RC23092 (W0401-142) January 29, 2004
Computer Science

IBM Research Report

Web Services QoS: External SLAs and Internal Policies Or: How Do We Deliver What We Promise?

Heiko Ludwig
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Web Services QoS: External SLAs and Internal Policies
Or: How do we deliver what we promise?

Heiko Ludwig
IBM T.J. Watson Research Center
hludwig@us.ibm.com

Abstract

With Web services starting to be deployed within organizations and being offered as paid services across organizational boundaries, quality of service (QoS) has become one of the key issues to be addressed by providers and clients. While methods to describe and advertise QoS properties have been developed, the main outstanding issue remains how to implement a service that lives up to promised QoS properties. This keynote speech revisits the current state of the art of QoS management applied today to Web services and raises a set of research issues that originate in the virtualization aspect of services and are specific to QoS management in a services environment – beyond what is addressed so far by work in the areas of distributed systems and performance management.

1. Introduction

Whether offered within an organization or as a part of a paid service across organizational boundaries, quality-of-service (QoS) aspects of services are important in a service-oriented computing environment. Dealing with QoS is a sign of a technology going beyond its stage of initial experimentation to a production deployment and many recent activities related to QoS of Web services indicate that this is becoming an increasingly relevant topic.

Efforts in the past years mainly focused on describing, advertising and signing up to Web and Grid services at defined QoS levels. This includes HP's Web Services Management Framework (WSMF) [1], IBM's Web Service Level Agreement (WSLA) language [2,3], the Web Services Offer Language (WSOL) [4] as well as approaches based on WS-Policy [5]. These efforts enable us to describe quality metrics of services, such as response time, and the associated service level objectives flexibly and in a way that is meaningful for the business needs of a service client.

However, one of the challenging issues is to associate or derive a system configuration that delivers the QoS of a Web service described using the abovementioned approaches. In many cases this is non-trivial. Sometimes we can rely on experience with tested, dedicated system configurations to decide, for example, the size of a cluster for a particular workload guaranteeing a particular response time for a given percentile of requests. In addition, managing a service at different QoS levels on the same infrastructure is not easy.

While managing QoS in distributed systems is not a novel problem, a number of additional issues arise in the context of a service-oriented computing environment. Those issues arise from the specific properties of Web services. For example, cross-organizational Web services may be accessed through the public Internet and client side QoS metrics have to include network properties in addition to properties of the service-implementing application itself. In addition, composite and recursively aggregated services – and the ability to aggregate is seen as a key benefit of Web services – gain new QoS properties that are not always easily derived from its parts.

The objective of this keynote is to revisit the state of the art in managing QoS in distributed systems and to raise the new issues that we face in an environment of Service-oriented Architectures (SOAs). First, we revisit and discuss further open issues related to QoS management of Web services in general. Subsequently, we investigate how existing approach from the area of distributed systems and performance management help address some of the QoS management issues. Finally, we elaborate on the outstanding issues that are not solved so far and potential directions to address them in further research by the Web services community.

2. Issues of Web services QoS

The first step to manage Web services quality is to **define** it. While this is important for Web services as well as in traditional distributed systems, explicit definition is particularly important in an environment transcending organizational boundaries. Quality is expressed referring to observable parameters relating to a non-functional property, for example the response time of a request. A level of quality is agreed upon as a constraint over those parameters, potentially dependent on a precondition. Hence, the party offering a Web service, in agreement with its customers and users, will define the QoS parameters and the particular instances of the service to which these parameters relate. In the case of a Web

service, a parameter such as response time can relate to an individual invocation of an operation or a class of operations all having the same (individual) quality properties of having an aggregate property, e.g., the average response time of this class of operations or another stochastic metric.

A further step in managing Web services QoS is the definition of the **semantics** of the QoS parameters. A Web service and its subscribers and users must understand what is meant. It is important what is measured where. For performance-oriented metrics this can be at different points, as figure 1 illustrates.

