Evaluating Quality Provisioning Levels in Service Oriented Business Environments

Konstantinos Tserpes, Dimosthenis Kyriazis, Andreas Menychtas, Antonios Litke, Costis Christogiannis and Theodora Varvarigou
Dept. of Electrical and Computer Engineering, National Technical University of Athens, 9, Heroon Polytechniou Str, 15773 Athens, Greece
{tserpes, dkyr, a_menychtas, ali, christ, dora}@telecom.ntua.gr

Abstract
This paper advocates the need for a mechanism that will allow the evaluation of the provided quality of service (QoS) by a service provider to a service customer in B2B service provisioning. Furthermore, this study goes on with presenting the feasibility of this mechanism by designing and testing a reference implementation that can be used in a Service Oriented Architecture (SOA) environment which is able to support business application services. Experience gained in the frame of the NextGRID IST [1] project that focused on the business perspectives of Grid computing and adopted SOA as its baseline architecture, has shown that such a mechanism is essential for enabling the economic viability of such large scale computing platforms.

1. Introduction
In a business context the level of service quality is defined in a Service Level Agreement (SLA) and it is a product of an agreement between -at least- two parties. Given that, the focus of this study was to produce a component that would allow the evaluation of the quality of a provided service according to a variable set of criteria not necessarily common to the negotiating and agreeing sides. The criteria are set by the party that wants to evaluate the quality of the service, therefore, two instances of this component are needed; one for each party (Service Provider and Service Customer).

The result is a design of a service, named Quality Information Provision Service (QIPS) that is comprised of two main components: the History and the Analyzer. Depending on which administrative domain this service is deployed on (the provider’s or the customer’s) the component names and usage change. On the Provider side the components are called QoS Components (QoS History, QoS Analyzer) because the criterion used for the evaluation is low level QoS information, like execution details. On the Customer side the components are called QoE components (QoE History, QoE Analyzer) because the evaluation criterion is the Customer’s Quality of Experience (QoE).

As it can be understood from the usage of the Quality History component, the QIPS is largely based on recording the evaluation criteria (QoS or QoE information) of specific interactions between providers and customers. This historic information is then used in order to have a prior knowledge to the other party’s abilities and dispositions or so as to improve oneself (e.g. by configuring the job or resource requirements) knowing a priori what the requirements of an interaction are.

However, considering past behaviour in future interactions is not a new idea. In [2] it is referred as "shadow of the future" and it forms a basis for the operation of several reputation systems. In [3] a mathematical framework is described for agent systems where decisions about interactions are taken based on policies that are exact requirements on agents' past histories. Moreover, in [4] a reputation system is proposed for evaluating the trustfulness of peers using historic data on various evaluation criteria that have been set in the operations of P2P systems. Other systems that use the same basic approach are some decision support systems such as [5] where medical history is used so as to model and generalize verified clinical information. Also in [6] various quality evaluation criteria are used in order to support and automate service selection in e-commerce markets. Even though the goal looks very much alike the work presented in this paper, in essence, it differs in the methodology, since this work is based on conclusions...
made based on historic data rather than fuzzy logic based frameworks. Finally, there are two related Technical Groups working towards standardization of similar approaches: One is the OASIS Open Reputation Management Systems (ORMS) Technical Committee [7] with the objective to advance the ability to use common data formats for representing reputation data; while the second is the OASIS Web Services Quality Model (WSQM) Technical Committee [8] the object of which is to prepare a quality model (WSQM) in the context of contracting for Web services between associates conceptually, in order to secure Web services at a specific level of service quality for standardizing quality.

The main differences of the QIPS in comparison to the abovementioned approaches, is that QIPS is meant to operate in business IT environments under a service oriented framework where the main actors are a requestor and an initiator, played by the service customer and the service provider respectively. It is considered that any type of interaction between the actors of such an environment can be defined by a bipartite SLA and in cases where more than one actors of one type have to be involved then the interactions are extended through the creation of new SLAs and perhaps the inversion of the actor roles (from customer to provider, and vice versa). The negotiation process is the simplest possible and it involves the creation of a discreet offer by the customer using an SLA template [9], [10]. These offers are sent to the provider who either rejects or accepts them, leading respectively to the creation of a new offer or the creation of an SLA. It is of little value for the service providers to create offers, even though they create the SLA Templates because this process is prone to denial-of-service attacks [11].

