TAPAS

IST-2001-34069

Trusted and QoS-Aware Provision of Application Services

TAPAS
D10: QoS Monitoring of Service Level Agreements

eport Version: Deliverable D10

Report Delivery Date: March 2004 (Revised May 2004)

Classification: Public Circulation

Contract Start Date: 1 April 2002 **Duration:** 36m

Project Co-ordinator: Newcastle University

Partners: Adesso, Dortmund – Germany; University College London – UK; University of Bologna – Italy; University of Cambridge – UK

Project funded by the European Community under the "Information Society Technology" Programme (1998-2002)

QoS Monitoring of Service Level Agreements

Carlos Molina-Jimenez[1], Santosh Shrivastava[1], Jon Crowcroft[2] and Panos Gevros[2]

(1) School of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, England,

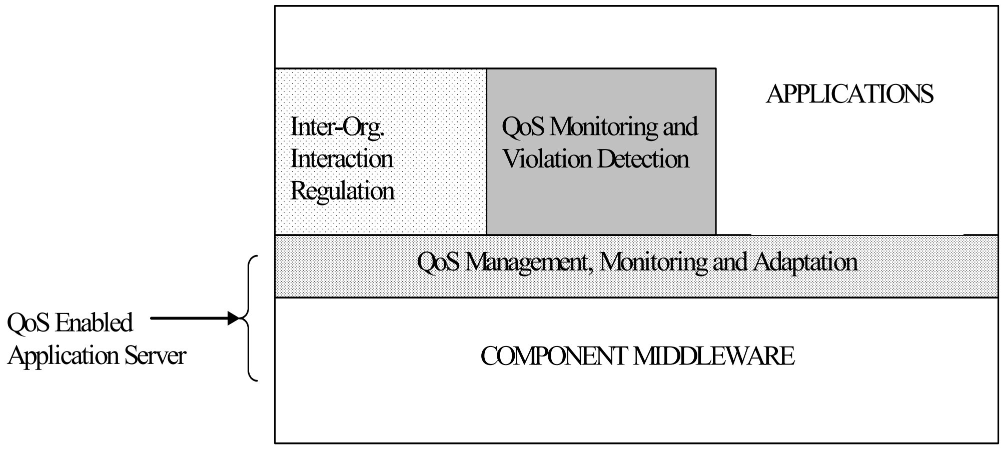
(2) Computer Laboratory, University of Cambridge, Cambridge, England.

Table of Contents

SUMMARY

This report, deliverable D10, describes the work done for developing QoS monitoring in TAPAS.

The figure shows the main features of the TAPAS architecture, described in greater detail in the supplement to deliverable D5 (An Overview of the TAPAS Architecture). If we ignore the three shaded/patterned entities (these are TAPAS specific components), then we have a fairly 'standard' application hosting environment: an application server constructed using component middleware (e.g., J2EE application server). It is the inclusion of the shaded/patterned entities that makes all the difference.


Inter-Org. Interaction Regulation
QoS Monitoring and Violation Detection
APPLICATIONS
QoS Management, Monitoring and Adaptation
QoS Enabled Application Server
COMPONENT MIDDLEWARE

TAPAS Architecture

We can see that QoS monitoring is occurring at two distinct levels: within an application server for providing QoS enabled services by controlling use of application server resources and at higher level for controlling application level QoS requirements. In TAPAS, QoS requirements are specified using the SLAng language described in deliverable reports D2 and D3.

Several of the rights and obligations in SLAs in a contract refer to the quality of service (e.g., service availability, performance guarantees). We assume that interacting entities cannot simply rely on the trust they have in one another and assume that QoS levels are being honoured. To be of practical use, a service provider must be able to demonstrate that the offered service meets the QoS levels promised to service users.

This report describes the fundamental issues that monitoring of contractual SLAs involves: SLA specification, separation of the computation and communication infrastructure of the provider, service points of presence, metric collection approaches, measurement service and evaluation and violation detection service. We develop an architecture and give reasons why currently it is practicable to offer guaranteed QoS only to consumers sharing Internet Service Providers (ISPs) with the provider. To focus only on basic issues, initially we keep our discussion abstract, general and independent of any middleware technology and implementation details.

Section 6 describes how the different components of our architecture are deployed in a real world example. The real world example we have chosen is an implementation of our QoS-enabled Group Communication Service (GCS), described in deliverable report D8. This example is appropriate as the GCS is capable of adapting to changes associated to the QoS provided by the underlying network during run-time with the aim of satisfying user requirements in the most appropriate manner. Section 7 describes how QoS monitoring will be incorporated in other parts of the TAPAS architecture.

The Auction demonstrator application planned for September 2004 (deliverable D15) will demonstrate how contractual SLAs for auctions (specified in SLAng) can be monitored and any violations detected.

1. Introduction

Monitoring of contractual Service Level Agreements (SLAs) between providers of a service (for example on-line banking, auctioning, ticket reservation, etc.) and consumers is a topic that is gaining in importance as more and more companies switch to conducting business over the Internet. For most services, any degradation in the level of the Quality of the Service (QoS) perceived at the consumer's end can have serious negative consequences. It is in the interest of the provider to make sure that the offered service meets agreed QoS. At the same time, consumers would also like assurances that QoS guarantees are being met. Contractual SLAs are intended to specify the level of QoS delivered to the consumer. For example in a stock exchange service where servers have to inform customers about market variations promptly, the latency and reliability attributes of reporting would be stipulated as clauses in the SLAs in the contract signed by the provider of the stock exchange service and customers. It is worth clarifying that the providers of business services that we discuss in this paper are known as *service providers* in the literature where as the providers of Internet connectivity are known as *Internet Service Provider* (ISPs); to prevent confusion between these two terms, we will call providers of business services, simply *providers*.

As the name suggests, monitoring of contractual SLAs is about collecting statistical metrics about the performance of a service to evaluate whether the provider complies with the level of QoS that the consumer expects. Such monitoring is frequently required to be carried out with the help of third parties to ensure that the results are trusted both by the provider and consumer. The state of art in the monitoring of SLAs by third parties is not yet well advanced: current contracts frequently leave SLAs open to multiple interpretations because they either contain ambiguous specifications of SLAs or no specification at all; likewise, they often do not unambiguously specify how the QoS attributes are to be monitored and evaluated.

It is worth mentioning that monitoring of SLAs has been studied in the past by researchers concerned with QoS of Internet communication; though work in this direction is related to ours, we emphasise that QoS of Internet traffic is not the main concern of this paper (see Section[RelatedWork]). More relevant to the central concern of this work are recent publications on monitoring of SLAs in e-commerce applications, Grid computing and Web services. However, in these works, the discussion of monitoring is often mixed with other details such as implementation and Web/Grid services technologies, making it difficult to identify, isolate, and reason about basic issues of monitoring. The contribution of our work lies actually in this direction. The aim of this paper is to bring to the system designer's attention the fundamental issues that monitoring of contractual SLAs involves: SLA specification, separation of the computation and communication infrastructure of the provider, service points of presence, metric collection approaches, measurement service and evaluation and violation detection service. We develop an architecture and give reasons why currently it is practicable to offer guaranteed QoS only to consumers sharing Internet Service Providers (ISPs) with the provider. To focus only on basic issues, we keep our discussion abstract, general and independent of any middleware technology and implementation details.

We begin by describing various issues concerned with the provisioning of networked services, and follow it up with a discussion on approaches to metric collection; this will enable us to come up with an architecture for monitoring of SLAs. We close our discussion with a summary of related work and conclusions.

2. Service Provisioning

2.1. Computation and communication subsystems

Conceptually speaking, services provided over the Internet can be regarded as composed out of two subsystems, namely, the computation and the communication subsystems (see Fig. 1).

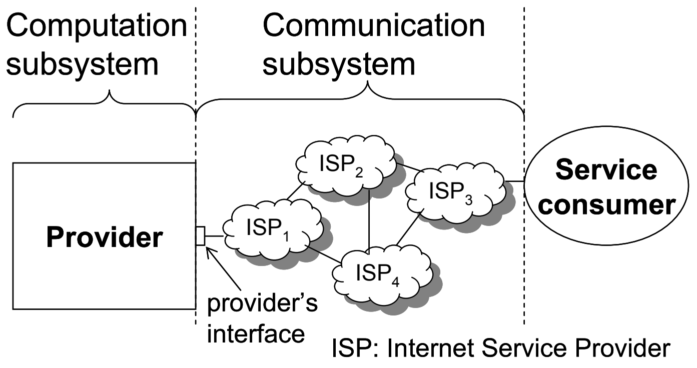


Fig. 1: Components of a service provision.

The computation subsystem consists of the infrastructure that the provider uses to produce the service before exposing it to the external world through its interface. On the other hand, the communication subsystem consists of the communication infrastructure used to deliver the service from the provider's interface to the door of the service consumer.