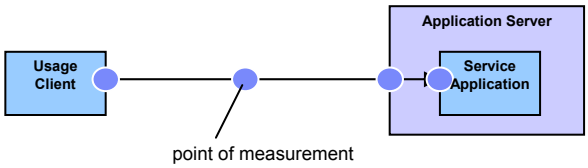

Figure 1: Points of measurements defining semantics of metrics.

Response time measurements taken from the Web service client are entirely different from those taken on the network between client and service, the application server inbound queue of a service provider or the service application itself. Hence, even a seemingly simple metric such as response time needs further qualification. Furthermore, aggregate properties must be defined, e.g., average response time. In this case, it is important to understand the averaging window, possibly a sampling rate and other aspects of the aggregation.

The definition of QoS parameters corresponds to the establishment of an ontology between a service provider and its clients. An ontology can be established in two approaches. (1) It can be a definition of terms and, potentially, the semantics of the relationships between them, as facilitated by DAML and OIL [6]. This approach results in a fixed set of well understood terms – in our case the QoS parameters. (2) Another approach uses constructive ontologies. Based on a set of well-know defined terms (as in 1) and a set of well-know composition operators, new terms (QoS) parameters can be defined by composing new parameters out of existing ones using the operators. For example, an average response time can be defined by applying the *average* operator to the *selection* of response times of an operation in the past minute. In this example, response time is a well-defined parameter and average and selection are composition operators. This is an approach that is proposed by WSLA. While it is easier to implement applications that can deal with a fixed set of terms that are all known at the time of system implementation, constructive ontologies provide more flexibility.

Having established common understanding of quality of service parameters and the associated guarantees given by the provider, it also has to be established to which relationships between a client and a server a QoS guarantee applies. A service may provide the same quality to all requesting clients, to each client individually or to a defined set of clients that a provider organization and an organization requiring a QoS level for multiple clients agree upon in a contract, which is also called an SLA.

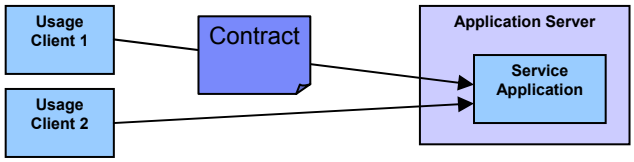

Figure 2: Contracts defining the scope of quality guarantees.

Clients will refer to the contract when requesting service according to a particular quality level.

The different scoping approaches of QoS guarantees require different means of establishing a particular quality level for a client: If a QoS level is associated with a service a client searches for a suitable service in a directory, e.g., UDDI and retrieves its quality definition, e.g., stated as a WS-Policy expression. In the case of an individual client or a contract, a negotiation mechanism, which can be very simple - must be provided. Once the contract is established, the provider organization must provision a service-implementing system such that it behaves as it has been agreed upon. This involves deriving the amount of resources needed and the runtime management of resources.

However, this is not simple. While we have developed – improvable – approaches to the issues raised above, the issue of provisioning and runtime managing a Web service-implementing system is not equally well understood yet. In the next section, we discuss what distributed systems and performance management approaches can provide.

3. Implementing service quality – current approaches

Different QoS parameters require different types of resources and different approaches to runtime management. Performance – or response time – management deals with shared or dedicated resources to be allocated to a particular scope of quality. It is based on the model that multiple workloads compete for resource, each workload having particular characteristics. Availability management is based on a model of failure of resources and allocates shared or dedicated redundancies. The management of those parameters has been subject of research outside the scope of Web services. There are, of

course, many more parameters such as *time to recover*, etc.

A number of performance management technologies, such as workload managers and network dispatchers, have been developed to control response times of individual systems and clusters and various availability management approaches. However, it is not straightforward to configure, for example, workload managers to satisfy response time goals for a set of different scopes of quality – for Web services as well as for any distributed system. In this section, we outline some typical approaches how QoS parameters are managed in distributed systems today. Since performance management is typically the first QoS aspect that needs to be addressed, we focus on this parameter in the subsequent discussion.

