A Concept for QoS Integration in Web Services

M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller
Freie Universität Berlin, Institut für Informatik
Takustr. 9, D-14195 Berlin, Germany
{tian, gramm, naumowic, hritter, schiller}@inf.fu-berlin.de

Abstract

With the growing popularity of Web services, a general QoS support for Web services will play an important role for the success of this emerging technology. Unfortunately, current Web service environments do not offer comprehensive QoS support. In this paper, we present an approach that does not only enable the QoS integration in Web services, but also the selection of appropriate services based on QoS requirements regarding server and network performance. Furthermore, we present how application requirements regarding communication QoS are mapped onto the underlying QoS aware network at runtime, as well as how users can obtain real-time information about server performance in order to monitor the accomplishment of assured services, giving the user an instant QoS feedback.

1. Introduction

Today, research activities in applications, Web services, and communication networks are running in many aspects widely independent from each other. In most cases, researchers of applications and Web service technologies assume that existing communication infrastructures provide reliable communication. Furthermore, researchers in middleware, Web services, and applications are not very considerate of the resources provided by the underlying networks. On the other hand, research activities in certain communication architectures and protocols are performed with less attention to requirements of actual applications. Therefore, most applications cannot actively consume the Quality of Service (QoS) that may be supported in the communication networks, and on the other hand common network technologies do not support application-dependent requirements.

The demand on highly reliable and highly available Web services increases as more and more companies and customers rely on them to satisfy business and personal needs [1]. The growing variety of customers requires a diverse range of QoS support. The QoS a service provider delivers will become a decisive criterion when services with the same functionalities are available at customers' choice.

Nowadays, we have sophisticated technologies and research results regarding QoS support in different domains. They are for example DiffServ and IntServ for the network layer QoS support, demand-based QoS support through an adaptive end system [2], QoS aware middleware [3], service differentiation in overloaded servers [4]. Most recent efforts on QoS support in Web services are for example IBM's Web Services Level Agreement (WSLA) [5] and HP's Web Service Management Language (WSML) [6]. These two languages have been developed to specify Service Level Agreements for Web Services. Electronic contracts are negotiated individually and then surveyed by a monitoring engine. Service offerings defined in the Web Service Offerings Language (WSOL) [7] provide different predefined classes of service for clients to choose from.

However, most of these approaches neither support the mapping of QoS requirements from higher layers onto the underlying network layer in terms of the Internet model nor considerate the server performance. Figure 1 gives examples for parameters on different layers when mapping applications and services onto certain transmission technologies or when pushing performance parameters from transmission technologies up to applications, respectively. The communication and cooperation between different layers allows an efficient utilization of the underlying network resources as well as a better support of application-dependent requirements.

In this paper, we introduce our current effort tackling the gap between the Web service layer and network layer, as Figure 1 illustrates. We have been developing an architecture that allows the dynamic definition, publication, and matching of both Web service offers and requirements regarding server performance, network performance, security, transaction, pricing as well as customer defined issues at both implementation time and runtime. Our architecture supports the dynamic mapping of requirements regarding the network performance from higher layers onto the underlying network layer at runtime. We have defined a Web service QoS XML schema that both clients and service providers apply to define the QoS parameters, so that the QoS parameters are comparable. Furthermore, our architecture allows users to obtain real-time information about server performance in order to monitor the accomplishment of assured services, giving the user an instant QoS feedback. Our approach is extensible and based on Internet standards such as XML schema, SOAP, WSDL, and UDDI. This ensures the independence of any particular programming model and other implementation specific semantics.

It should be noted that our architecture does not by itself address the problems of routing, load balancing, security, transaction, and pricing. Instead, we concentrate on defining the WS XML schema, mechanisms for efficient selection of QoS-aware Web services, dynamic mapping at runtime, and instant QoS information delivery.

The goal of our prototypical implementation shown in this paper is to prove the feasibility of our concept. Strategies for parameter definition, selection of QoS-aware services, and pricing can be implemented individually. The study of these strategies is beyond the scope of this paper.

