and more and more businesses are planning to build their future solutions on Web Services technology. By now, SOAP and WSDL have become reliable standards in the field of Web Services execution. While the concept of UDDI allows for automatic discovery of services implementing a common public tModel interface, there have been only few attempts to find a standardized form to describe the Quality of service (QoS) with which the service is performed. Nevertheless, the QoS a provider delivers will become a decisive feature when it comes to selecting one from many availably services providing the same functionality. Today, we have sophisticated technology to actively differentiate between various QoS levels both on transport level (DiffServ, IPQoS, CoS for UMTS/ATM/RCL) and server level (load balancing, transaction differentiation, HTTP request differentiation), yet there are only few means to describe the desired QoS on application layer. To tackle this, we propose a conceptual framework based on a Xml schema to declare both clients' QoS requirements and the QoS level service providers, and we have designed and implemented a framework for C# / .NET application developers to assign QoS requirements to a client application's service proxy which can then be used to inquire a QoS service broker for the best QoS offer fulfilling the requirements from all offers available for services that implement the specified interface.

Keywords: *Framework, QoS – Quality of Service , WS-Web Services, SOAP/WSDL/UDDI*

I. INTRODUCTION

The increasing industrial and academic involvement in the still emerging Web Service technology clearly shows the potential of Web services to become one of the pillars of the software industry. Competing Web services that implement same or similar functionalities are already and will be available on the market. As offered functionalities are similar, the quality of offered services will be decisive for the success of the service providers. While service offers with no guarantees on throughput, response time, security, availability, reliability, etc. are accepted in some simple cases, most likely this will not be acceptable when a Web service becomes an important part of an application composed of various Web services [5].

A service-oriented architecture (SOA) is essentially a collection of services that communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Hence some means of connecting services to each other is needed. A key driver for SOA implementations is the hope to save development time and costs through a higher degree of reuse of components in the form of readily implemented services [3],[4]. To achieve this aim it is necessary, among other things, to make Web services discoverable. SOA services have self-describing interfaces in platform-independent XML documents. Web Services Description Language (WSDL) is the standard used to describe the services.

SOA services communicate with messages formally defined via XML Schema. Communication among consumers and providers or services typically happens in heterogeneous environments, with little or no knowledge about the provider. SOA services are maintained in the enterprise by a registry that acts as a directory listing. Applications can look up the services in the registry and invoke the service. Universal Description, Discovery, and Integration (UDDI) is the standard used for service registry. Web services are self-described software entities which can be advertised, located, and used across the Internet using a set of standards such as SOAP, WSDL, and UDDI. Web services encapsulate application functionality and information resources, and make them available through programmatic interfaces, as opposed to the interfaces typically provided by traditional Web applications which are intended for manual interactions. However, discovering web services using keyword-based search techniques offered by the existing UDDI registry does not yield results that are tailored to client's needs. Several web services may share similar functionalities, but possess different *non-functional* properties. When discovering web services, it is essential to take into consideration, the functional and non-functional properties in order to render an effective and reliable service selection process.

Nowadays, both Web Service providers and clients are concerned with the QoS guaranteed by web services. From the client point of view, web service based QoS discovery is a multi-criteria decision mechanism that requires knowledge about the service and its QoS description. However, most of clients are not experienced enough to acquire the best selection of web service based on its described QoS characteristics. They simply trust the QoS information published by the provider; however most of web services providers do not guarantee and assure the level of QoS offered by their web services.

Based on the above we propose a Web Services discovery architecture that contains an extended UDDI to accommodate the

Manuscript received September 2, 2010; revised December 9, 2010

QoS information, and QoS Broker to facilitate the Web Service discovery. Measuring the degrees to which the web services can deliver the functionality through a combination of QoS parameters becomes significant, particularly in distinguishing services competing in the same domain.

The QoS parameters can be used to characterize the web services' overall behavior. Service providers QoS claims may not be trustworthy. Hence some method is needed to automate the process of measuring QoS for registered web services. Current UDDI registries don't have built-in-capabilities to validate or monitor published web services. They include only metadata about businesses and their related web services. If the UDDI registries let service providers publish their QoS claims, they could publish false or inaccurate information or the published information could be passive or outdated. Hence the clients should be able to obtain web service information based on QoS metrics from a trusted service broker.

QoS delivered to a client may be affected by many factors, including the performance of the web service itself, the hosting platform and the underlying network. A set of verification procedures is essential for providers to remain competitive and for clients to make the right selection and trust the published QoS metrics. For the success of any QoS based web services architecture, it should support a set of features: 1) QoS Verification and Certification to guide web service selection 2) QoS aware web services publishing and discovery.

Traditionally, QoS is associated with network parameters such as bandwidth, packet loss rate, and jitter. However, QoS in the realm of Web services is more than just traffic parameters. Beyond the network aspects, QoS for Web services covers server performance, security, transactional, and monetary aspects, and all components and layers participating in the Web service communication process.