As suggested by the figure, in this work we abstract away the internal complexity of the computation subsystem and represent it as a single unit; however it is worth clarifying that in practice computation subsystems are composed out of several components such as computers, databases, and other computation subsystems, linked by LANs and WANs; and hidden behind an interface. Naturally, a provider can expose one or more interfaces. Our simplification is justified by the fact that it is now common practice for providers to offer their services through interfaces that hide the complexity of their infrastructures. For example, the interface would hide that the computation infrastructure includes components that belong to several autonomous and independent enterprises. As the figure suggests, with current Internet technology, the communication subsystem that the service consumer and provider see consists of a set of one or more autonomous and independent ISPs that work together to route messages from source to destination. In the figure for example, we can rely on ISP_1, ISP_2, and ISP_3 to provide the communication subsystem, alternatively, it can be built out of ISP_1, ISP_4 and ISP_3. Though not shown in the figure, we can have several service consumers interested in the service offered by the provider.

The QoS received at the end of the service consumer is affected by both, the QoS of computation subsystem and the QoS of the communication subsystem. Whereas the QoS of the computation subsystem is mostly under the control of the provider, the QoS of the communication subsystem depends on the QoS of each ISP used to compose the communication path. In practice, different ISPs provide different QoS. With this assumption in mind, it is not difficult to imagine that the QoS of the communication subsystem that relies on a communication path composed out of ISP_1, ISP_2, and ISP_3 is not necessarily the same as that of a communication path out of ISP_1, ISP_4 and ISP_3.

2.2. Service points of presence

In the discussion of Fig. 1 we mentioned that the general case is to have several service consumers interested in using a given service. It is sensible to assume that these consumers are connected to the Internet at different ISPs. This is illustrated in Fig. 2. It is in the interest of the provider to deliver its service to where its potential consumers are located. We define the *points of presence* of a provider as the ISPs from where the service can be accessed with guaranteed QoS. The provider shown in Fig. 2 has three points of presence, namely, ISP_1, ISP_4 and ISP_7. To be able to exercise effective end-to-end QoS control, a provider needs to take on the responsibility of guaranteeing agreed upon QoS not just at its interface, but at its points of presence. What matters for the service consumer is the level of QoS they will receive at a given point of presence; how this is realised should be left to the provider.

QoS guarantees are relatively easy to provide at the interface of the provider but is less likely to be used by service consumers as it requires a direct connection, for example, by means of leased lines, to the interface of the provider. However, it might be attractive to users of the service with high performance requirements, such as service owners and service monitors. Beyond the interface of the provider, the issue of guaranteed QoS is more complex because the communication subsystem located between the provider and the service consumer is likely to introduce delays, jitters (variation in the time between packets arriving), packet loss, connection loss and other communication-related disturbances. Because of this, the provider can offer its service consumers different level of QoS that will depend on the service points of presence.

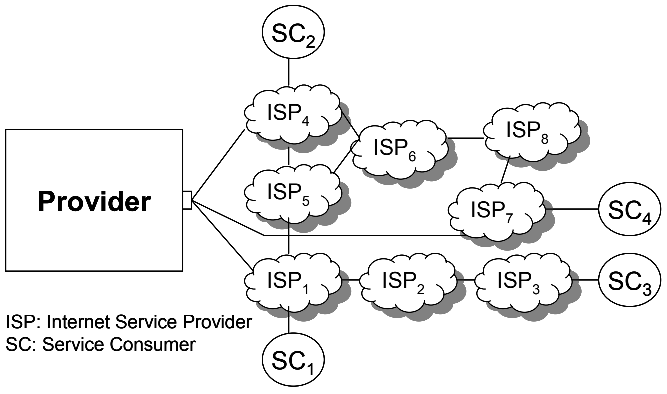


Fig. 2: Service points of presence with multi-homing.

The current business model of the dominant ISPs seems that they are more interested in providing guaranteed communication level QoS only within their own boundaries, rather than in collaborating with other ISPs to guarantee QoS over larger areas. Guaranteed QoS over large areas is extremely difficult because it implies collaboration among several autonomous organisations; each of them with their own resources, policies and business goals [1].

Another fact that prevents ISP collaboration is the structure of the relationships between ISPs. Currently, such structure is approximately hierarchical. Between tiers, ISPs are in a customer-provider relationship where the higher-tier (let us say ISP_A) is an ISP provider of transport of Internet packets to lower-tier ISPs (let us say ISP_b and ISP_c). The higher-tier ISP will often offer its customers SLAs that include clauses about overall packet treatment. Thus for example, ISP_A will offer ISP_b guaranteed level of QoS for the aggregation of packets coming from ISP_b into ISP_A and vice-verse. Unfortunately, higher-tier ISPs normally do not offer SLAs to individual hosts connected to its lower-tier ISPs. The reason for this is that the management overheads are unbearable and the fine grain mechanisms do not work well. Because of this (following our previous example) it is entirely possible for a given host connected to ISP_b to perceive poor performance while ISP_A is still, statistically speaking, meetings its obligations with respect to ISP_b. Another fact to take into consideration is that between peer ISPs there are rarely SLAs. For example, it is very uncommon to see SLAs between ISP_b and ISP_c in practice.

At the lowest level, ISPs like ISP_b and ISP_c will often offer its customers (individual end users, now) explicit service levels, which typically refer explicitly to delay and loss characteristics at the packet level. These may be statistical (e.g. the 95% of delay will be 100ms between customers of this ISP, or the mean packet loss probability will be no more than 10^{-5}), or they may be bounds (no packet delay will be more than 100ms). SLAs guarantees at the network layer is achieved today typically by network design (provisioning) and is based on extensive measurement and modelling work; this is made possible as network providers now understand the typical source behaviours, and the typical traffic patterns (the traffic matrix and its dynamics [2]).

In summary, it does seem that the most influential factor here is the current business approach of the dominant ISPs which is based on offering QoS guaranteed within their boundaries as a competitive differentiator [3]. Guaranteed QoS results in higher revenues for a provider. For this reason providers will be motivated to have as many points of presence as possible; these points of presence would be strategically located to target potential customers, for example, a provider that offers auction services in Spanish should have one or more points of presence in Mexico city and in other large Spanish speaking cities.

A provider can increase its number of points of presence by means of multi-homing Internet connection. As its name suggests, *multi-homing* consists in having several links to the Internet. This is shown in Fig. 2, where the provider has three Internet connection, namely, to ISP_1, ISP_4, and ISP_7, resulting in three points of presence.

Another approach to which a provider can resort to increase its number of points of presence and to widen its geographical coverage is to use *collocation:* providers wanting to offer guaranteed level of QoS to the ISP's subscribers can bring their servers to the ISP's site and connect them directly to the ISP's network (see Fig. 3).

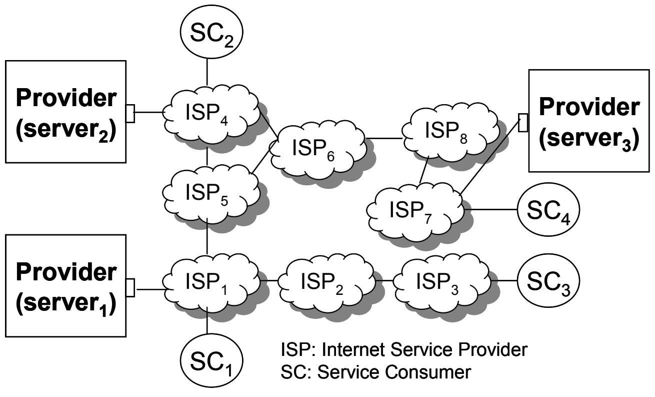


Fig. 3: Service points of presence with collocation within ISPs.

As an aside comment we can mention that these three servers might need communication amongst themselves to maintain 'single image' consistency. Depending on the degree of dependency and on the application one might need a leased line (this is not shown in the figure) to connect the three servers together.

From the discussion presented above, we can summarise that a provider can offer guaranteed level of QoS only to service consumers connected to the ISPs to which the service provider is connected. Service consumers that do not share ISPs with the service provider can be offered only best effort QoS. The service providers shown in Fig. 2 and Fig. 3 can offer guaranteed level of QoS_1 to the service consumer SC_1 and other customers connected to ISP_1; service consumer SC_3 and other consumers connected to ISP_3 can be offered only best effort QoS.