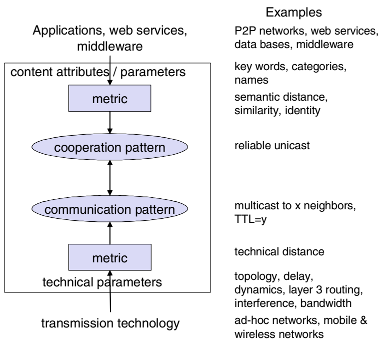

Examples
Applications, web services, middleware
P2P networks, web services, data bases, middleware
content attributes / parameters
key words, categories, names
metric
semantic distance, similarity, identity
cooperation pattern
reliable unicast
communication pattern
multicast to x neighbors, TTL=y
metric
technical distance
technical parameters
topology, delay, dynamics, layer 3 routing, interference, bandwidth
transmission technology
ad-hoc networks, mobile & wireless networks

Figure 1. Mapping of applications and services onto communication technology

The remainder of this paper is outlined as follows. After discussing some related work, we present the architecture of our QoS aware approach and discuss the specification issues in section 3. We conclude with an outlook of future work.

2. Related Work

To enable standardized QoS specification for Web Services, three sophisticated approaches for QoS specification within SLAs for Web Services have been developed simultaneously. Some QoS parameters are also considered in the course of process modelling.

IBM's Web Service Level Agreement (WSLA) framework [5] was designed to enable the specification and monitoring of the QoS with which a Web service is provided through electronic Service Level Agreements (SLA). While the form of an SLA specification is provided with the XML-based WSLA language [8] the aspect of monitoring the compliance with an associated SLA is implemented in the SLA Compliance Monitor which is part of IBM's Web Services Toolkit.

HP's Open View Internet Services product enables a business to manage services against service level agreements with service level violations being instantly reported. They describe a theoretic Web Service QoS parameter specification model and introduce Web Service SLAs in the form of the XML based Web Service Management Language (WSML) [6]. Automated SLA compliance monitoring has been realized with the Business Management Platform Agent. Furthermore, QoS aware service choice can be achieved through dynamic service ranking according to the different effects that the SLAs in question will have on a composite business process, which are simulated in HP's Business Process Simulation Environment on the basis of Service Level Information (SLI) provided by service providers.

While both WSLA and WSML foster a concept of individually negotiated customized SLAs, a research group from the Carleton University in Canada has developed the notion of providing various classes of service for one and the same functional service specification, which differ in QoS level and management efforts provided as well as a related prices. Distinct service offerings are formally described in the XML-based Web Service Offerings Language (WSOL) [7].

Distinguishing between different service levels will influence the strategies for the design of business processes that use Web Services service invocation. With this in mind, it is not surprising, that features related to QoS such as security, message reliability and transactional QoS are also issues in business process management (BPM) standards. So far, the most influential standards have been Microsoft's XLANG [9] and IBM's Web Services Flow Language [10]. As the most recent development the Business Process Execution Language for Web Services (BPEL4WS) [11] is designed to realize a convergence of the XLANG and WSFL to endorse further standardization with "a common model shared by leading organizations" [12]. High level QoS support might also be addressed during the work on the WS-Integration specifications.

All the concepts available today differ from our approach in that they do not use standard parameters. Furthermore, they neither support the mapping of QoS requirements from higher layers onto the underlying network layer in terms of the Internet model nor considerate the (Web service) server performance. An effective way of comparing offers for a dynamic selection has not yet been developed.

3. Web Service QoS Architecture

We propose QoS support in the Web service layer. By utilizing our system, service providers can augment their

Web service offers with QoS aspects while clients can define their requirements related to QoS parameters. QoS parameters such as processing time, request rate, response time, availability, reliability, security protocols, transaction, price, and customer defined parameters are declared for the Web service layer QoS support by clients and servers. Standard and customer defined parameters such as delay, bandwidth, jitter, and packet loss are defined for the network layer QoS support by both parties. It is not necessary to use the predefined parameters, one can define her own parameters by applying the WS-QoS Ontology (section 3.2.3).