3.1 Allocating dedicated resources

An approach to manage performance of workloads that is frequently used is the allocation of dedicated resources to the different workload. A number of *servers* running a service application are the resources to be allocated. Usage clients send messages to access services. A *dispatcher* receives those messages from clients and assigns the requests to servers to be processed. Multiple algorithms can be used for the dispatching of requests for the same service, from simple round robin to algorithms that take into account the current workload on the servers and the expected time to start processing a new call. The dispatcher measures performance QoS parameters such as response time and makes those values available to a *provisioning manager*. According to the performance goal of the service, the provisioning manager adds new servers to the server pool of a particular service in case of goal underachievement or withdraws servers in case of overachievement. Adding a server to a pool typically involves the installation and configuration of the service application and the reconfiguration of the dispatcher, which involves costs in terms of setup time. Provisioning managers use information on provisioning cost, moving averages of workloads and sometimes sophisticated models of future behavior of clients to make their decisions [7].

In a typical environment, however, a service provider would offer multiple services or qualities. The illustration in figure 3 outlines a typical multi-service configuration. While the main components stay the same, the provisioning manager has to decide to which server pool for a particular service a given server is assigned. In addition, if multiple services cannot meet their performance goals, it must arbitrate which service is given preference and receives additional resources and from which service they are taken. To be able to arbitrate, the provisioning manager must have a utility function describing the value of reaching or missing each services performance objectives to be able to compute the marginal benefits of allocating a server to one service or another.

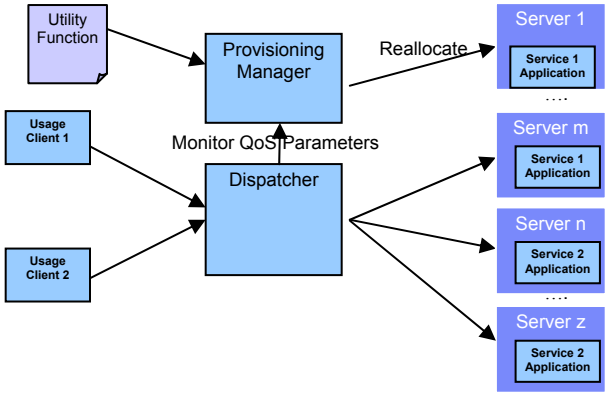

Figure 3: Allocating dedicated resources.

Using this approach of dedicated resources a service provider can also offer the semantically same service at different quality of service levels by making them technically separate services. Hence, a service provider can allocate separate servers for each customer requesting a new performance level. The dispatcher is relatively simple and a different dispatcher can be used for each individual service.

3.2 Service differentiation on shared resources

In some cases, using dedicated resources and offering different performance levels as separate services neither uses resources well nor provides a flexible and general interface to a service from a client's point of view. Hence, a more sophisticated dispatcher is used such as the one proposed in Levy et al [8].

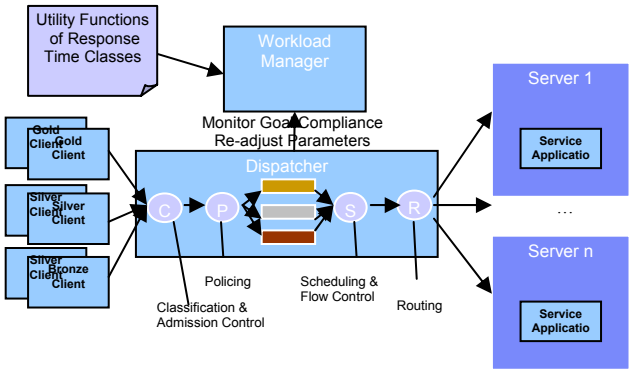

Figure 4: Managing shared resources.

Figure 4 illustrates a configuration of a more sophisticated dispatcher for shared resources. A key approach to reduce the number of server pools for different performance levels is the establishment of a – small – number of performance classes to which

individual performance guarantees can be mapped. For example, one could establish a class of requests responded to in less than one second, less than 5 seconds, less than 10 seconds, and best effort. A performance guarantee for a client of 6 seconds could then be mapped to the less-than-5-second class. This results in a lower number of server pools to deal with, even if individual servers are not shared between different types of services. Typically, however, each server would have multiple service application installed to increase flexibility. Those classes of service are often private to the service provider and not disclosed to its customers. In addition, all clients use the same dispatcher for all requests, labeling the request with the particular scope of quality they require, for example in the SOAP header of a Web service request.