This paper targets mainly the following issues in Web service communication:

1. The definition of QoS aspects and parameters related to the Web service layer.
2. The efficient lookup and selection of services at runtime according to clients' requirements.
3. The mapping of QoS aspects and parameters, which are defined in the Web service layer, on the underlying communication layer and on the participating components.

The main contributions of this paper are the design and performance measurements of the QoS framework targeting the overall QoS support for Web services. Our QoS framework is based on the QoS XML schema that allows service clients and service providers to define QoS-aware requirements and offers. Furthermore, the QoS XML schema allows domains and components along the Web service communication process to actively support the clients' QoS requirements. The flexible and extensible WS-QoS framework addresses various aforementioned QoS aspects, not only the classical network aspect.

We have implemented our QoS framework. We have conducted performance measurements of our framework. The measurement results prove the advantages of applying our framework for QoS aware Web service communication.

Another outstanding part of this work in comparison to other Web service related efforts is that we consider QoS through different layers and components that participate in Web service communication. Users can define their QoS requirements due to various aspects on a higher level such as in the application layer by applying the QoS XML schema. These QoS aspects and their QoS parameters are evaluated and mapped at runtime to achieve QoS fulfillments.

The remainder of the paper is organized as follows. *Section 2* introduce the background information on WS, QoS and outlines the related research conducted in the area of web services discovery and QoS . In *Section 3*, we describe our proposed architecture. The fundamental goal of the design of the WS-QoS architecture is QoS support during the whole communication process. Our framework supports standard conformity, scalability, extensibility as well as QoS mapping between different layers in terms of the Internet model. *Section 4* presents the prototypical implementation of the framework. *Section 5* explains how to apply the implementation and presents some performance measurements and *Section 6* summarizes the paper and presents conclusions and possible future research in this direction.

II. RELATED WORK

2.1 Web services

According to the Web service specification available at W3C [6], the definition of a Web services "is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web related standards".

With the increasing popularity and importance of Web services, a number of Web service related standards have been defined beyond the basic Web service protocol stack in different areas such as transaction , security , addressing , discovery , composition , etc. The basic interaction model consists of three parties, service provider, service requestor, and the service registry. As shown in Figure 3, the service provider publishes its services in the service registry; the service client finds the required service in the service registry. After discovering an appropriate service in the service registry, the service client invokes this service at the service provider architecture (SOA) ,the functionality and information on where and how to access Web services (i.e. under which universal resource identifier (URI) using which protocol) is described in a WSDL file. This file is commonly referenced from a UDDI registry for standardized service discovery.

A UDDI registry is basically a data base with UDDI logic and interfaces for publishing and searching Web services [10]. Industrial categorization, contact information, and technical information about services such as contact information of a company or industry categories, can be stored and viewed there by either using a Web interface or an application program interface (API). However, entries are often inaccurate or expired [13] and UDDI does not provide a mechanism for automatically updating the registry as services (and service providers) change [16].

Since the communication between clients and UDDI is a client/server interaction, UDDI can become a performance bottleneck in case of overloading or even unavailability. Furthermore, [13] shows that performance considerations of current UDDI implementations do have a major impact on the overall acceptance of UDDI. Section 4 introduces the WS-QoS broker, which improves the standard Web services interaction model.

A Web service can be described by its static functional attributes and its dynamic non-functional parameters including QoS aspects. The different QoS aspects are for example : server performance including throughput, availability and reliability, network performance including bandwidth, jitter and delay, security and transactional support, configuration and management capabilities as well as cost.

With the emergence of functionally equivalent services implementing a common service type (e.g. tModel in terms of

UDDI), the non-functional QoS properties associated with services will become distinctive criteria for the success of a company offering e-business through Web services. It is also imaginable that a business will offer its services in various classes of quality to meet different customers' requirements, according to what they might be willing to pay in return. Therefore, the need to unambiguously specify both QoS properties and different QoS levels in some kind of contract such as SLA and to prove the Web services' compliance arises.

In terms of SOA, Web services are building blocks from which more sophisticated applications can be created. An introduction to current specifications for Web service composition is given in [2]. Our QoS framework does not deal with service composition but can be well applied for looking up appropriate services in a service composition process.[5]

Three different phases of QoS management can be identified. Firstly, QoS constraints on certain parts of a provided service are formulated in a specification. Since the purpose of such a specification is to reinforce a certain level of QoS, parameters are monitored by constant measurements at runtime. Finally, the values measured have to be tested against the negotiated specification and appropriate activities should be carried out in order to control the QoS conformance ensuring a low violation rate. In case of violations, compensation may be refunded to the service consumer.

The following subsection gives an overview of five selected major approaches towards QoS specification and management for Web services, coming from both the industry and the academia. These approaches[6,16] are :

- the Web Service Level Agreement (WSLA) developed by IBM ,
- the Web Service Offering Language (WSOL) developed at Carleton University, Canada ,
- SLAng developed at University College London, UK ,
- a UDDI eXtension (UX) developed at Nanyang Technological University, Singapore , and
- UDDIe developed at Cardiff University, UK .