Service providers can offer different classes of the same services. Of course, the different classes of services are charged differently. For example, the QoS could differ in three categories, platinum, gold, and bronze, each with different parameters as shown in Table 1.

Table 1. An example of different classes of a service

Class of service	Platinum	Gold	Bronze
Processing time	0,3ms	0,7ms	0,9ms
Throughput	200 requests/s	150 requests/s	100 requests/s
…	…	…	…
Price per service usage	0.05€	0.03€	0.01€

All the QoS parameters are defined in a standardized form based on our Web service QoS (WS-QoS) XML schema, which will be described in section 3.2.

3.1. Web Service Broker

We introduce a Web service broker (WSB) in order to accelerate the client lookup process for services, since a WSB can prefetch information about Web service offers a client could be interested in. That means a Web service client will contact the WSB for looking up a service instead of doing this with a UDDI registry. The WSB holds up-to-date information on offers currently available for a group of services. Offers are grouped by the interface (tModel) that the services providing them implement. The first time a service is requested and the WSB does not have up-to-date information about this service, one or more UDDI registries associated with the broker are inquired. The WSDL files for these services are then checked for WS-QoS extensions and available offers are built. From then on this newly created offer list is consulted to find the best match for clients.

To keep an up-to-date list of all services implementing a given interface (tModel), the UDDI registries are regularly checked. The available services are then regularly checked for new offers. Once an offer expires, it is deleted from the WSB's registry. If the validity of the offer is extended, it will be re-detected during the next check.

When a client application inquires the WSB for the cheapest available offer, it sends its QoS requirements as a part of the request. In the order of their price, the WSB then tests the available offers whether they fulfil the client's requirements. The first compliant offer is returned to the client. It is worth noting that one can implement her own strategy for defining the QoS parameters and the selection of the appropriate services. We just give here a simple idea of how the selection could be done.

There are two implementations of the WSB. One is a local object running within the application. This ensures a highly performing service selection and detailed information on available offers. The other implementation uses a remote Web Service to obtain the access point of the most appropriate service. This version is mainly intended as a (Web) service for multiple client applications that could use a single private service broker running within their network domain. This service broker could be used by any other WS-QoS compliant implementation, too.

Figure 2 depicts the participating roles service providers, clients, UDDI registries, and the Web service broker and their interactions. Note that there are no service brokers in the standard Web service model [13].

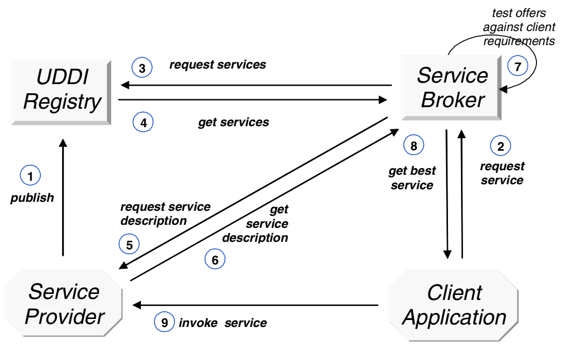


Figure 2. Interactions between the four participating roles

When the service broker does not have any information about a required service, the interactions between the roles are as follows. Note that we assume that service brokers normally analyze service offers in advance.

1. Service providers publish their Web services to UDDI registries. Web services available in UDDI registries are identified uniquely by an interface key (tModel).