The offers are formulated by assigning specific values in the SLA Templates, thus, creating SLA offers. As already mentioned SLA Templates are created by the service providers and contain details about the service such as the End Point Reference (EPR) [12] of the service instance, the date and time of creation; SLA or QoS terms and ranging values associated to these terms, and; billing information.

This paper is structured as such: Section 2 details the difference and the general concept of the two instances of the QIPS (QoS and QoE -based instances). Section 3 sets out to describe the architecture and implementation details of such a service, in the context of service oriented architectures. Section 4 provides an example use case scenario of QIPS in an existing system. The scenario is about the Evaluation SLA Template on behalf of the provider. Section 5 presents a second example of the QIPS usage but this time for supporting the service selection process on the customer side. Finally, Section 6 presents the conclusions of this work that were produced while testing the abovementioned components in a reference architecture. Finally Section 7 presents other works that are cited in this document.

2. Evaluation Method

As already stated, the quality of a service is evaluated using different criteria depending on which side one stands. Service providers have direct access to the application service and in most of the cases to the infrastructure itself. On the other hand, the service customers are only able to monitor the result of the service invocation using their own experience from the application (or the application outcome). The latter means that they can evaluate the provided quality level using only their own judgment as they don’t have access to the technical details of the infrastructure [13].

Given that both the sides agree on the provided level of quality once they agree to an SLA, this is a clear indication that there must be a mapping between the customer QoE and the provider QoS criteria. If there wasn’t, the two sides would never agree to an SLA. But this implies that SLAs are based on a common terminology; in a “language” that both the sides understand. In a business context, this means that the SLA terms must be closer to the understanding of the customer because providers must always “speak” the customers’ language. Customers do not understand technical details and usually they don’t care about them. Moreover, the exchange of monitored information about a service invocation would never work because of trust issues. The providers can always manipulate information for their own benefit (if they have a malevolent behaviour) and also they would never agree to expose their infrastructure and business details. Using a trusted third party (TTP) to monitor the providers on the customers’ behalf is not a necessarily viable model, because it doesn’t change the reluctance of the providers to expose trade secrets.

The conclusion is that it is only safe to consider that each party of an agreement uses different parameters to evaluate quality and the mapping between them is clear (but only known to the provider and the application developer). The provider is using low-level infrastructure and service related parameters as a basis of the evaluation whereas the customer uses high-level, application specific and close to the customer’s understanding indices. We name the first, QoS parameters, while the latter QoS indices [14].
To better explain the difference between QoS parameters and indices we refer to the example of a 3D-rendering application. Suppose that this application is offered through a Grid service, where a number of providers are making the application available. The customers are designers who create wireframes of their videos and submit them using a specific format for rendering in the Grid environment. Grid handles the functional and non-functional requirements of this application. Since the one party of the agreement is an animation designer, this implies that the terms of the SLA should be close to his understanding, regarding the video rendering application. Therefore the terms of the application-specific part of the SLA should be based on subjective quality evaluation rather than objective [15] and could something like the aspect ratio, the number of lights and shaders, the video format, the video and audio codec and the deadline.

In response to these parameters the provider must map them into low-level parameters (QoS parameters) that mainly include: the selection of the renderer and the rendering algorithm, its specific settings, resource selection and configuration and invocation of particular license services. In addition, the provider will have to take some actions in order to meet the deadline in combination with the abovementioned settings such as resource reservation and execution management.

In summary, the proposed evaluation method is based on using historic data for evaluating B2B quality provisioning. The data can originate from any kind of information source (e.g. specialised monitoring mechanisms). The selection of these sources is done by the application developer and the service provider according to the customer requirements. These requirements express the view of the customer in what criteria can be used for evaluating the past performance of those entities from a quality provisioning point of view. The algorithm for rating the quality provisioning capabilities from a set of relevant past records may differ from case to case. Of course, this holds for the selection of the relevant records too. For this reason, the QIPS must be fully configurable and not rely on static parameters and sources where it can extract these parameters.

3. Architecture

QIPS is basically a mechanism that allows quality provisioning evaluation according to specific criteria, set by the owner of the mechanism - the one who would like to evaluate quality. As mentioned above, the principle of its operation is based on the assessment of historic quality information. In essence, the abovementioned criteria are data parameters coming from various sources such as SLAs, application service consumer reports, statistical data, infrastructure information that is monitored or even benchmarks. These parameters are logged into a repository and revoked under specific conditions in order to lead to conclusions regarding the quality provisioning. In other words, the owner of the QIPS is configuring it in order to monitor particular parameters that the owner believes are of importance in order to evaluate quality provisioning.