Common denominators of the approaches are the use of XML and the conformance with the existing Web service technologies such as WSDL and UDDI. While WSLA fosters individually customized SLAs, WSOL introduces a formal specification of classes of service. SLAng can be used for SLA specifications in general not only for Web services, aiming at a wide usage. UX is able to select services based on reputation among federated UX servers. UDDIe extends the standard UDDI API in order to associate QoS properties with Web services.

In the following, we compare the introduced approaches based on different aspects. We don't give an overall assessment of each approach. The table gives rather an overview of the main emphases of the introduced approaches. Selected assessment criteria include requirement specification, class of service, QoS aspects, QoS mapping, and flexibility:

Requirement specification: Both Web service clients and providers need means to specify non-functional requirements and offers. The specification should ensure the compatibility and comparability of the specifications done by clients and service providers.

Class of service: QoS parameters differ in quality, quantity, and the corresponding monetary charge. Grouping similar parameters into a class or category that characterize a service will ease the utilization of the service.

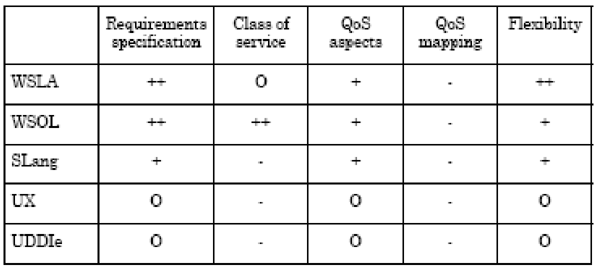
QoS aspects: A Web services related framework should support more than the classical QoS parameters such as jitter and bandwidth. Aspects such as security, reliability, transaction as well as custom defined aspects should also be considered.

QoS mapping: An overall QoS support requires QoS support during the whole communication process, ranging from the QoS specification to monitoring at runtime. QoS has also to be considered through the different layers in terms of the Internet Model. Specifications in higher layers have to be carefully mapped onto lower layers.

Flexibility: An approach should be easy to use, extensible, and standards conforming.

The assessment of the introduced approaches is summarized in Table 1. The symbols mean:

"++": excellent concept
"+": good concept
"O": satisfying
"-": poor or not available



	Requirements specification	Class of service	QoS aspects	QoS mapping	Flexibility
WSLA	++	O	+	-	++
WSOL	++	++	+	-	+
SLang	+	-	+	-	+
UX	O	-	O	-	O
UDDIe	O	-	O	-	O

Table 1 Assesment of the approaches

III. THE DESIGN OF THE ARCHITECTURE

The fundamental goal of the design of the QoS architecture is QoS support during the whole communication process. Standard conformity, scalability, extensibility have to be supported as well as QoS mapping between different layers in terms of the Internet model.

Our framework is fully compatible to standard Web service protocols such as SOAP, WSDL, and UDDI and targets the following main requirements:

- designing an architecture that allows both service clients and service providers to specify requests and offers with QoS properties and QoS classes,
- enabling an efficient service lookup and selection in order to accelerate the overall lookup process for service requestors,
- providing a flexible way for service providers to publish and update their service offers with different QoS aspects and parameters.
- considering the QoS requirements regarding different layers and participating domains of a Web service communication process at runtime in order to achieve overall performance gains.

2.2.The The QoS Xml Schema

There are three different kinds of root elements for a WS-QoS Xml Document: A *WSQoSRequirementDefinition* element specifies client QoS requirements. These are minimal requirements which must not be violated by underperformance. A *WSQoSOfferDefinition* element contains one or more specifications of QoS offers that a service provider is willing to deliver for a set price related to the defined QoS level.

Finally, a *WSQoSOntology* element holds definitions of QoS parameters and protocol references.

QoS Info

The most important of all elements are those of the type *tQoSInfo*, which hold information on the level of QoS regarding the server performance, transport QoS support
and protocol required for providing security and transaction support. In a *serverQoSMetrics* element, values for the standard parameters processing time, requests per second, reliability and availability can be declared. Moreover, custom server QoS metrics can be declared in a *customMetric* element as a child node of the *serverQoS-Metrics* element. In most cases, neither the client nor the service provider knows over what kind of network technology the messages will be exchanged.

Therefore, it does not seem appropriate to declare explicit values for metrics like the response time for a service. We have decided to declare transport QoS priorities, which are to be interpreted by the underlying network layer(s) and then mapped into specific metrics helping to provide a distinct transport service level. In a *transportQoSPriorities* element, priorities can be declared for the four standard transport parameters delay, jitter, throughput and packet loss rate.