2. Clients ask the WSB for services that implement a certain interface and accomplish the required QoS requirements.
3. If the WSB does not already hold up-to-date information on offers that accomplish clients' requirements, the WSB will request Web services according to the interface key from one or more UDDI registries. Note that we would prefer the model in which the WSB prefetches information of offers that clients could be interested in. This would accelerate the lookup phase significantly.
4. The UDDI registries return a list of services that implement the interface key.
5. The WSB asks the service providers for service descriptions, e.g. WSDL files.
6. The service providers return their service descriptions with QoS offers.
7. The WSB tests the offers against the clients' requirements.
8. The WSB returns the most appropriate service to the client.
9. The client directly invokes the service with the original QoS requirements. At this time, the QoS requirements regarding the network performance are actively mapped onto the underlying transport technology, as described in section 3.3.

Note that the WSB in step 7 tests the offers (step 6) against clients' QoS requirements sent in step 2.

3.2. The XML schema for Web service QoS definitions

For the QoS aware dynamic selection of Web services the QoS parameters defined by both service providers and clients must be compared by the WSB or any other instance. In our prototypic implementation, the WSB selects the cheapest service fulfilling the requirements from all available offers for services that implement the specified interface. Of course, the WSB could implement any other decision strategy. In order to standardize the QoS specification for efficient comparison, we have designed a Web service-QoS XML schema. The standardization through the XML schema is essential since it allows an easy lookup and selection of services.

Both QoS requirements and QoS offers are specified by elements of the type *tQoSDefinition*. While only one requirement element is allowed as a root element, multiple offers may be held in a collection. The collection may also contain references to WS-QoS files to include further offers. 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.

Next to standard parameters, custom parameters can be declared, referring to a public WS-QoS ontology. Therefore, an element *WSQoSOntology* has been designed to hold definitions of QoS parameters and protocol references.

In the following subsections, we will describe the QoS definition and components participating in this process, how offers and requirements are matched, the mapping of the QoS requirements onto the QoS aware network as well as how service providers deliver real-time information about the server performance to the user.

3.2.1. QoS Definition

Elements of the type tQoSDefinition are either instantiated as an element *requirements* expressing a client's QoS requirements or as a *qosOffer* representing a minimal QoS level a service provider guarantees to provide. The *qosOffer* element is extended by an attribute *expires* which denotes a point in time until which the offer will be valid.

Figure 3 shows the type *tQoSDefinition*. An element of this type holds one or more elements of the type *tQoSInfo*. These can be defined for the scope of an individual operation in an *operationQoSInfo* element or for the whole service in a *defaultQoSInfo* element.

In its *contractAndMonitoring* node, a node of the type *tQoSDefinition* provides references to protocols needed for service management and QoS monitoring as well as entries of third parties that one side would be willing to trust. The *price* element relates the specified QoS level to the cost of service usage per invocation. Finally, an extension element allows future extensions to the schema.

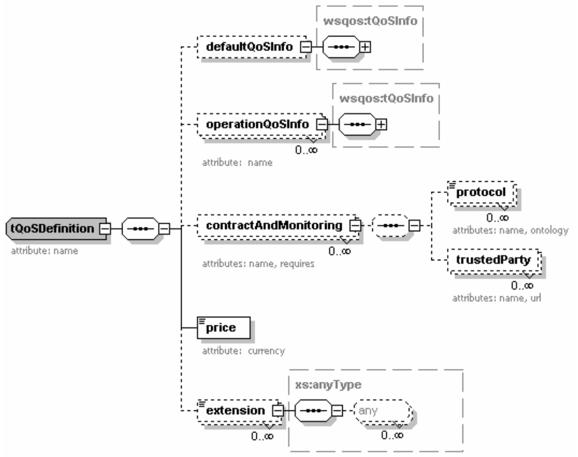


Figure 3. Structure of the type tQoSDefinition

3.2.2. QoS Info

The most important of all elements are those of the type *tQoSInfo* as depicted in Figure 4. It holds 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 for the Web service server performance as well as customer defined server QoS metrics.