In the beginning of an SLA negotiation phase the involved parties must always make a selection among a plethora of entities (e.g. the provider may have to select the terms of an offer or a particular service configuration and the customer may have to select a service or even a service provider). Then each party can invoke the respective QIPS instance in order to evaluate the past performance of those entities from a quality provisioning point of view. The algorithm for rating the quality provisioning capabilities from a set of relevant past records may differ from case to case. Of course, this holds for the selection of the relevant records too. For this reason, the QIPS must be fully configurable and not rely on static parameters and sources where it can extract these parameters.

3.2 Design

The design of the QIPS is depicted in Figure 1 which illustrates the basic principles of the Quality History mechanisms. The section provides an overview of this design, explaining each component's functionality and high level protocols and interfaces.

![Figure 1: Component model diagram for the QIPS components](image)

The Quality History is basically an interface to the Database component (DB Component). Its purpose is to expose the DBs API (any kind of XML DB) as a web service. Among others, it parses the incoming message and feeds it to the DB Component, according to its configuration.

The DB component is where information related to the execution of a job is stored. This information is not uncorrelated to the information contained in the SLAs. On the contrary, it derives from the Terms of the supporting SLA Schema which in our proposed
mechanism is based on the WS-Agreement specification. The data stored in the Quality History is the product of the monitoring and evaluation of execution information using whichever means or criteria (depending on whether the one monitoring is the provider or the customer). This execution is the result of an agreement (SLA) that started from an SLA Template. Therefore, the intermediate components that send this information to the Quality History must be aware of details related to the SLA Template in use. This implies that the components mediating between the execution environment and the Quality History must be properly configured with the respective policies.

The Quality Analyzer also acts as an interface for querying the DB. However, its role is somehow more complex as it is supposed to do some calculations in order to create the output report, like aggregate the results of the queries in one report.

The Quality History PROVIDES two interfaces:

a) reportQualityHistory(Event). This interface conforms with the NextGRID SLA Template Evaluation Specification [16]. It receives an XML message following the WS-Eventing specification. This event contains the reporting information regarding the executing job and the particular SLA Template. It also triggers a respective policy so as to parse the right information and store it in the right way in the DB Component. In certain applications this policy may imply the realization of a small piece of code to implement specific actions. The policy is set by the service provider and it is set by the Business Management domain of the provider.

b) Configure(Policy). This interface is there to receive policies according to a policy schema that “explains” to the Quality History component how to treat the respective incoming events. The History component must be somehow aware of what pieces of information it must expect and how to treat them. This configuration is feasible by setting a respective policy according to the application and business context.

The DB Component PROVIDES three interfaces:

a) Insert(Report). The Insert interface is part of the DB API. It results in storing the XML document (the Report) in the DB.

b) Administrative Actions Port. This port is for database administrative purposes (e.g. maintenance, corrections, etc). It has two interfaces:

i) Update(MatchReport, NewReport). The Update interface is also part of the DB API. It acts as a normal update function of the database replacing the entry which matches the MatchReport XML with the content of the NewReport.

ii) Delete(MatchReport). The Delete interface is also part of the DB API. It acts as a normal delete function of the database, deleting the entry the content of which match the MatchReport.

c) Query(SLATemplateEPR). As above, the Query interface is also part of the DB API. It acts as a select function in a relational database using the SLA Template EPR as an ID.

The Quality Analyzer PROVIDES two interfaces:

a) getQualityHistory(SLATemplateEPR). This interface triggers the querying of the database using the SLA Template EPR. The result is then the proper treatment of the reports in order to create the final Quality History report. This treatment is done by an appropriate code plug-in according to the component configuration.

b) Configure(Policy). It is symmetrical to the Configure interface of the Quality History component but here, the Quality Analyzer is the central component.

The QIPS as described can be used for various purposes. The following sections present two of its use case scenarios, one for the provider and one for the customer in a SOA environment.