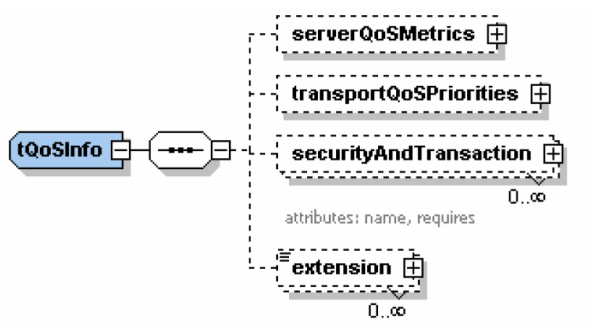


Figure 1. Structure of a QoSInfo element

Like with the server QoS metrics, custom transport QoS priorities can be declared in a *customPriority* element added to the *transportQoSPriorities* element. Security and transaction management for Web Services is realized by a variety of protocols. Most of them already have sophisticated mechanisms of negotiating key and session information. Therefore, security and transaction support at this level will be restricted to listingprotocols needed for a successful service execution. The *securityAndTransaction* element of a QoS info can holdseveral *protocol* elements, each referencing a specificprotocol.

A reference to a protocol in a QoS info can either require or offer compliance with the protocol in question. In the first case, another QoS info will not be compliant with the first specification if it does not at least offer using this protocol as well. In the later case the protocol is offered in case the other party expects it, but interaction without the protocol is also allowed.

QoS Ontology

Custom metrics, custom priority and protocol support statements all have an attribute ontology, which references a file containing a *QoS Ontology* where the referenced types are defined respectively. By using the combination of the ontology's Url and the parameter name, a reference is unique. A *custom transport QoS priority* is defined by a distinct name and a human readable definition of what metric the priority refers to in a *priorityDefinition* element.

A *custom server QoS metric* defined in a *metricDefinition* element also has a name and a human readable description of what is measured, but it also includes information on the standardized unit it is measured in and the scope of service
invocations the metric is aggregated on, that is, whether the value is valid for the port on which the service is invoked, the whole service or even all services of the provider.

Furthermore, it has to be stated whether the value is valid for all service executions or for executions requested by the user only. Finally, the direction of how values are to be compared is declared, which is essential for an automated check of whether an offer fulfills a set of requirement. Accordingly, in a *protocolDefinition* element, a *protocol* is defined by its name, a human readable description of the purposes of using this protocol and the Url of an overview document, the protocol specification if possible.

QoS Definition

An element of the type *tQoSDefinition* holds one or more QoS info elements plus specification of contract and management support and a specific price. QoS information can be defined for specific operations only in explicit *operationQoSInfo* elements or for the scope of all operations in a *defaultQoSInfo* element. Both the *defaultQoSInfo* and *operationQoSInfo* elements are of the type *tQoSInfo* as explained above. The *contractAndMonitoring* node can hold references to protocols needed for service management and/or QoS monitoring and entries of third parties that on one side would be willing to trust. Finally, the *price* element relates the specified QoS level to the cost of service usage per invocation. Elements of the type *tQoSDefinition* are either instantiated as a *WSQoSRequirementDefinition* element expressing a client's QoS requirements or as a *qosOffer* representing a minimal QoS level a service provider guarantees to provide for all requests where this offer is selected. The *qosOffer* element is extended by an attribute expires which denotes a point in time until which the offer will be valid.

QoS Offer Definition

Offers for one service can be declared in a *WSQoSOfferDefinition* element which is introduced into the service's WSDL file as an extension element of the service
description's service node. Apart from defining offers in a *WSQoSOfferDefinition* element, offers in further QoS files can be referenced in an include element. This allows for dynamically adjusting offers without changing the WSDL file. Furthermore, an offer could be referenced from multiple WSDL files and thus be reused for different services.

The QoS XML schema is the core of the WS-QoS architecture. The QoS XML schema enables the specification and thus the compatibility and comparability of QoS statements defined by both service clients and servers. All components participating in Web service communication such as WSB, Adaptation Layer, web servers apply the QoS XML schema in order to provide QoS support in different layers and domains.

Three steps are defined in a Web service communication process from the client's point of view. They are the definition of requirements, service discovery and selection, and service invocation. The QoS architecture ensures QoS-awareness during the whole Web service communication process resulting in a QoS-aware cross-layer communication. In contrast to the classical ISO/OSI layered architecture that does not consider the inter-working of different layers, the cross-layer communication model has several advantages:

- Higher layers have knowledge about the parameters and routing algorithms of underlying network technologies

- •Higher layers have knowledge about the current communication structures and their dynamics
- •Resulting in higher layers can actively consume the QoS support of the low layers
- •Lower layers have knowledge about the specific requirements from the higher layers
- •All the knowledge can be merged in respect to QoS parameters of different aspects in order to support application-dependent requirements.