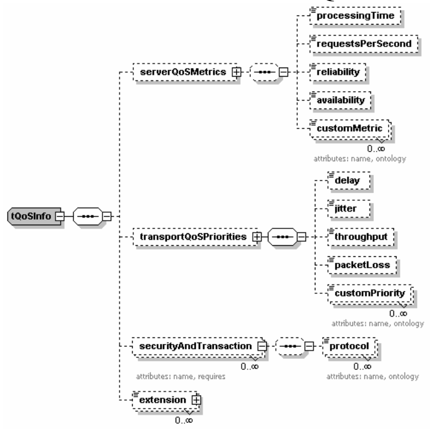


Figure 4. Structure of the type tQoSInfo

A *transportQoSPriorities* element specifies priorities for the four standard transport QoS parameters delay, jitter, throughput, and packet loss rate and optionally customer defined transport QoS priorities.

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 listing protocols needed for a successful service execution in a *securityAndTransaction* element. As for a QoS definition, extensibility is supported.

3.2.3. WS-QoS Ontology

Custom defined metrics, priorities, and protocol support statements all have an attribute *ontology*, which references a file containing a WS-QoS Ontology where the referenced types are defined. By using the combination of the ontology's URL and the parameter name, a reference is unique. A customer defined 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.

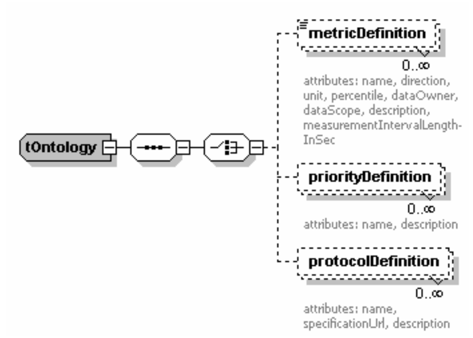

tOntology
metricDefinition
0..∞
attributes: name, direction, unit, percentile, dataOwner, dataScope, description, measurementIntervalLength-InSec
priorityDefinition
0..∞
attributes: name, description
protocolDefinition
0..∞
attributes: name, specificationUrl, description

Figure 5. Structure of a WS-QoS Ontology

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 declares a standard unit and the direction of how values are to be compared.

Accordingly, in a *protocolDefinition* element, a protocol is defined by its name, a human readable description about the reasons for using this protocol and the URL of an overview document of the protocol specification if available.

Figure 5 shows the structure of a WS-QoS Ontology element.

3.3. Network Layer QoS Support

In the previous section, we have introduced our approach that allows both the client and service providers to define QoS requirements and offers based on the WS-QoS schema. A Web service broker helps the clients to find the appropriate offers.

As shown in section 3.2.1 and 3.2.2, client applications can not only define QoS requirements concerning the server performance, transaction, security, and pricing, but also the network performance. As Figure 1 depicts, it is essential that the underlying QoS enabled transmission technology takes the application requirements into account in order to achieve an overall performance gain. The WS-QoS XML schema provides a generic mechanism for an application to specify such requirements. Application designers and programmers need not have any knowledge about the underlying transport technology at the design or implementation time, since the mapping takes place at runtime.

In order to control and set the requirements of the client application concerning the network performance, we have to deal with the network streams exchanged between the client application and the remote Web service provider. Note that we assume that the underlying transport technology supports QoS such as DiffServ, ATM, or UMTS. Since different QoS enabling technologies have different QoS mechanisms the QoS parameters for transport are declared as priorities, rather than absolute values. An instance located between the Web service layer and the transport layer evaluates the transport QoS parameters and maps them onto the underlying technologies meaningfully. We call such an instance QoS proxy.

On the client side, a QoS proxy resides between the Web service client and the network interface. The QoS proxy observes the traffic on a specific port, through which the Web service client sends its requests to the server. The QoS proxy maps the client's requirements onto the current QoS-aware network after detecting the transport QoS parameters set by the client application.

On the server side, a QoS proxy or any other instance such as a traffic shaper or a load balancer is located between the network interface and the Web service. It sets the QoS parameters according to the client requirements onto the underlying transport technology when the Web service provider sends responses to the service client.