2.3. Contractual SLAs

Earlier we pointed out that current Internet business contracts often leave computation and communication requirements unspecified and open to interpretation. This mean that the receiver of the service does not have a clear idea about the quality of the service (QoS) he will receive from the provider. This undesirable situation can only be prevented with the inclusion of a precise specification of the level of computation and communication service expected from the trading partners. By this we mean a specification that has no room for multiple interpretations but a precise and unique meaning that remains the same to the contracting parties and also to third party observers that might be used to monitor the quality of the delivered service. Specifications with this degree of precision are not trivial since they require the use of a formal notation. This formal notation should allow to specify the level of service that trading partner are expected to deliver or receive and also, it should allow to perform logical and mathematical operations (such as modelling and correctness validation) to reason about the service level at delivery time and ideally prior to developing and deploying the service. An example of such formal notations is SLAng (Service Level Agreement Language) which is, as it name suggests, a formal language with a well defined syntax and semantics for describing service level specifications [4].

In the context of this paper we assume that the level of QoS that a service provider is expected to provide to a given consumer is specified in the clauses of a contract signed by the service provider and the service consumer. The SLA monitoring subsystem, whose architecture we will present in a subsequent section, could form part of a larger *electronic contract management system*. A conventional contract is a document that stipulates the rights and obligations that two or more signatories agree to honour during their interactions. An electronic contract management system will contain an *executable contract* (that is a representation of a conventional contract) to monitor and enforce the rights and obligations of the signatories at run-time. We identify two aspects of contract monitoring: (i) functional aspects concerned with monitoring that business interactions follow agreed message sequence patterns (e.g., a cancel purchase order message can only be sent if a purchase message was sent previously); and (ii) non-functional aspects concerned with the quality of service (the topic of this paper). Monitoring of functional aspects of contractual interactions is not within the scope of the paper (but see the subsequent section on related work).

As shown in Fig. 4, the contract contains, among other clauses, a list of $m \geq 1$ service level agreements (SLA^1,…,SLA^m). Each SLA^i specifies the highest (or lowest) acceptable value for a list of $n \geq 1$ parameters ($P_1^i, \ldots, P_n^i$), when certain condition, C^i, holds. For example, the contract can stipulate that Alice (the provider) has the obligation to provide Bob (the service consumer) a service with a latency not greater than three seconds when Bob places less that 10 requests per second and with a latency not greater than five seconds when Bob places more than 10 requests per second. Fig. 4 also suggests that the contract is conceptually placed between the two interacting parties to monitor their business interactions.

Central to the issue of contractual SLA monitoring is the collection of metrics about the level of QoS delivered by the provider. For this reason, we will discuss metric collection first and defer the discussion of monitoring to a subsequent section.

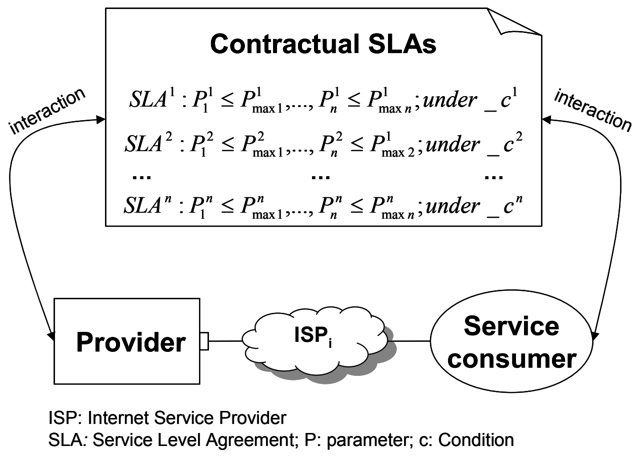

Contractual SLAs
$SLA^1 : P_1^1 \leq P_{\max 1}^1, \ldots, P_n^1 \leq P_{\max n}^1; under\_c^1$
$SLA^2 : P_1^2 \leq P_{\max 1}^2, \ldots, P_n^2 \leq P_{\max 2}^1; under\_c^2$
… … …
$SLA^n : P_1^n \leq P_{\max 1}^n, \ldots, P_n^n \leq P_{\max n}^n; under\_c^n$
interaction
interaction
Provider
ISP_i
Service consumer
ISP: Internet Service Provider
SLA: Service Level Agreement; P: parameter; c: Condition

Fig. 4: Contract between a provider and a service consumer.

3. Approaches to Metric collection

Metric collection is central to contract monitoring. As its name implies, it is all about gathering statistical information about the performance of a provider. A good discussion of the advantages and limitations of existing techniques for metric collection is presented in [5].

Metric collection involves several issues: (i) Are we using passive (packet sniffing) or active (packet interception, probe with synthetic operations) metric collectors? (ii) From what point or points (provider, service consumer or network in between) are the metrics to be collected? (iv) Who is in charge of collecting the metrics? (v) What information can be deducted from the collected metrics? With these questions in mind and without paying attention to implementation details, we can divide the existing techniques for metric collection into four general categories (see Fig. 5).

The box called MeCo in the figure represents the Metric Collector and is to be understood as the machinery used to measure and store the metrics that result from the assessment of the level of service delivered by the provider. The MeCo component can be realised as one or more pieces of software possibly in combination with some hardware components.

Fig. 5(a) shows what we call *service consumer instrumentation*. The main idea behind this scheme is that the metrics are collected by the interested party itself (the service consumer in our example) as the service is used. Because of this, the MeCo is installed inside the service consumer. In this scenario, MeCo can be realised as a piece of software installed in the service consumer's browser.

The scheme shown in Fig. 5(b) can be described as a *provider instrumentation* approach. In this scheme, the provider is in charge of collecting the metrics; consequently, the MeCo is deployed inside the provider. Notice that with this approach the measurements about the provider performance are taken directly from the provider's resources.

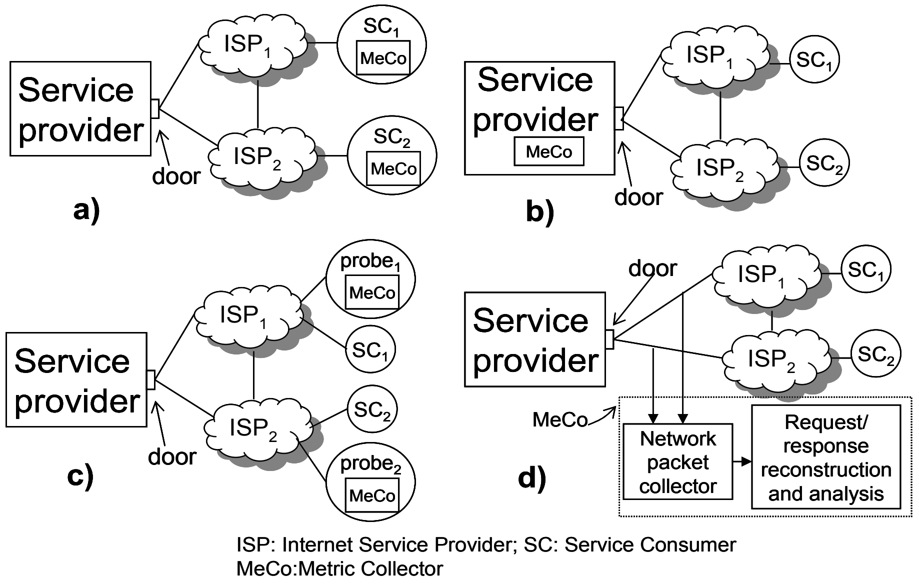


Fig. 5: Approaches to metric collection.

The scheme shown in Fig 5(c) is what can be called *periodic polling with probe clients*. In this scheme, metrics are collected neither by the provider or the service consumer but by third parties (Probe_1 and Probe_2 in our figure). Precisely, Probe_1 and Probe_2 are two trusted third parties trusted by the provider and the service consumer. From the point of view of their functionality they are two synthetic clients strategically located and equipped with a MeCo. They are there to periodically probe the provider to measure its response. The MeCo can be realised as in the service consumer instrumentation scheme.

Finally, in Fig. 5(d) we show what can be called a *network packet collection with request-response reconstruction* approach. The main idea behind this schema is to install a MeCo somewhere in the path between the provider and the service consumers to collect all the packets (either by interception or by sniffing) coming into and out of the provider. Next, the packets are analysed (by looking at the TCP headers) in order to reconstruct all the relevant request-response pairs generated by each service consumer. Since the MeCo is not installed inside the provider or the service consumer, it can be realised by a trusted third party as in the scheme of Fig.5(c).

4. An architecture for QoS monitoring by third parties

We assume that the interaction between the provider and the service consumer is regulated by a signed contract. The contract stipulates, among other things, the obligations that the two business parties are expected to honour. The goal of monitoring is to watch what a business partner is doing, to ensure that it is honouring its obligations. We assume that monitoring is to be carried out with the help of third parties to ensure that the results are trusted both by the provider and consumer. Also, for the time being we will assume that the service consumer does not want his computer to be disturbed with metric collection machinery.