Given a mix of client requests as input and a set of flexible and shared resources, the dispatcher processes requests in multiple steps: First, it is verified that an incoming message is allowed at a requested level of performance (*admission control*) and it is *classified* to the internal classes of service. Subsequently, the request is put into a queue associated with its class of service. Each queue has a given length to ensure that the performance guarantee associated with its class of service can be met. The *policing* function checks the queue status before queuing the request. Then, a *scheduler* and *flow controller* takes requests from the class of service queues. There are multiple algorithms to take requests from queue. A frequently used approach is weighted round robin. Each queue is given a weight, e.g., 5 (best), 2 (medium) and 1 (worst). When server capacity becomes available, the scheduler takes 5 requests from the best queue, 2 from the medium and 1 from the worst, and then starts over at the best. When no request is waiting in a queue, it is skipped. Finally, the *router* puts the next request on a server to be processed. This function is equivalent to the dispatcher function in the case of dedicated resources as discussed above.

The class-of-service weights of the dispatcher are determined by a *workload manager* that supervises the dispatcher. Driven by a utility function describing the benefits of the classes of service, the workload manager monitors the goal compliance of the classes of services and readjusts the weights of the corresponding queues. For example, if not many "best" requests are arriving at a given moment and performance goals are exceeded but "medium" goals are not met, the medium request queue will be given more weight at the expense of the "best" queue. An example of a more complex workload manager for is WLM [9].

Using this approach of shared resources, resource usage is significantly higher for services being offered at different performance levels at the expense of a more sophisticated dispatcher, which could become a bottle neck, and a classification of performance levels that can lead to over-serving many requests.

3.3 Combining approaches

To avoid a dispatcher bottle neck and to deal with different services and large numbers of performance classes on those different services, those above-mentioned approaches are often combined, as outlined in figure 5.

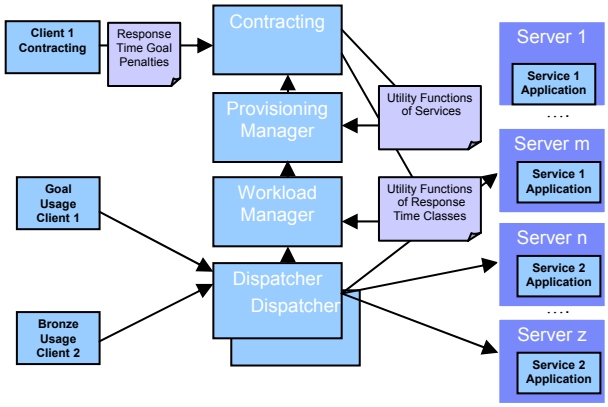

Figure 5: Combining approaches for comprehensive response time management.

The server pool is divided by the provisioning manager into servers associated with a particular service, to a set of operations of a service, or further into servers dedicated to one or more classes of service. As discussed above, the provisioning manager uses a utility function for the arbitration between its segments. The requests to each of the segments of servers are driven by a separate dispatcher. A dispatcher is associated to a workload manager that periodically reevaluates the parameters of its scheduling algorithm, e.g., the queue weights.

This raises the issue where the utility functions come from. If the service provider offers differentiated services, a *contracting* function allows prospective customers to subscribe to a service and negotiate the QoS terms as well as the pricing and penalty conditions. Based on priding and penalty, the contracting function derives the corresponding utility functions for provisioning and workload manager. Depending on the flexibility offered to customers in the subscription process, the derivation of the utility functions is often not an automated process but may involve considerations of a system administrator of the service provider that is familiar with the cost functions of the server cluster [10].