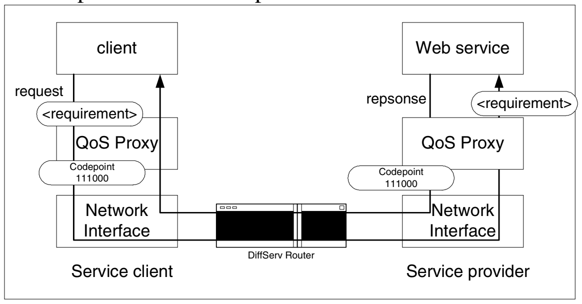


Figure 6. QoS Proxies map client's requirements onto the underlying transport technology

Figure 6 depicts the participating components and the data flows during the interaction between a Web service client and the service provider at runtime. In this case, we assume that the QoS-aware network is a DiffServ network. The QoS information regarding the network performance specified by the client is placed in the SOAP header, which will be parsed by the QoS proxies on both the client and server side. Based on the QoS information, the QoS proxies mark the DiffServ specific DiffServ code points (DSCP) in the IP packets. DiffServ routers in the network will treat the traffic between clients and server depending on the DSCP. For simplicity, we only show the interaction between the Web service client and provider, ignoring the UDDI registries and the WSB, which are Web services, too.

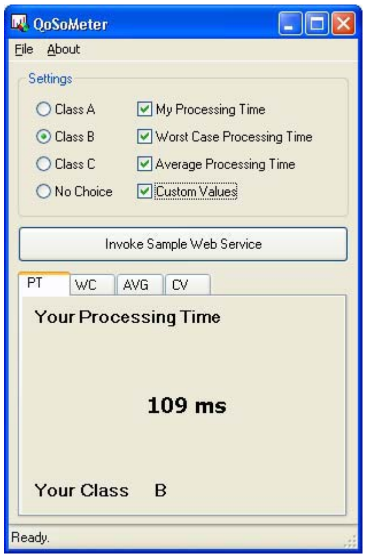
3.4. Server Performance Observation

Our architecture informs users about the current server performance in real-time. We introduce a QoS channel between the server and the client. The QoS channel is realized by placing information into the SOAP header. The user defines what QoS information regarding the server performance she is interested in. The server delivers the required information to the client by applying the QoS channel. The client knows the service time, which is defined as the time interval between the moment the client requests the service and the moment the client receives the response. The server provides information about its performance such as the processing time of the current request, worse case and average values for processing requests of different classes, and customer defined values.

The processing time denotes the time interval a Web service needs to process a service request. The time a request spent for queuing at the Web service host computer is not included. If the client knew both the queuing and processing time of its request it would be able to derive the network performance from this information, since it knows the service time. Mechanisms for determining client perceived response time is discussed in [14]. Note that we use the QoS channel to transport the server performance, for now. But one could apply the generic QoS channel to transport further information such as packets queuing time.

A graphical user interface (GUI) on the client side shows the server and network performance. The usage of the GUI is fully flexible. The user can switch off the GUI completely; she can choose QoS parameters she is interested in from the GUI. She can be alerted instantly in case of server underperformance. Since she can request statistics about the server performance of other classes of the same service, she can get a feeling what would happen if she paid for a better or worse class of the same service as she does.

Figure 7 shows a sample GUI that allows the user to select a service class, in this case class A, B, C, or no choice, and information about the server performance such as the processing time of the current request, and statistics about worse case and average processing time. The user can define custom values as well. The corresponding Web service treats the client requests differently depending on the client class. The tab pane "AVG" shows the average server performance in all three classes.