4.1. Architecture

The architecture that we propose for monitoring the level of QoS delivered by a provider to a given service consumer_i at a given service point of presence ISP_i, is shown in Fig. 6. Notice that for the sake of simplicity only one point of presence and one service consumer is shown in the figure. However, in a general scenario, the provider would have one or more points of presence; each of them with an arbitrary number of service consumers.

To keep the figure and our discussion simple and without loosing generality we assume that the provision of the service is unilateral, that is, only the provider provides a service. Because of this, only the performance of the provider needs to be measured and evaluated. In practice, it is quite possible to find applications with bilateral service provision, where the contracting parties deliver something to each other and applications where the performance of the consumer affects the performance of the provider. We will show the generalisation of our architecture later. Though it is not shown in the figure, the assumption here is that the business between the provider and each of its service consumers (service consumer_i for instance) is regulated by a signed contract. The contract clearly stipulates the SLAs at the service point of presence. Similarly the contract stipulates metrics that are to be measured and with which frequencies, to asses the performance of the provider. With these observations in mind, it makes sense to think that a provider will have several instances of the scheme shown in the figure, that is, one instance for each of its service consumers.

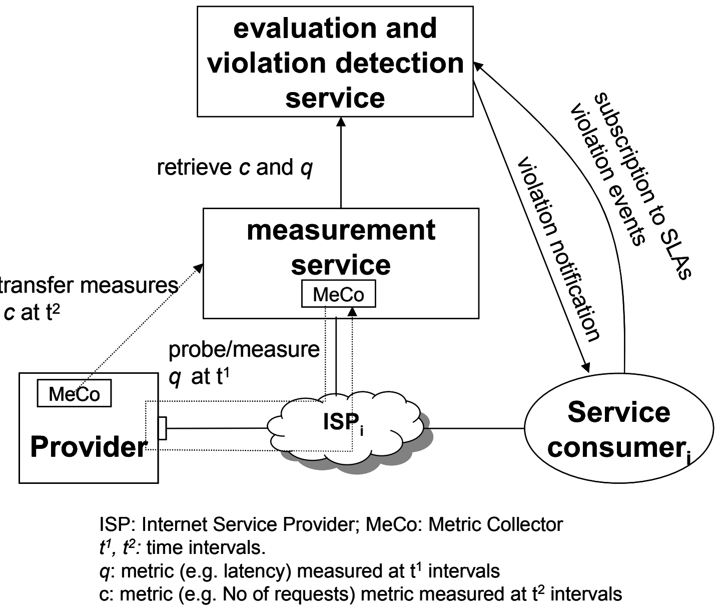

evaluation and violation detection service
retrieve c and q
measurement service
MeCo
transfer measures c at t²
probe/measure q at t¹
MeCo
Provider
ISPi
Service consumeri
violation notification
subscription to SLAs violation events
ISP: Internet Service Provider; MeCo: Metric Collector
t¹, t²: time intervals.
q: metric (e.g. latency) measured at t¹ intervals
c: metric (e.g. No of requests) metric measured at t² intervals

Fig. 6: Architecture for unilateral monitoring of QoS.

Two third party services are required:

- **Measurement service:** an enterprise trusted by the provider and the service consumer and with expertise in measuring a given list of metrics at specifies intervals and storing the collected results in its databases.
- **Evaluation and detection violation service**: an enterprise trusted by the provider and the service consumer. It is there to retrieve metrics from the databases of the measurement service, perform computation on them, compare the results of the computation against high or low watermarks and send notifications of violations to the service consumer when violations of SLAs are detected.

Notice that, for the sake of simplicity, in the figure we show single enterprises performing the functions of the measurement, and the evaluation and detection violation services. In practice, the measurement service can be performed by several enterprises that compensate their functionality with each other or replicate them to provide more reliability. Naturally, the evaluation and detection violation service can be realised in a similar way.

Notifications of violations are represented as events. We envisage an event notification system offering the service consumer the possibility to subscribe to events in which it is interested. It is not difficult to imagine that the service consumer can dynamically subscribe and unsubscribe to different events, perhaps in accordance with the momentary needs of the applications that it is running. To simplify the figure, notifications of violations are sent only to the service consumer; however, these notifications can be sent to other parties (for example, the provider) who express interest by means of subscriptions. The issue about where and how notifications of SLA violations are processed by the service consumer falls out of the interest of this work. However, we can briefly mention that such notifications can be caught by the contract management system (as implied by Fig. 4), that will, after interpreting them, take the necessary actions, such as sending a complaint note or a penalty bill to the provider.

4.2. Metric collection to build the measurement service

The contract would stipulate the level of QoS that the provider is obliged to deliver to the service consumer at the service point of presence ISP_i when certain conditions (for example, no more that 10 requests per second) in the usage of the service hold. This implies that although the service $consumer_i$ of Fig. 6 is not delivering any service to the provider, it still has obligations to honour; consequently it has to be monitored as well. It can be said that in general, monitoring is a symmetric activity. This is why measurement services rely on two kind of MeCo.

- Provider’s performance MeCo: a MeCo for collecting metrics about the level of QoS delivered by the provider at the service point of presence.
- Consumer’s behaviour MeCo: a MeCo for collecting metrics about the behaviour of the service consumer.

The critical issue here is to find a suitable approach for deploying the two MeCo (see Section 3). The architecture shown in Fig. 6 illustrates the situation where the service consumer does no wish to be disturbed unduly with metric collection responsibilities. This requirement prevented us from using schema of Fig. 5(a) for implementing the provider’s performance MeCo. A more suitable candidate to implement this MeCo is scheme Fig. 5(c). The basic idea is to think of the measurement service as a trusted third party equipped with a MeCo that is hired by the contracting parties to work as the probes. Because the contract dictates that the SLAs are to be guaranteed in all connection points within the ISP_i, the provider’s performance MeCo is free to probe the provider from anywhere as long as it does not leave ISP_i. The dotted arrowed line that goes from this MeCo to the provider and back to the MeCo, is there to show that to probe the service, this provider’s performance MeCo issues a synthetic operation (at agreed upon intervals) and waits for a response.

A limitation of this approach is that because the MeCo is connected to ISP_i at a different point as service $consumer_i$, its perception of the provider’s performance might be different from that seen by service $consumer_i$. Ideally and to enhance the accuracy on the measurements the MeCo should be placed as close as possible to service $consumer_i$. Thus if service $consumer_i$ is prepared to be disturbed with measurement responsibilities, we can place the MeCo inside service $consumer_{i,}$ this would give us the highest accuracy.

The metrics collected by the MeCo inside the measurement service can provide a great deal of information about level of QoS at the service points of presence; unfortunately, it can say little or nothing about the origin of potential problems; it does not have enough information to say whether a degradation of the service is caused by an underperformance of the provider or by an overload condition generated by the service consumer. For example it has not enough information to say whether an unsatisfactory latency is caused by a provider's malfunctioning database or by an unexpectedly high number of queries generated by the service consumer. In other words, it can not say whether the service consumer is honouring its obligations.

The most suitable approach for implementing the consumer's behaviour MeCo is the one shown in Fig. 5(b). This MeCo is in the right location to collect metrics at the level of detail needed to asses the behaviour of the service consumer. For instance, this MeCo can collect information about the number of requests issued by the service consumer and, if needed, about the resources (number of CPUs, database servers, disk memory, cryptographic keys and TCP ports) demanded by each request. Likewise, it can tell whether the service consumer is maliciously or accidentally placing illegal operations on the provider. The dotted arrowed line pointing from this MeCo to the measurement service is meant to show that the metrics collected by this MeCo are transferred at some point and over the Internet to the measurement service who stores them.

Another alternative for implementing the provider's performance MeCo is scheme of Fig 5(d). With this approach the MeCo does not probe and collect metrics from the service points of presence; instead it is connected somewhere to the communication line between the provider and ISP_i, to collect packets as explained earlier. Naturally, it is possible to implement this MeCo as a trusted third party. Unfortunately, this scheme cannot be used to measure consumer's resource usage; further, the work of packet collection and request-response reconstruction and analysis is not a trivial task; it requires the deployment of specialised hardware and software somewhere in the communication link between the provider and the service consumer; and a great deal of packet analysis, whereas the approach based on Fig. 5(c) seems to be more straightforward and natural.

The specific nature of the metrics to be collected depends on the application. On the application and on the SLAs depends also the interval at which the metrics are to be collected. This information is extracted from the contract and provided to the measurement service. For example, the measurement service might be requested to collect metric about the latency (to perform a given operation) of the service every five minutes or to collect metrics about the availability of the service every three minutes.