4. SLA Template Evaluation

In SLA Template Evaluation the provider needs to assess the profitability of an SLA Template, which constitutes an initial offer to the customer. In this scenario the SLA Template Editor plays a central role. The SLA Template Editor is a component or person in duty bound to create SLA Templates either starting from scratch or by changing an existing. The objective of the SLA Template Editor is to create SLA Templates that will lead to offers that optimize the profit/cost ratio on behalf of the provider. In order to achieve that, he uses historic information regarding the income and penalties generated by the use of the SLA Templates. This information is provided by the QIPS.

The flow of information on the provider side that is of interest in the particular use case is depicted in Figure 2.

Figure 2: High-level view of the components' interaction in the SLA Template Evaluation scenario

The execution environment is mainly consisted of a GRIA [17] server where jobs are executed according to the terms of a specific SLA. The execution takes place on a specific platform (Linux OS) in which a dedicated wrapper is “attached” (written in PERL) to each available application. The wrapper parses the output messages translating them to the required report
(Report₁) according to the application. This report is then packaged into an event message that is sent to the Evaluator. The Evaluator is a component that translates infrastructure-level information to something meaningful according to the particular configuration.

Therefore, in this scenario, the Evaluator receives the event which triggers the respective policy (logging policy). The logging policy dictates the evaluator to activate a ‘logging action’, a java plugin that uses particular pieces of the incoming Report₁ and does some calculations in order to generate the outgoing report (Report₂). This report is again wrapped as a WS-Eventing event message and is sent to the Quality History.

Whenever the Quality History component receives this type of event, it invokes the necessary configuration (policy) and it stores the necessary information. This information is then used by the Quality Analyzer whenever it is triggered by the SLA Template Editor. The Quality Analyzer accumulates the reports and does some further calculations according to the deployed policy in order to produce the final Report (Report₃).

The final Report₃ is sent to the SLA Template Editor. The SLA Template editor can then combine the aggregated data with some billing information and conclude to information such as: a) The total profit from the application; b) The cost incurred through penalties; c) The correlation of the two above mentioned points.

In general the triggering and resulting actions are the outcome of a standardized interaction described in NextGRID Basic Profile Specification [18]. In particular it is based on the generation and relaying of WS-Eventing messages that trigger respective policies that in turn trigger the respective actions.

The purpose of the information relay is to simplify the processes that need to be triggered. Each component needs to do a minimal number of calculations for which it is needed to have all the necessary information available.

Example reports for the 3D Rendering scenario could contain the following details (as depicted in Figure 3).

5. Service Selection Decision Support

Decision Support is the scenario involving the use of the QIPS components in the customer domain. The basis of the scenario is typical. The customer is requesting a service having some specific quality requirements. In return the respective Discovery Service returns a number of available services from various providers that can fulfill the needs of the customer and perhaps the associated SLA Templates. The customer then has to select a provider according to his own business model, sign the respective SLA and use the service. Clearly, the customer will most likely want to monitor the job while it is executed in order to assure that none of the SLA terms is violated. For that reason the customer needs to evaluate the provided quality using different criteria than the provider, namely the Quality of Experience (QoE) criteria.

It yields from the above that it is important to the customer to enter agreements that will offer the best QoE. In order to achieve that, he must consult the past agreements and see whether his demanded level of QoE was satisfied.

The purpose of this Decision Support System (DSS) is to provide the customer with the support mechanism to select the service that will most likely satisfy his QoE requirements. The DSS uses history of the past collaborations in order to rate the available services. Then it makes this rating available to the customer in the form of recommendation. It is important to mention that the DSS must be fully customized by the customer in order to be able to reflect his opinion, as in some cases the DSS can be combined with a broker that can make the decisions on behalf of him (the customer). The flow of information in this case is depicted in the following figure (Figure 4).

![Figure 4: High-level view of the components’ interaction in the DSS scenario](image)

There are two asynchronous processes taking place in this scenario related to the QIPS. The first one involves the Quality History component and the submission of historic QoE information from the GRIA Client, whereas the second involves the Quality
Analyzer component and the recommendation of a service instance for invocation whenever the GRIA Client requests it. The GRIA client in this case acts as an orchestrator contacting all the respective services such as service discovery, prioritization and selection mechanisms or even brokers.

The first process is triggered when the GRIA Client collects QoE information on behalf of the customer, wraps it up in an event (Report1) which then sends to the Quality History component. This event triggers a respective policy that in essence explains the way that the reporting QoE data must be handled and stored in the database. This policy is set by the customer or the application developer. The QoE information can derive from any source, e.g., evaluation reports filled in by the end-user and SLA violations.