Settings

Class A [✓ Class B] Class C No Choice

[✓] My Processing Time
[✓] Worst Case Processing Time
[✓] Average Processing Time
[✓] Custom Values

Your Processing Time

109 ms

Your Class B

Figure 7. A GUI informs the user about the server performance instantly

4. Conclusions and Future Work

In this paper, we have introduced our current effort on QoS support in Web services and the dynamic mapping of requirements from Web service layer onto the underlying QoS aware network layer. Our approach allows the dynamic selection of Web services depending on various QoS requirements. The QoS definition regarding network performance can be stated independently of the underlying network. Its mapping onto the current transmission technology takes place at runtime. Our approach allows the user to receive instant information about the server performance.

We have built a testbed in order to conduct performance measurements of our architecture. We are interested, for example, in the performance of the WSB for selecting the most appropriate service in comparison to the standard lookup model. Another interesting issue is to extend our architecture with support for mobile clients.

5. References

[1] D. Menasce and V. Almeida, *Capacity Planning for Web Services*, Prentice Hall, ISBN 0130659037, 2002.

[2] M. Bechler, H. Ritter, J. Schiller, "Quality of Service in Mobile and Wireless Networks: The Need for Proactive and Adaptive Applications", *33rd Hawaii International Conference on System Sciences-Volume 8*, http://www.computer.org/proceedings/hicss/0493/04938/04938026abs.htm, Maui, Hawaii, January 04 - 07, 2000.

[3] K. Nahrstedt, et al., "QoS-aware middleware for ubiquitous and heterogeneous environments", *IEEE Communications Magazine*, http://cairo.cs.uiuc.edu/publications/paper-files/IEEEComm01.ps, Nov. 2001.

[4] T. Voigt, R. Tewari, D. Freimuth and A. Mehra. "Kernel Mechanisms for Service Differentiation in Overloaded Web Servers", *2001 Usenix Annual Technical Conference*, Boston, MA, USA, http://www.sics.se/~thiemo/usenix01.ps, June 2001.

[5] A. Dan, A. R. Franck, A. Keller, R. King, H. Ludwig (IBM), „Web Service Level Agreement (WSLA) Language Specification", http://dwdemos.alphaworks.ibm.com/wstk/common/wstkdoc/services/utilities/wslaauthoring/WebServiceLevelAgreementLanguage.html, 2002.

[6] A. Sahai, V. Machiraju, M. Sayal, L. Jie Jin, F. Casati (HP), "Automated SLA Monitoring for Web Services", http://www.hpl.hp.com/techreports/2002/HPL-2002-191.pdf, 2002.

[7] V. Tosic, B. Pagurek, K. Patel, "WSOL – A Language for the Formal Specification of Classes of Service for Web Services", *Research Report OCIECE*, http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRep03-03.pdf, Feb. 2003.

[8] A. Keller, H. Ludwig (IBM), "The WSLA Framework: Specifying and Monitoring of Service Level Agreements for Web Services", *IBM research report RC22456*, http://www.research.ibm.com/resources/paper_search.shtml, 2002.

[9] S.Thatte (Microsoft), "XLANG - Web Services for Business Process Design", http://www.gotdotnet.com/team/xml_wsspecs/xl ang-c/, 2001.

[10] F. Leymann (IBM), "Web Services Flow Language (WSFL 1.0)", http://ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.

[11] IBM, Microsoft, Bea, "SAP Business Process Execution Language for Web Services 1.1", http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, May 2003.

[12] BPMI.org BPML / BPEL4WS, "A Convergence Path toward a Standard BPM Stack", http://www.bpmi.org/downloads/BPML-BPEL4WS.pdf, August 2002.

[13] Heather Kreger, IBM Software Group, "Web Service Conceptual Architecture (WSCA 1.0)", http://dwdemos.dfw.ibm.com/wstk/common/wstkdoc/ettk/wstk/doc/WebServicesArchitecture.pdf

[14] D. Olshefski, J. Nieh, and D. Agrawal, "Inferring Client Response Time at the Web Server", *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2002)* Marina del Rey, CA, http://parapet.ee.princeton.edu/~sigm2002/papers/p160-olshefski.pdf, June 15-19, 2002, pp. 160-171.