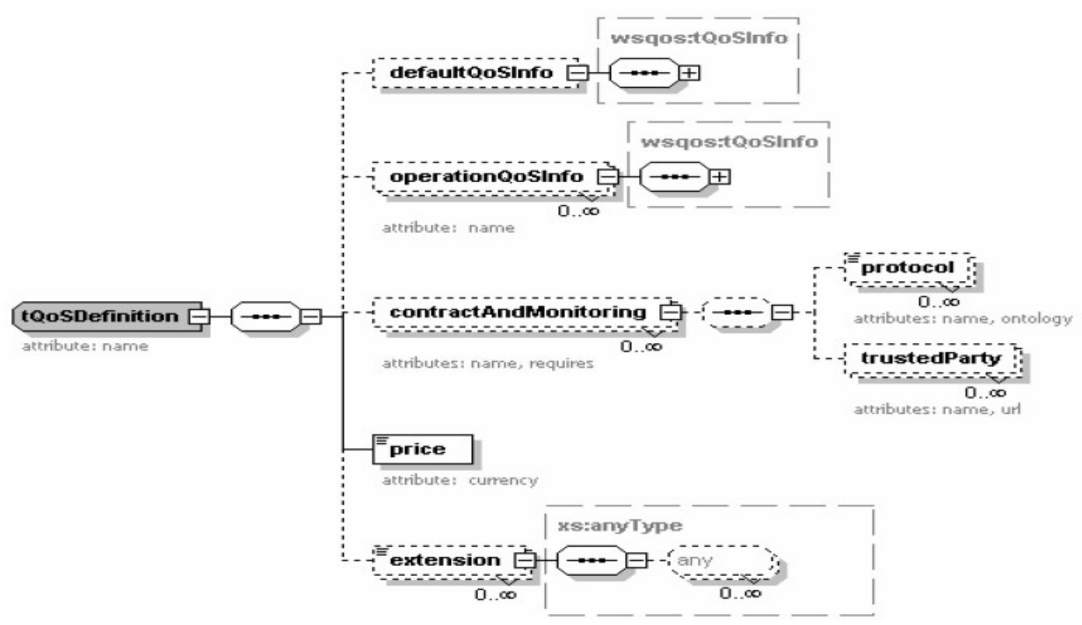


Figure 2. Structure of a QoSInfo element

IV. THE PROTOTYPE IMPLEMENTATION

Due to the requirements discussed in Section 3, we implemented the QoS framework with the following objectives:

- The QoS API that allows C# and ASP .NET application developers to define QoS requirements for both client applications and Web service offers
- The QoS Editor that allows the editing of the QoS parameters through a graphic user interface (GUI)
- The Requirement Manager that is responsible for retrieving clients' requirements
- The Broker, which is responsible for the QoS-aware service selection .
- The QoS Monitor, which is used for examining the compliance of offers

A scenario for QoS-aware service selection

From the client point of view, a client application can use one or more types (*tModel*) of Web services. The interfaces described in WSDL of Web services are known at the implementation time. A proxy class (in the context of Microsoft Visual Studio .NET also known as a Web Reference) is generated from the tModel's WSDL description for each service type. Static WS-QoS custom attributes or *import* attributes referencing dynamic requirements in a WS-QoS XML file can now be assigned to the newly created proxy class and its methods (known as web methods in Visual Studio .NET). Finally, the proxy class is handed over to an instance of the QoS Requirement Manager, which will retrieve the attributes through the reflection technique and thus holds a representation of the current client requirements. One can also use the Requirement Manager to adjust the QoS requirements at runtime without recompiling any code.

On initialization, the client application creates an instance of a QoS Requirement Manager. A QoS Broker (WSB) is already running in the same network. Before the service invocation, the client application will use the Requirement Manager to state its current QoS requirements and then inquire the WSB for the most appropriate service offer available that fulfills its requirements. The WSB selects the most appropriate offer on behalf of the client from the WSBs local database. We assume in this case that the WSB has already a local and up-to-date cache of the services the client is asking for. Therefore, the WSB does not contact any UDDI and service providers for searching the required service. This model results in a short response time. Once the client gets the required offer from the WSB, the client will invoke the service with the desired QoS properties.

The QoS properties are transmitted in the SOAP header to the service provider that can treat the request based on the QoS properties. For example, it could set the thread priority or, as a load balancer, forward it to one of various possible application servers. Yet, the information is not only intended for the Web service provider: A QoS proxy is able to interpret the desired transport QoS priorities and mark the outgoing packets accordingly so that the higher layer applications can take advantages of the QoS support that the underlying QoS-aware transport technologies provide. Furthermore, the information in the SOAP headers can be used to perform encryption or digital signatures.

Having processed the client's request, the service provider will send the response back to the client. The service provider has to ensure that it can carry out the client's requirements about the transport and security. The service provider should set the requirements in the SOAP header so that they can be evaluated and carried out by the participating components on the way back to the client. The components are e.g. the QoS proxy on the server side, QoS-aware routers in the network, or an access point for mobile devices.

The editor

The QoS Editor allows both the service client and the service provider to easily edit their QoS requirements or offers, respectively. They neither need to know the details of the QoS XML schema nor have any programming skill. One or more XML-based *.wsqos* files are generated automatically. The WSDL files are normally generated automatically by a tool such as wsdl.exe in case of the .NET runtime. In case of a service offer, one or more references of the *.wsqos* files are added manually into the WSDL file of the service. In case of a service request, the QoS Requirement Manager will retrieve the values defined in a *.wsqos* file.