In the figure, we can imagine that the evaluation and detection violation service is retrieving the latest n value of the metric c, and the latest k values of the metric q. We can imagine that q is a metric that defines the latency of an operation and c is the metric that defines the number of employees from the service consumer's logged into the provider at a given moment of time, that is, the working conditions of the provider. If this is true then the evaluation and detection violation service can compute the latest average latency under the latest average number of users, with an accuracy that depends on the interval (t^1 and t^2 respectively) with which q and c are measured by the measurement service.

4.3. Mutual monitoring

In practice, there are applications where the business partners provide a service to each other, that is, where distinction between the provider and the service consumer is blurred. In these applications the interacting parties need monitor each other's QoS. This is in fact a more general scenario than the one shown in Fig. 6. The generalisation of our proposed architecture for monitoring contractual SLAs is shown in Fig.7.

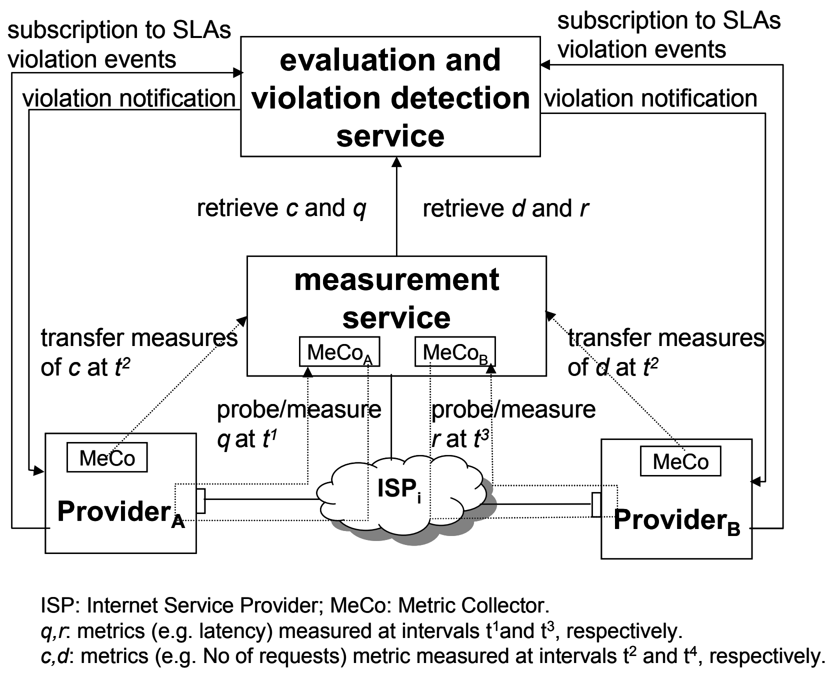

subscription to SLAs violation events
evaluation and violation detection service
subscription to SLAs violation events
violation notification
violation notification
retrieve c and q
retrieve d and r
measurement service
transfer measures of c at t2
transfer measures of d at t2
MeCoA
MeCoB
probe/measure q at t1
probe/measure r at t3
MeCo
MeCo
ISPi
ProviderA
ProviderB
ISP: Internet Service Provider; MeCo: Metric Collector.
q,r: metrics (e.g. latency) measured at intervals t1and t3, respectively.
c,d: metrics (e.g. No of requests) metric measured at intervals t2 and t4, respectively.

Fig. 7: Architecture for bilateral monitoring of QoS.

4.4. Recursivity

An aspect of monitoring that we have not discussed yet is the customer-provider relationship between the provider and ISP_i. Notice that from the point of view of Internet connection, the provider is a consumer of ISP_i. This suggests that their interaction needs monitoring as well. Incidentally, our proposal of Fig. 6 can be used to perform this task. We believe that our architecture is general enough and recursive in that it can be placed between any pair of interacting business partners to monitor their interaction. Naturally, it can be placed between ISP_i and the service $consumer_i$ to monitor their interaction.

4.5. Monitoring within a provider

A provider should take steps to ensure that the incidents of violation detection are minimised; for this it will have to take a proactive monitoring resource usage inside its enterprise. The central idea here is that rather than reacting to contractual violations notified by the notification and violation detection service, the provider should prevent them from reaching its service points of presence. For this to be possible, the provider has to deploy its own monitoring mechanisms to monitor its own resources and take corrective measures so that they deliver the expected level of QoS. Proactive monitoring and managing is a local and private activity; it is performed independently of the monitoring discussed here; though this independency does not necessarily mean that the two monitoring mechanisms cannot benefit from each other; however since proactive monitoring and managing is private, it is up to the provider to decide what, how and when to monitor, perhaps after analysing the SLAs it has signed with each of its service consumers.

5. Related work

The importance of monitoring the level of QoS delivered by providers has gained the attention of several researchers; in particular, monitoring of contractual SLA's has been identified as an important issue in several research projects. Its relevance was first identified by researchers concerned with the performance of Internet network protocols and more recently by researchers in the field of e-commerce, Grid applications and Web services.

An example of a system designed to perform network protocols monitoring is Nprobe[6]. Nprobe is a system for passively and simultaneously monitoring different levels of the protocol stack. Nprobe is built on top of the operating system and requires modification of the kernel and of the firmware of the network interface card. To work as a monitor, a computer is first deployed with the Nprobe system and then placed somewhere in the network to capture packets, process them (for example time stamp them, discard meaningless information, etc.) and store them on disk for off-line reconstruction to analyse loss, round-trip, time, etc. Another system that performs passive monitoring of multiple network protocols is Windmill[7]. Windmill was designed to measure the performance of application level protocols such as BGP, DNS and HTTP. As Nprobe, Windmill is built on top of kernel of the operating system. Once a computer is deployed with Windmill, it can be placed in strategic points in the network to passively eavesdrop on target protocols. Packets collected by Windmill are used for reconstructing the request-response interactions of the high level protocol of interest. This high level protocol reconstruction can recursively call and reconstruct the lower layers of the protocol stack to observe error conditions and other protocol events. Another system that also performs traffic monitoring is the EdgeMeter architecture[8]. EdgeMeter is a distributed meter system designed to monitor QoS of traffic of IP networks. EdgeMeter's architecture is distributed in the sense that it can be deployed to collect metric in the provider's enterprise and in the service consumer's. Metrics collected by EdgeMeter can readily be used for billing; likewise, they can be useful for network planning and QoS monitoring of applications. EdgeMeter relies on some principles of active networks: mobile code is transferred over the network to the party (provider, service consumer or both) interested in collecting metrics, where it is deployed and executed. Because of this, EdgeMeter cannot be used where this kind of disturbances are unacceptable.

It can be argued that the information collected by network protocol monitors such as Nprobe, Windmill and EdgeMeter can be used to monitor end-to-end QoS (the focus of interest of our work). In our view, this might be possible but impractical because of the substantial amount on work on request-response reconstruction; we believe that a monitoring system like our proposal of Fig. 6 and Fig. 7 that focuses on measuring the performance of representative high level operations (for example, place a bid, send a purchase order, etc.) as seen from the service consumer's perspective is more realistic. Not surprisingly several researchers are working in this direction. We will discuss next the results that are the most relevant to our work.

A system designed to monitor end-to-end performance is ETE (End-to-End) [9]. It measures performance of transactions which are considered to be formed of sequences of events (for example, request sent, socket opened, response received, etc.). For example, it can measure the time elapsed between the placement of a request to fetch a Web page and the arrival of the last bit of the requested page. Sensors to detect the occurrence of events of interest are deployed in the application, middleware and operating system layers of the provider's, the service consumer's or both, platforms. Events are received by a transaction generator who reconstructs the transactions for further response time analysis. The strong side of ETE is that it does not need to sniff or catch all incoming or outgoing network packets to reconstruct a transaction; likewise, it allows a provider to customised its measurement to the usage pattern of a given service consumer by means of an event subscription mechanisms. ETE is relevant to our work because it illustrates how a MeCo placed inside the providers of Fig. 6 and Fig. 7 can be built. Similar ideas could be used for building a MeCo inside service consumer_i.

The work that has greatly influenced our research is that conducted by the team at IBM working on Web Service Level Agreement Framework (referred to here as WSLA-F). As reported in several publications (see for example [10,11,12]), the project addresses issues related to service management in Web service environments; among these issues are the definition of a language for SLAs specification, creation and the implementation of a SLA compliant monitor. The SLA compliant monitor implementation includes a measurement service, a condition evaluation service and a deployment service. It is worth noting that this measurement service collects metrics from two points. First it collects metrics from inside the provider, that is, directly from the managed resources. Secondly, it collects metrics from outside the provider by issuing probing requests or intercepting client invocations [10]. Although the WSLA-F papers contain illuminating discussions about metric collection, metric evaluation, implementation and deployment of the SLA compliant monitor, it is driven by implementation interests; consequently, it overlooks some fundamental questions. For example, they do not discuss the effects of the communication path between the provider and the service consumer and the path between the provider and the measurement service. In particular, they do not explain to what points in the Internet the service is delivered.