The second process starts with the customer requesting a service. This is done through a GRIA client who initiates a process that results in collecting a number of offers from all the available providers. The offers are given in the form of SLA Templates prefilled from the providers. For assisting the customer selecting a service, the GRIA Client submits a report (Report2) to the Quality Analyzer component with the list of the available service instances. Then the Quality Analyzer uses information from the Quality History database and evaluates it according to its configuration. Again, this configuration is done through the submission of policies set directly by the customer or most likely the application developer. A simple example of how the Quality Analyzer can treat the historic QoE information is the calculation of the violation probability using SLA Violation number against the total SLA Template usage:

\[
\text{success\_rate} = 1 - \frac{\text{num of times a SLA was violated}}{\text{num of times a SLA was used}}
\]

This process triggers the creation of a new event (Report3) that is sent to the Service Selector carrying information about the confidence level of each available service instance. Using this rating (and perhaps other information as well) the service selector invokes a particular service instance on behalf of the customer.

In both the processes, the format of the policies is defined again by the same policy schema, while the format of the Reports from the WS-Eventing Schema.

Example reports for the 3D rendering scenario could contain the details depicted in Figure 5.

6. Conclusions

Our experience with business and service-oriented IT environments led to the conclusion that there is a market need for a method that will allow trustful and reliable service quality evaluation on behalf of the parties involved in the service provisioning process. The method proposed in this paper is based on the principle that the most reliable way to predict a service’s performance in terms of quality, is to study its past behaviour [19]. Therefore, we conceptualized a mechanism that based on each party’s quality criteria it can evaluate the levels of the provided quality. QIPS has a basic, simple design so as to be able to fit the needs of multiple scenarios and operate effectively in different administrative domains. The proposed design was tested in a Grid environment based on SOA principles in the frame of NextGRID IST project. The architecture proposed by NextGRID [20] formed an ideal testbed for testing QIPS given that it is based on service-oriented principles but also because: a) All interactions are predicated by an SLA, which implies that the quality level of the provided service is defined through an SLA and at the same time that there is a business relationship between the agreed parties. These two highlight the need for evaluating quality and that trust issues in the quality evaluation process exist since the only shared information between the competing parties (provider and customer) is a contract with perhaps ambiguous terms; b) Extensive capabilities for service construction and composition are provided which means that services can be created and configured according to specific quality demands and this enhances the need for dynamic quality evaluation; c) All services operating in such an environment can expect to find a minimal service infrastructure, which implies that simplistic services with minimal information flows are needed that can communicate with any type of managerial service in an administrative domain. In other words, QIPS has to have minimal but generic interfaces to communicate with any kind of source of information or requestor in a service-oriented infrastructure. The “generic” part, is guaranteed through the use of well-established technologies such as Web Services and their associated well-accepted standards such as WS-Eventing.
Through the use of the QIPS in testbeds constructed based on the NextGRID architecture we concluded to some encouraging results. For instance, the implementation of the Decision Support case for a 3D-Rendering scenario satisfied basic needs of the end-user (the animation designer) such as advanced usability. The service selection hints, or the automation of the process ensured that the designer wouldn’t be bothered with technicalities, while at the same time he could rest assured that the system would operate on his benefit, as long as his quality criteria were correct. It is worth mentioning that the most interesting outcome of this scenario was not the reduction of the SLA violations -given that Grid middlewares such as GRIA ensure reliability to a great extent- but rather the strong QoS guarantees that the client application management system could provide. The jobs were not only able to be executed correctly with a high probability but also with a guaranteed reduced cost [21].

While testing the SLA Template Evaluation Scenario we ended up to similar results. In this case, the end-user is the service provider offer manager, who creates SLA Templates for each application. In this case, it was simple to this end-user to acquire information about the profitability of an SLA Template and make appropriate adjustments. The interesting part here, is that the offer manager had direct access to the QIPS database, which allowed him to do some kind of fine-tuning in the SLA Templates and improve their profitability (e.g. change the value ranges in some SLA terms) rather than building them from scratch.

10. References

[13] Philipp Masche et al., The increasing role of SLAs in B2B, Proceedings of the 2nd international conference on web information systems and technologies, , April 2006, Setubal, Portugal