In the GUI for defining custom QoS properties, one can define :

- the name of the requirement,
- the scope in which the requirements are valid, possible scopes are individual operation of a service or the whole service,
- the standard metrics of standard QoS aspects such as processing time, request per second, availability, and reliability as server QoS metrics,
- the price for the service usage the client is willing to pay or the service provider is going to charge, and
- custom metrics by applying ontology.

The requirement manager

On initialization, the QoS Requirement Manager obtains a reference to the service proxy class to which either requirement attributes have been assigned or a reference to a *.wsqos* file is given. The Requirement Manager retrieves the QoS attributes either from the proxy class or from a *.wsqos* file. It then collects all *import* attributes, builds WS-QoS definition objects and sets their parent property to receive update messages in case that a *.wsqos* file has been changed. Finally, the newly built WS-QoS definition objects are added to those retrieved from the static attributes.

The monitor

We have developed the QoS Monitor, which examines all available offers and the current client requirements, making it possible to check the compliance of offers. If no appropriate offer can be found, the overview of possible offers will help users to evaluate what requirements might be inappropriate and users could then make adjustments needed in order to find a match.

Moreover, the QoS Requirement Manager can be configured to log current requirements. Once this file is registered in the monitor, requirements can be viewed in the requirement watch window or directly in the offer window of the GUI. Finally, the package *QoS Util* provides a SOAP extension attribute, which can be assigned to the proxy's web methods. The SOAP extension will log QoS SOAP headers of all service requests and responses. One can register this file in the monitor as well use it to survey QoS SOAP headers in the SOAP header watch window of the GUI.

V. APPLYING FRAMEWORK

When implementing a WS-QoS compliant Web service three issues have to be taken into consideration: First, the service should implement a generic service interface (e.g. *tModel*), which already defines the use of an optional QoS SOAP header to transport QoS information within service requests and responses.

Second, the service has to implement a strategy to provide WS-QoS offers, which should be adjustable to changing situations of service utilization.

Finally, to achieve the QoS level(s) associated with distinct offers, the selected offer and further QoS requirements have to be evaluated when receiving a request. Apart from developing new services one can also qualify existing service implementations as QoS compliant by making just a few alterations to the code

To make the service available for dynamic look-up through the WSB, one should implement a specific *tModel*, which specifies a WS-QoS compliant Web service with the functionality of the service. The more common a *tModel* is, the more attractive it is for a client to search services implementing the *tModel*. Therefore we encourage reusing an abstract service description rather than inventing a redundant interface.

If no such *tModel* exists, one has to specify a new one. The easiest way to create a *tModel* is to implement a dummy service featuring a WS-QoS SOAP header as described below and call up the WSDL file for this service. Clients will be able to generate a proxy object from this interface and use the proxy for various services only by setting the proxy's access point URI to the location of a concrete implementation. In any case, concrete services should use the same XML namespace as used in the service interface.

The QoS level provided by a service is part of its nature. QoS offer definitions are therefore expected to be held in a QoS extension to the service's WSDL file. This allows for a fast discovery of WS-QoS offers, since a service's description file is commonly referenced from its entry in a UDDI registry along with further information such as the service access point. The WS-API provides several ways to publish QoS offers, depending on the personal strategies and requirements. It should be decided whether

- QoS offers are held directly in WSDL or a further WS-QoS file is referenced from the WSDL's WS-QoS extension,
- the validity period of QoS offers is updated manually or by the QoS Offer Manager Object provided by the QoS API,
- the service description referenced is a static WSDL file or XML generated by a server-side engine.

.