Another work of relevance to ours is the one presented by Kakadia [1]. In this paper, Kakadia addresses the issue of delivering end-to-end QoS over a communication path composed out of several autonomous enterprises. The paper contains a very informative discussion about the technical problems (limited bandwidth, delays, packet looses, jitters, etc.) that a packet faces as it traverses, hop-by-hop, from the provider end to the service consumer's. The author reports that one of the main difficulties in providing QoS to consumers by this approach is that the packets must traverse several private networks with proprietary resources, QoS implementation, policies and business objectives. This heterogeneity makes it extremely difficult to implement packet classification, resource reservation and prioritisation mechanisms that cooperate to keep delays, packet losses and other communication problems under control.

The difficulties in offering guaranteed level of QoS over communication paths composed out of several vendors is pointed out in [13] as well; although it does not propose a solution for providing service with guaranteed level of quality it presents a good introduction to the topic and clear definitions of related concepts such as availability, throughput, packet loss, latency and jitter.

The issue of defining service level agreements is discussed in [4]. In this work, an XML based language called SLAng is suggested as a language for precisely defining service level agreements in contracts between providers and service consumers. SLAng elements in contracts impose behavioural constraints on providers and service consumers involved. SLAng semantics ensure absence of inconsistencies and ambiguities in the definition of the SLAs. Likewise, it provides a formal basis for comparisons between levels of service offered by different providers.

Work conducted on resource accountability by Chun et. al. [14] bears some similarity to our work. In this work, resource usage in a federated system is monitored with the purpose of ensuring that users do not accidentally or maliciously misuse the resources. The monitoring mechanism works as follows: a metric collector is associated with each active user to collect traces about what resources (CPU, memory, disk, TCP and UDP port, etc.) the user is accessing. The metric collectors report the metrics that they collect to a central module that evaluates the users' behaviour and signal anomalies. Our work is similar to this in that we are interested in collecting metrics about the provider's work load generated by the service consumer; in this situation, work load is actually the same as resource consumption. On the other hand, our work is different in that, we are interested in assessing the performance of the system as seen by the users (service consumers) form the points to where the service is delivered.

As well as experimental implementations of QoS monitoring systems, there are also commercial ones; Keynote, for example, is a company that upon request will connect a probing computer at a specified point in the Internet to periodically probe a provider; in addition, Keynote can deploy its machinery within the provider's enterprise to collect performance metrics directly from the provider's resources [15]. Keynote is a good example of the trusted third party that could play the role of the measurement services of Fig. 6 and Fig. 7.

As stated earlier, the discussion of monitoring and enforcement of business operation clauses falls outside the interest of this paper; very briefly we can mention that a possible approach to monitor and enforce business operation clauses is to use finite state machines [16]; the paper also contains a discussion on other approaches.

6. Implementing QOS Monitoring in the TAPAS Multicast Service

6.1. Architecture for Mutual Monitoring

In figure 8 we describe how the different components of our measurement service are deployed in a real world example. The real world example we have chosen is an implementation of our *QoS-enabled Group Communication Service* (GCS), described in deliverable report D8. This example is appropriate as the GCS is capable of adapting to changes associated to the QoS provided by the underlying network during run-time with the aim of satisfying user requirements in the most appropriate manner (e.g., minimising message forwarding). Furthermore, users may dictate the QoS guarantees the GCS may provide in terms of probabilistic metrics (e.g., the percentage of reliable message delivery acceptable).

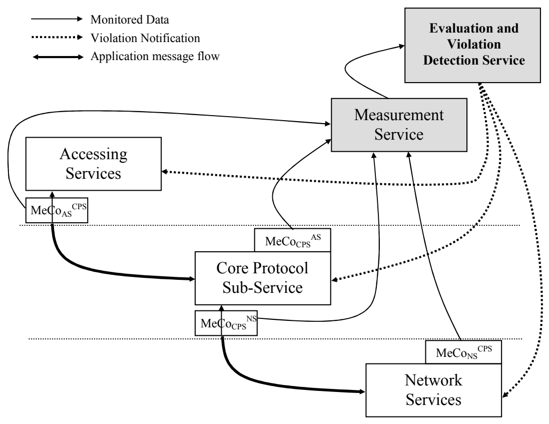


Fig. 8: Deployment of Mutual Monitoring of QoS for Reliable Multicast.

The *Core Protocol Sub-Service* (CPS) is responsible for implementing the logic of the GCS and providing user access to reliable multicast. We have described the user in our example as *accessing services* (AS). Such services may be considered the application or protocols responsible for higher level message guarantees (e.g., message ordering). The CPS requires access to underlying network services to enable message dissemination across computer networks. We describe such services as *network services* (NS). An NS may provide QoS guarantees to the CPS (e.g., mean message delivery latency) and so may directly influence the way the CPS functions.

The monitoring requirement is satisfied by Metric Collectors (MeCo). A MeCo is co-located with a protocol layer (identified by subscript) and is responsible for monitoring the QoS of a protocol layer (identified by superscript). A MeCo collects QoS metrics and passes them to the *Measurement Service*. The measurement service then correlates information gained by one or more MeCos and provides a suitably formatted message for consumption by the *evaluation and violation detection service*. The evaluation and violation detection service is responsible for informing protocol layers of SLA violations.

In our implementation each protocol layer exhibits interfaces via CORBA RPC. Via such interfaces, a protocol layer may access message dissemination services of the protocol layer directly beneath them in the protocol stack. A CORBA call back mechanism is used by a protocol layer to deliver messages to protocol layers immediately above them in the protocol stack. Data relating to the metrics of QoS is passed to the measurement service via the *Java Messaging Service* (JMS) by MeCos and then by the measurement service to the evaluation and violation detection service via JMS. The evaluation and violation detection service passes notification of violations of an SLA to interested parties (protocol layers) via JMS.

Figure 9 shows the CORBA IDL for the core protocol service. Two interfaces are provided that provide access to group lifecycle and message handling services for clients. The CPS interface provides two methods:

- **RMCast** – Allows clients to issue multicast messages to a particular group (used by AS).
- **RMDeliver** – Allows delivery of messages to CPS (used by NS).

```
module CoreProtocolService {

     struct groupMember {
          string memberID;
          string IPAddress;
          short portNumber;
     };

     struct groupProperties {
          string groupID;
          sequence<groupMember> memberList;
     };

     interface GroupManager {

          boolean createGroup(in groupProperties prop);
          boolean deleteGroup(in string groupID);
          boolean addMember(in groupMember);
          boolean deleteMember(in groupMember);
          groupProperties retrieveProperties(in string groupID);

     };

     interface CPS {

          void RMCast(in string groupID, in string msg);
          void RMDeliver(in string groupID, in string msg);

     };
};
```

Fig. 9: CORBA IDL from CPS.

We chose CORBA RPC for inter-protocol layer communication to enhance the interoperability of our system and to enable a MeCo to be integrated into our service in a non-intrusive manner via CORBA interceptors. Interceptors enable the interception of messages (down calls and up calls) without any change to application logic. Via the use of interceptors a MeCo may obtain metric measurements related to QoS of a protocol layer. For example, MeCo_{cps}^{NS} allows the gathering of metric data relating to the performance exhibited by the NS layer as viewed by the CPS.

We chose JMS for passing messages between the measurement/evaluation services and the protocol layers as such communications are message oriented and may be consumed as and when appropriate with minimal impact on performance and so promote a scalable solution. For example, there may be many instantiations of different protocol layers requiring similar message type communications with the measurement service. Rather than require synchronous RPC on a per-protocol layer basis (a non-scalable solution) a more appropriate approach would be to enable messages to be passed to the measurement service via event channels (provided by JMS) that are associated to particular message types (we assume different instances of protocol layers would use the same event channels). Figure 10 provides a sample message that would be passed via JMS relating to the performance metrics. This message would be exhibited by the CPS and relates to the settings that govern the behaviour of the RMCast protocol.

```
<?xml version = "1.0"
encoding = "UTF-8"?>
<ExchangeDoc>
  <pars>
    <eta val="4.60"/>
    <rho val="2"/>
    <omega val="0"/>
  </pars>
  </ExchangeDoc>
</xsd:schema>
```

Fig. 10: Example of performance metrics described in XML.

We now provide a detailed description of the different monitoring and evaluation present in the system. As our primary concern is the CPS, we concern ourselves with monitoring that may directly influence evaluations and violations that may impact the function of the CPS.