3.4 Implementing other QoS aspects

In addition to performance management, another main QoS parameter that is important to service providers and

clients is *availability*. All major vendors of systems management software such as IBM/Tivoli, Hewlett-Packard and Hitachi address this issue. In the context of Web services, a service is not available if it does not yield response in a defined amount of time, which is usually much higher than a response time guarantee. It is being addressed by making redundant resources available, which includes servers as well as, potentially, network connections. Again, redundant resources can be dedicated to a particular scope of QoS or can be shared between multiple services. A utility function drives the arbitration in case of need of redundant resources.

Other QoS such as time to recover have been equally addressed in theory and as available systems.

4. Web services-specific issues

As discussed in section 3, systems management approaches provide many solutions that can be adapted to a service-oriented architecture. For example, the dispatcher approach to manage shared resources can be used by using an entry in the SOAP header of a message as a label for a requested QoS level. However, in the field of service-oriented computing and Web services, we face a number of additional issues. Those additional issues regarding the implementation of quality of service mostly originate in the abstract concept of a service and its – intentionally – weak association with actual computing and networking resources.

4.1 Taking networking into account

Clients want to define QoS parameters from their perspective. While this is not so much an issue in traditional distributed systems, where the network and other system environment components are under control of the same organization, it has to be taken into account by providers and users of Web services.

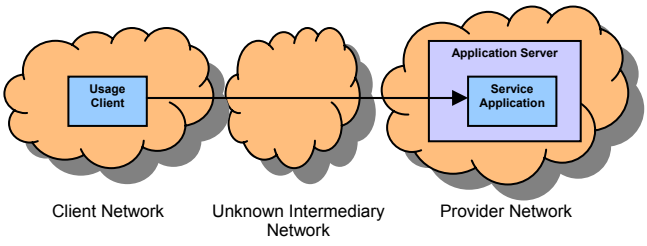

Figure 6: Client access over public networks.

Many non-corporate network users and small and medium-sized businesses don't have QoS guarantees regarding their access to the Internet. However, it is important for them to obtain quality of service guarantees as they perceive a service. Innovative solution are requested to be able to bundle QoS properties of a service application with the properties of the provider's network, the client's network and the network connecting both.

4.2 Composite web services

Services are often to be included in composite Web services defined using, for example, BPEL4WS [11] or other description languages. Using those languages and corresponding workflow management systems, services of different providers can be integrated. Figure 7 outlines a simple example of a stock trade workflow.

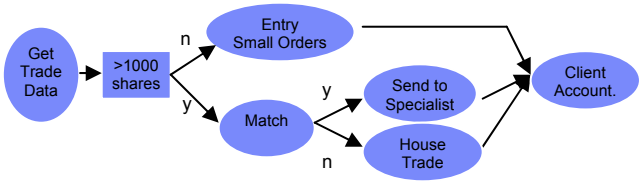

Figure 7: Contingent control flow of services.

In this case of a process composed of services, we have to understand how the individual QoS properties of one element of a composite service contribute to the overall QoS of the process. This is particularly interesting in the case of stochastic QoS: In 90% of the cases, the average response time will be less than 2 seconds. In addition to the likelihood of different alternatives on the process graph, we have to take into account the QoS properties of individual services. In the case of processing time, for example, the variance of service processing times of individual steps is increased by the variance of the overall process such that a it is difficult to give good aggregate QoS parameters for the whole process.

Approaches to address this problem may start with defining QoS properties of services when modeling a process and binding to particular services [12]. In addition, composite services may specify QoS properties dependent on the input data and the expected path of the execution. In addition, alternative services may be invoked at the same time for a particular step, the best – fastest – one used and the others aborted. However, this area requires more research and novel ideas.

4.3 Recursively composed services

Finally, composite services as outlined above can be offered as web services, thereby creating recursive service relationships. The figure below shows a multi-level service aggregation of our previous example.

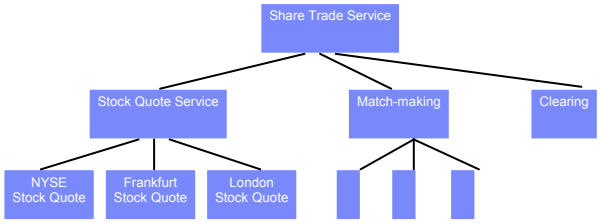

Figure 8: Recursive service composition.