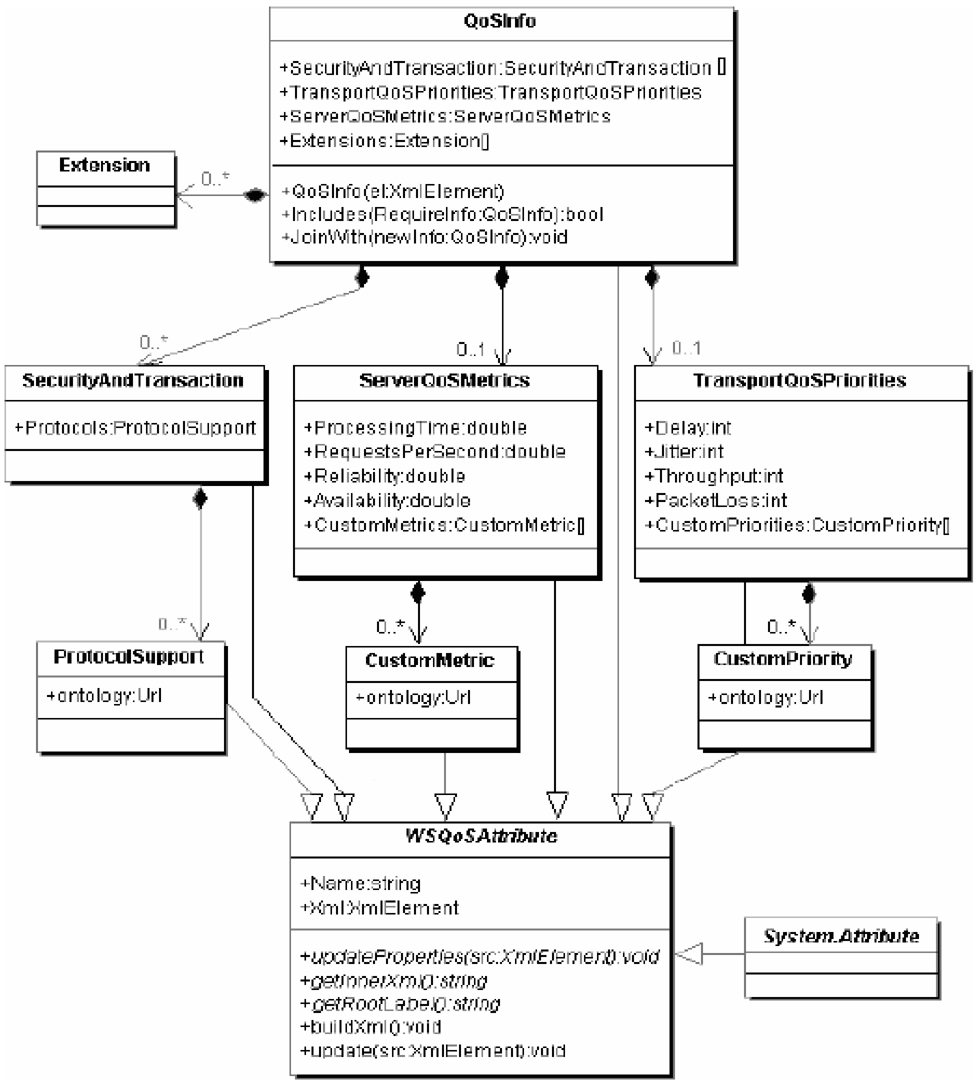


Figure 3. The Framework QoS API

Figure 3 gives an overview of the classes of the WS-QoS API representing elements of the QoS XML schema.

VI. CONCLUSIONS AND FURTHER DEVELOPMENTS

In order to analyse the state of the art for the proposed research domain, we introduced five different frameworks dealing with QoS and Web services.. One can easily identify that our framework addresses QoS-awareness during the whole Web service communication process, ranging from the definition of QoS requirements and offers to QoS-aware lookup, selection, and invocation. Reiterating the same analysis for the domain including our solution, we can consider the following :

Requirement specification: The flexible and extensible XML schema allows the specification of QoS requirements and aspects. The schema is extensible with custom metrics and aspects.

Class of service: Class of service is supported by the framework. QoS definitions can be grouped to classes such bronze, silver, and gold. They can be than assigned to either a whole Web service or each methods of the Web service. Most of the introduced approaches do not support class of service.

QoS aspects: our framework supports various QoS aspects ranging from traditional networks metrics to more high level metrics such as security, server performance. Each aspect encompasses of different *QoS parameters*. Both QoS aspects and parameters can be extended in a flexible way. Most of the other approaches just defined two or three simple QoS metrics without any extensibility.

Another important contribution of the framework is the *dynamic mapping* of high level QoS definitions onto different components such as communication network, and server hosting Web services. None of the other approaches supports QoS mapping.

We proposed a framework that ensures QoS-awareness during a whole Web services communication process, namely, the QoS-aware specification of Web services requirements and offers, the lookup and selection of Web services based on the specified QoS issues, and finally the QoS-aware Web service invocation at runtime.

We designed the QoS XML schema, which is core of the WS-QoS framework. All components such as clients, servers, routers, access points that participate in Web service communication, apply the schema in order to support QoS-awareness. The proposed schema is easy to use, standard conform, and fully extensible. We demonstrated these properties in our prototypic implementations and performance measurements.

We propose a Web Service Broker that is responsible for looking and selecting the most suitable Web service offer based on client requirements, which is defined by applying the QoS XML schema. We demonstrated the feasibility and performance of the WSB with our prototypic implementation and in various scenarios of the performance measurements.

We proposed that only an overall QoS support can ensure the fulfillment of clients' requirements due to QoS. It is not sufficient to consider QoS in each layer in terms of the Internet model separately. The different layers should communicate and cooperate with each other. Traditionally, only network metrics such as jitter and bandwidth are considered as QoS. In the realm of Web services, more QoS aspects should be taken into consideration to improve the total performance of Web service communication. Therefore, our QoS XML schema encompasses not only the traditional network QoS aspect, but also application and Web server, security, transaction, SLA, and pricing related aspects and parameters. One can augment the schema with custom aspects and parameters easily.

We introduced an Adaptation Layer between the Web services and the communication.The Adaptation Layer understands the QoS requirements for the underlying communication network and maps the high level requirements onto the concrete network technology at runtime and The is realized by the QoS proxies and ensures that the high-level definition of the network metrics can be specified in a technology independent way.