6.2. Monitoring & Evaluation

The overall performance of the CPS is influenced by the QoS provided by the NS and the usage made of the CPS by the AS. Therefore, the following MeCo are required for determining SLA violations in the system:

- **$\mathbf{MeCo}_{\mathbf{AS}}^{\mathbf{CPS}}$** – Co-located with AS and responsible for monitoring the QoS provided to the AS by the CPS. These metrics are based on the interception of messages between the AS and the CPS using CORBA interceptors.
- **$\mathbf{MeCo}_{\mathbf{CPS}}^{\mathbf{AS}}$** – Co-located with CPS and responsible for monitoring the usage the AS makes of the CPS. These metrics are based on information supplied directly from the CPS.
- **$\mathbf{MeCo}_{\mathbf{CPS}}^{\mathbf{NS}}$** – Co-located with CPS and responsible for monitoring the QoS provided to the CPS by the NS. These metrics are based on the interception of messages between the CPS and the NS using CORBA interceptors.
- **$\mathbf{MeCo}_{\mathbf{NS}}^{\mathbf{CPS}}$** – Co-located with NS and responsible for monitoring the usage the CPS makes of the NS. These metrics are based on information supplied directly from the NS.

From our descriptions we may determine two basic types of MeCo a protocol layer may require: (i) aid in determining if a protocol layer is used inappropriately, (ii) aid in determining the QoS provided to a protocol layer by a lower protocol layer. As described above, type (ii) uses CORBA interceptors to gain the relevant metric data. Type (i) requires a protocol layer to exhibit an interface that allows a MeCo to gather information as and when required. Such an interface is based on XML message exchange. We use SOAP based messages to transfer this information from a protocol layer to an associated MeCo of type (i).

Periodically a MeCo constructs appropriate summary information based on the metric data gathered and prepares a message in the form of XML for passing, via JMS, to the measurement service. We use XML as the evaluation and violation service uses XML based language constructs for determining if SLAs have been violated. The measurement service assumes responsibility for correlating the QoS data received from a number of MeCo instances into a form appropriate for acceptance by the evaluation and violation detection service. This is required as the tailoring of such information is dependent on the instances of SLAs that govern QoS between multiple instances of clients and servers (protocol layers). For example, different instantiations of a protocol layer may exist on a per client application basis, each with their own SLA. Placing the tailoring of QoS information at the MeCo level would require an instantiation of a MeCo on a per-client basis. However, by having a per-protocol layer type MeCo gathering QoS information we can construct the appropriate information in the measurement service. This allows a protocol layer to only require a single MeCo, irrelevant of the number of clients (higher protocol layers) associated to it. This is a more scalable solution as the MeCo appears light-weight in the fact that the unnecessary processing burden related to the many different SLAs a protocol layer may be participating in is confined to the measurement service.

Protocol layers register their interest to event channels (provided by JMS) on a per-SLA basis. Violation of an SLA results in the evaluation and violation detection service issuing an XML message detailing the type of violation that has occurred on the appropriate SLA event channel. The responsibility of consuming such messages is left to the individual protocol layers. This decoupled communication is ideal in that the evaluation and violation service does not have to contact directly each protocol layer that is associated to an SLA. Once protocol layers have consumed messages indicating SLA violation the negotiation process between protocol layers may be enacted. Such negotiation is protocol layer dependent and is detailed in previous literature related to CPS, AS and NS.

7. Current Status and Integration Plan

7.1. Current status

As discussed at the beginning of this report, QoS monitoring of SLAs is occurring at two distinct levels: within an application server for providing QoS enabled services by controlling use of application server resources and at higher level for controlling application level QoS requirements. This two level monitoring view generalises to multiple (greater than two) level view, because an application server itself makes use of various services which could be QoS enabled and each such service will need QoS monitoring. Given this observation, we have developed basic concepts of monitoring of contractual SLAs of services and presented the monitoring architecture (figs. 6 and 7). We have performed an implementation of this architecture as required within the QoS enabled multicast service (see section 6, and deliverable report D8). Next we discuss plans for implementing monitoring in other parts of TAPAS.

7.2. Integration Plan

The work performed by us so far forms the basis of the integration effort to be completed by September 2004 as a part of TAPAS platform demonstration, auction application hosting (deliverable D15). The work being extended in three directions:

(i) *Integration with SLAng*: Implementing automatic extraction of measurement, monitoring and violation detection information from SLAng SLA specification.

(ii) *Integration with QoS enabled application server*: Implementing QoS monitoring of resource usage within the TAPAS application server.

(iii) *Application level monitoring*: Implementing QoS monitoring and SLA violation detection of auction hosting application.

7.2.1. Integration with SLAng

In TAPAS, QoS requirements are specified using the SLAng language described in deliverable reports D2 and D3. Figure 11 provides a diagram that describes the general principles of our approach to SLAng and monitoring integration. Using the diagram we now present an overview of the major parts of our service followed by a more detailed description of how implementation of our system is achieved.

A monitoring and measurement service is installed on a per-node basis within the JBOSS cluster configuration. Such a service is responsible for gathering metric data from a cluster and propagating this data to the SLAng engine. The SLAng engine implements the logic that may determine if violation of SLA has or has not occurred. An appropriate deployment scenario would be a SLAng engine located on a distinct node in the system (hosted by a third party). The monitoring part of the service is responsible for gathering metric data from a number of metric collectors (MeCos) that are distributed throughout a node. MeCos intercept client requests and associated replies and construct messages that include client request related metric data and sends these messages to the monitoring part of the service. As the system is a wholly Java solution, the sending of messages from MeCo to monitoring may be achieved via Java Remote Method Invocation (RMI). The measurement part of the service is responsible for correlating all the metric data into a form acceptable by the SLAng engine and sending such information to the SLAng engine. The sending of metric data is achieved via message oriented middleware (MOM). As we are predominantly concerned with a Java solution we use JMS as our MOM. Once the SLAng engine has consumed metric data from the JMS violation of SLA identification may be carried out. If such a violation has occurred a message is constructed by the SLAng engine that informs the participants in an SLA that the SLA has been violated. This message is distributed to the appropriate SLA participants via JMS.

MeCos may be placed within a container that is responsible for managing bean execution (application logic tier) or in axis that is responsible for managing client requests (web service tier). In the container scenario interceptors are placed in the JBOSS interceptor stack and capture data related to bean invocation via RMI. In the axis scenario interceptors represent axis handlers and are used to capture data related to SOAP invocations. MeCos capture three pieces of data relating to client invocations: (i) sender ID, (ii) type of invocation (determined by method name), (iii) and the time taken to satisfy client request. A possible deployment scenario for MeCos is described in figure 11. We see that Organization X is accessing a service provided by organization Y. Organization X has a MeCo located in its web tier that records how long requests take to succeed and propagates such information to its own monitoring and management service that in turn supplies appropriate information to the SLAng engine via JMS. In organization Y a MeCo in the web tier captures information relating to the usage of services belonging to organization Y. A further MeCo is installed in a node associated to the JBOSS cluster belonging to organization Y that captures information relating to the usage of specific EJBs hosted by organization Y. The MeCos in organization Y propagate their information to organization Y's monitoring and measurement service. This information is then suitably formatted by the monitoring and measurement service and passed to the SLAng engine via JMS.

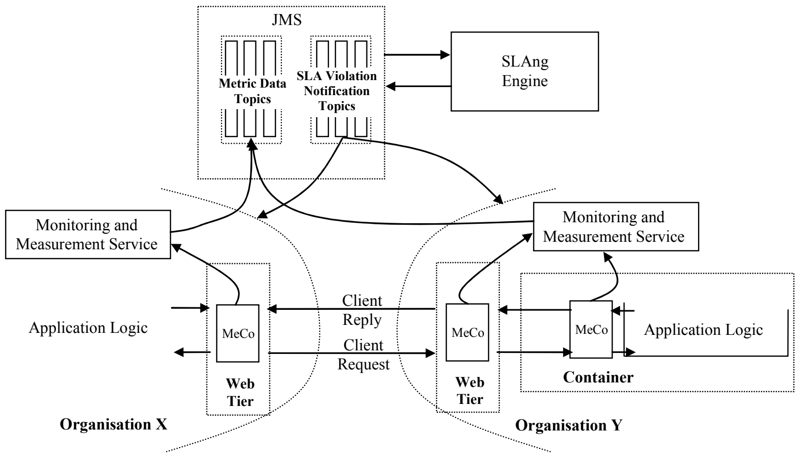


Figure 11: Monitoring Integration with SLAng

We assume a number of invocation types may be common across multiple organizations and SLAs. A topic is created in JMS that is associated with a type of invocation, allowing any organization that accesses/provides such an invocation to issues their metric data to a well defined topic (a topic represents a medium for communicating related messages between senders and receivers). In addition to the three parameter types associated to metric information identified by a MeCo, the identifier of the service provider must be added to a piece of metric information before placing such information as a message on the appropriate topic. This allows the SLAng engine to determine the parties involved in a consumer/provider relation and apply the appropriate SLA in determining the validity of such a relation. JMS is used to propagate SLA violations back to organizations. Each SLA has a unique identifier. It is this identifier that is used to create a topic from which all interested parties (involved in an SLA) may subscribe and consume all notifications of SLA violations.