With an increasing number of aggregation steps, stochastic QoS properties get very broad and meaningless very quickly. How can we find meaningful QoS properties for those aggregates and what are the limits of aggregation from a QoS point of view? In addition, how can a service provider of an aggregate service ensure a reasonable level of quality to its clients, particularly with respect to response time and availability?

A promising approach lies in using the dynamic properties of public Web services in that can be purchased on the spot from – potentially – multiple providers. Services can be bought from different providers and only those will be used in a given invocation that respond fast. For this approach, we need mechanisms that invoke multiple services in parallel and use the results of the fastest. In addition, we need the corresponding business models for Web services that only pay (in full) for a service if it was faster than others.

With respect to availability management, service providers can buy redundant services. However, they must make sure that they, in turn, do not depend on the same underlying service, which would annul their redundancy, at least partially. We need methods to describe dependencies on other services and match-making approaches that take those dependencies into account.

5. Conclusions and call for future work

In this keynote we discuss open research issues related to the QoS management of Web services. While the representation of Web services QoS properties has been addressed in current work, the implementation of QoS relies mainly on existing work related to allocating dedicated resources and managing the workload of shared resources. However, due to the virtualization of services and the trend to compose services to complex aggregates and to re-offer them, new approaches are needed to be able to implement the QoS of those complex services on a level required by business customers. More research is needed to apply the flexibility offered be Web services to solve the problems raised by this flexibility.

References

[1] N. Catania, P. Kumar, B. Murray, H. Pourhedari, W. Vambenepe, K. Wurster, Web Services Management Framework, Version 2.0, Hewlett-Packard, http://devresource.hp.com/drc/specifications/wsmf/WSMF-WSM.jsp, July 16, 2003.

[2] H. Ludwig, A. Keller, A. Dan, R. King, R. Franck, "A service level agreement language for dynamic electronic services", Electronic Commerce Research, Kluwer Academic Publisher, 3:2003, pp. 43-59.

[3] H. Ludwig, A. Keller, A. Dan, R. King, R. Franck, Web Service Level Agreement (WSLA) Language Specification, Version 1.0, IBM Corporation, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, January 28, 2003.

[4] V. Tosic, B. Pagurek, K. Patel, "WSOL – A Language for the Formal Specification of Classes of Service for Web Services". Proc. of ICWS'03 (The 2003 International Conference on Web Services), Las Vegas, USA, June 23-26, 2003, CSREA Press, pp. 375-381.

[5] D. Box, F. Curbera, M. Hondo, C. Kale, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, J. Shewchuk, Web Services Policy Framework (WSPolicy), http://www.ibm.com/developer-works/library/ws-policy, May 28, 2003.

[6] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, DAML+OIL (March 2001) Reference Description, W3C, http://www.w3.org/TR/daml+oil-reference, December 18, 2001.

[7] C. Crawford, A. Dan, "eModel: Addressing the Need for a Flexible Modeling Framework in Autonomic Computing", IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2002).

[8] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. N. Tantawi, A. Youssef: "Performance Management for Cluster Based Web Services", Integrated Network Management VII, Managing It All, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), IFIP Conference Proceedings 246, Kluwer Academic Publisher, 2003, pp. 247-261.

[9] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive algorithms for managing a distributed data processing workload", IBM Systems Journal, Volume 36, 2, 1997, pp. 242-283.

[10] H. Ludwig, "Electronic Contracts", Technology Supporting Business Solutions: Advances in Computation: Theory and Practice, Nova Science Publishers, 2003, pp. 3-28.

[11] T. Andrews, F. Curbera, H. Dholakia, Y Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Business Process Execution Language for Web Services Version 1.1,

http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 05 May 2003.

[12] E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, P. Devanbu, "GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions", Proceedings 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, Scotland, UK, 2004.