With the QoS xml schema we have created a platform-neutral infrastructure to describe client QoS requirements and service QoS offers. Our API provides easy access to our technology for developers; with a few lines of code they can add the functionality of the QoS aware service broker. With these two components we have realized QoS aware dynamic service selection. Future work will have to concentrate on defining appropriate custom parameters as well as investigating whether the standard parameters were chosen appropriately. Moreover, the implementation might well be revised for the purpose of performance optimization to keep the waiting time due to the enhanced service selection algorithm as short as possible.

There are several other extensions to the framework. One of them is the signature of dynamic offers. Service offers consisting of various QoS parameters and aspects are provided with a signature identifying offers for the valid time period. That results in that all actors such as WSB, clients and service providers need not send and process the various metrics, since the signature of an offer identifies the metrics belonging to an offer is unique.

Another interesting project would be the integration of the framework in a project dealing with a Web service execution plan. We think that our framework can be applied to Web service orchestration, when a chain of different services is to be found and selected among competing services at runtime.

ACKNOWLEDGEMENTS

"This work was supported by the PNII-IDEI 328/2007 QAF-Quality of Service Aware Frameworks for Networks and Middleware research project within the framework National Research, Development and Innovation Programme initiated by The National University Research Council Romania (CNCSIS - UEFISCSU)"

REFERENCES

[1]Martin-Diaz, O., Ruiz-Cortes, A., Corchuelo, R., and Toro,M., 2003, "A Framework for Classifying and ComparingWeb Services Procurement Platforms", Proc.of 1st Int'l Web Services Quality Workshop, Italy, pp. 37-46.
[2] Blum, A., 2004, "UDDI as an Extended Web ServicesRegistry: Versioning, quality of service, and more". Whitepaper, SOA World magazine, Vol. 4(6).
[3] Ran, S., 2004, "A Model for Web Services Discovery with QoS". ACM SIGEcom Exchanges, Vol. 4(1), pp.1–10.
[4] Majitha, S., Shaikhali, A., Rana, O. and Walker, D., 2004,"Reputation based semantic service Discovery", In Proc. Of the 13 th IEEE Intl Workshops on Enabling Technologies Infrastructures for collaborative Enterprises (WETICE), Modena, Italy, pp.297-302.
[5] Rajendran, T. and Balasubramanie, P., 2009, "An Efficient Framework for Agent-Based Quality Driven Web Services Discovery", IEEE International conference on Intelligent Agent and Multi Agent Systems (IAMA2009), Chennai.
[6] Keller, A. and Ludwing. H., 2002, "The WSLA framework: Specifying and Monitoring Service Level Agreements forWeb Services", IBM Research Report.
[7] ShaikAli, A., Rana, O.F., Al-Ali, R., and Walker, D.W., 2003, "UDDIe: An extended registry for web services". In Proc. Of the Symposium on Applications and the Internet workshops, IEEE CS, pp 85-89.
[8] Chen, Z., Liang-Tien, C., Silverajan, B. and Bu-Sung, L.,2003, "UX-An architecture providing QoS-aware and federated support for UDDI". In proc. of the Int'l Conf. on web services, CSREA Press, pp 171-176.
[9] Liu, Y., Ngu, A. and Zheng, L., 2004, "QoS Computation and Policing in Dynamic Web Service Selection", Proceedings of WWW 2004 Conf.
[10] Diego Zuquim Guimaraes Garcia and Maria Beatriz Felgarde Toledo, 2006, "A web service Architecture providingQoS Management", Institute of Computing, University of Campinas, Sao Paulo, Brazil, pp -189-198.
[11] Tian, M., Gramm, A., Ritter, H. and Schiller, J., 2004, "Efficient Selection and Monitoring of QoS aware Web Services with the WS-QoS Framework". Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'04) Exchanges, vol. 4, no. 1, pp. 1–10.
[12] Eyhab Al- Masri, and Qusay H. Mahmoud, 2007, "QoSbased Discovery and Ranking of Web services", Proceedings of IEEE International Conference.
[13] Ziqiang Xu, Patrick Martin, Wendy Powley and Farhana Zulkernine, 2007, "Reputation Enhanced QoS-based Web services Discovery", IEEE International Conference on Web Services (ICWS 2007).
[14] Demian Antony D' Mello, V.S.Ananthanarayana and Santhi.T, 2008, "A QoS Broker Based Architecture for Web Service Selection", Proceedings of IEEE International Conference.
[15] Demian. A. D'Mello and Ananthanarayana, V.S., 2008, "A QoS Model and Selection Mechanism for QoS-Aware Web Services", Proceedings of the International Conference on Data Management (ICDM 2008).
[16] Serhani, M.A., Dssouli, R., Hafid, A. and Sahraoui, H., 2005, "A QoS broker based architecture for efficient Web services selection". In Proc. of the IEEE Int'l Conf. on Web Services, IEEE CS, pages 113–120.