We use JMS as the basis for message exchange between the SLAng engine and organizations as the addition of new organizations and SLAs to our model would only require the creation of the appropriate topics and subscriptions. There is no need to tightly couple communications (e.g., RPC) between organizations and the SLAng engine, making the MOM approach significantly more scalable and flexible.

7.2.2. Integration with QoS enabled application server

QoS control in application server is being implemented, as discussed in D7, by two principal middleware services, named *Configuration Service (CS)* and *Controller Service (CTRL)*. The former service is responsible for discovering, negotiating, and reserving the resources necessary to meet the QoS requirements of a particular application component, hosted by that application server; the latter service is responsible for monitoring the reserved resources, and possibly adapting the component execution in case the QoS delivered by these resources deviates from that required by the component itself. Work is under way (led by Bologna) on the development of CS and CTRL services within the JBOSS application server. Figure 12 shows how these two services are intended to fit in with the other JBOSS services.

JBoss consists of a collection of middleware services whose implementation is based on a microkernel termed Java Management eXtension (JMX); these services can interoperate via the JMX microkernel. Specifically, JMX provides the Java developers with a common software bus that allows them to integrate components, such as modules, containers, and plug-ins. These components are declared as *MBean services* that can be loaded into JBoss, and can be administered using JMX.

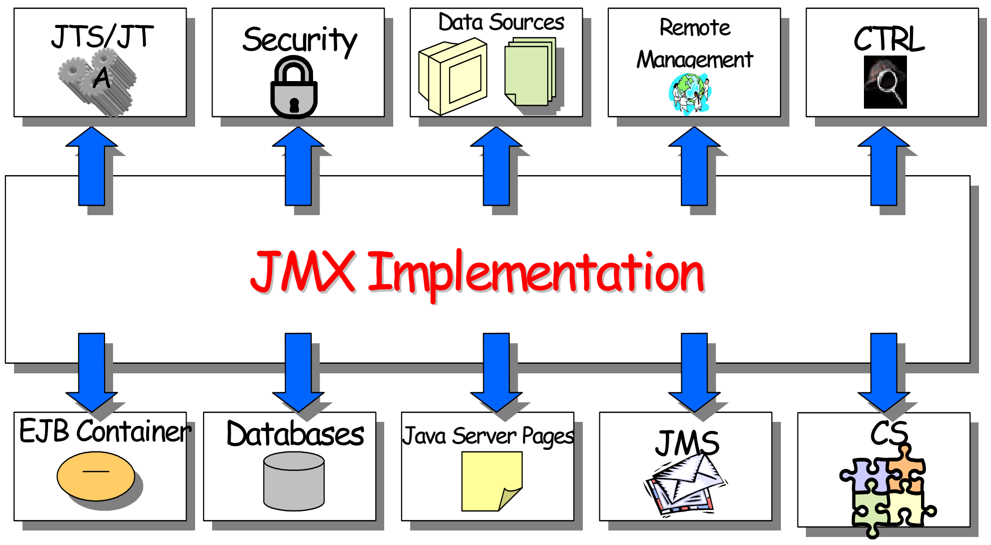


Fig. 12: The JBoss JMX Microkernel

Both the CS and CTRL services work at two level: at the level of individual nodes (computers) of a cluster (micro level); and at the level of a cluster (macro level). We describe here the macro level functionality of the CTRL service, implemented by the MacroResourceManager MBean.

The macro resource middleware tier of the TAPAS QoS-enabled application server is currently implemented by the MacroResourceManager MBean, and integrated into the JBoss server by means of the JMX software bus.

Figure 12 depicts the UML diagram of the principal MBeans we have implemented in order to extend the JBoss application server, version 3.2.3; each MBean in Fig. 13 exposes an interface consisting of methods to be used for invoking it.

The MacroResourceManager MBean uses the following auxiliary JMX services, in order to implement the cluster configuration, reconfiguration, and monitoring functionalities. Specifically, the it uses a MeasurementService, in order to save periodically the cluster state, and the InterpreterService in order to transform the initial SLA, specified in a xml form, into a Java object.

Both the MeasurementService and the InterpreterService are currently implemented as MBeans.

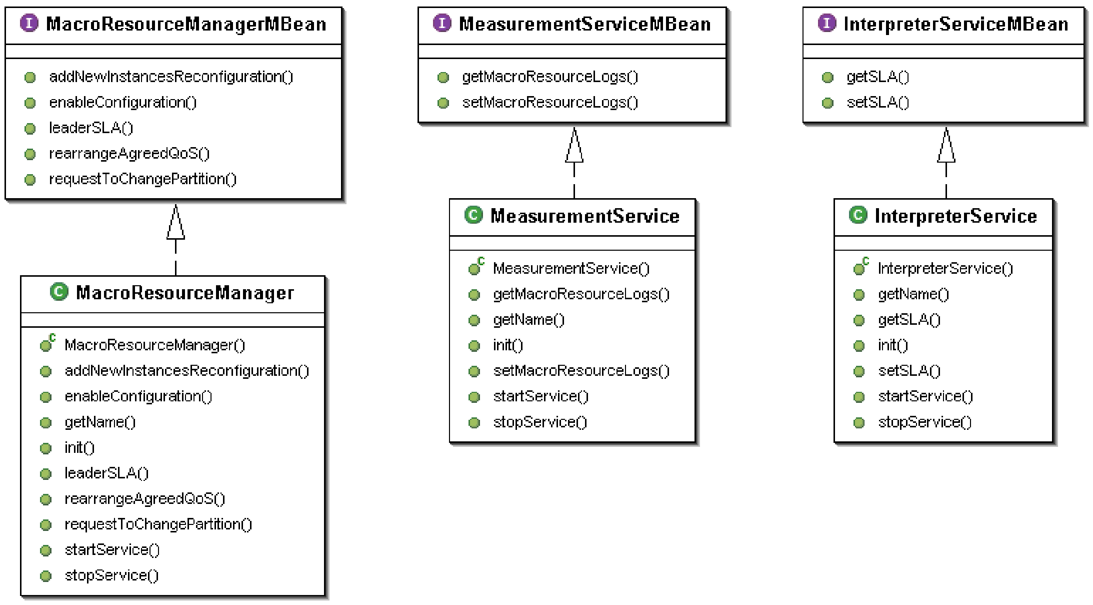


Figure 13: MBean Classes and Interfaces

As mentioned above, the macro resource tier performs two principal activities; namely, the configuration/reconfiguration of the cluster, and the cluster monitoring. Figure 14 below summarizes the implemented Java classes that form the currently available Macro Resource Middleware Tier prototype.

The architecture of the cluster monitoring is based on the high-level QoS Monitoring architecture described in section 4. Specifically, the MacroResourceManager MBean implements the cluster monitoring using both the MeasurementService MBean, and the Evaluation and Violation Detection Service (implemented as a class, as illustrated in Fig. 14, below).

This latter Service is responsible for monitoring, at run-time, the adherence of the run-time execution environment to the SLA; i.e., it detects whether the run-time environment conditions (obtained from the MeasurementService) are close to violating the SLA, and decides the cluster reconfiguration strategy to be performed, if necessary.

The cluster monitoring is generally used for checking the status of the hosted application home cluster. This can be done by interrogating a specific cluster membership Metric Collector (MeCo) that detects (i) the current view of the cluster membership, (ii) new members that join the cluster, (iii) dead members that leave the cluster, and (iv) the performance status of the cluster, in terms of the *throughput*, *response time*, and *probability of rejection* architectural parameters (note that these three parameters allow one to detect whether or not the nodes of the cluster are overloaded).

The monitoring activity is currently implemented by the MacroResourceMonitoringImpl Java class (Figure 14), and enabled by the MacroResourceManager MBean, which starts the monitoring thread. This thread is currently able to check the (i) overall cluster membership (through the getMembership() method in Figure 14) (ii) the new members that join the home cluster (through the getNewMembers() method in Figure 14) and (iii) the dead members that leave the home cluster (through getDeadMembers() method in Figure 14).

The cluster membership MeCo, which provides one with the above mentioned information about the clustered node membership, is implemented by using the JGroup communication interface available in JBoss, and uses the HAPartition service of the JBoss clustering framework.

Finally, the current implementation of the MacroResourceMonitoring sends periodically the data obtained from the MeCo service, about the cluster membership, to the Measurement Service, whose task is to maintain them in stable storage (e.g., for logging purposes).

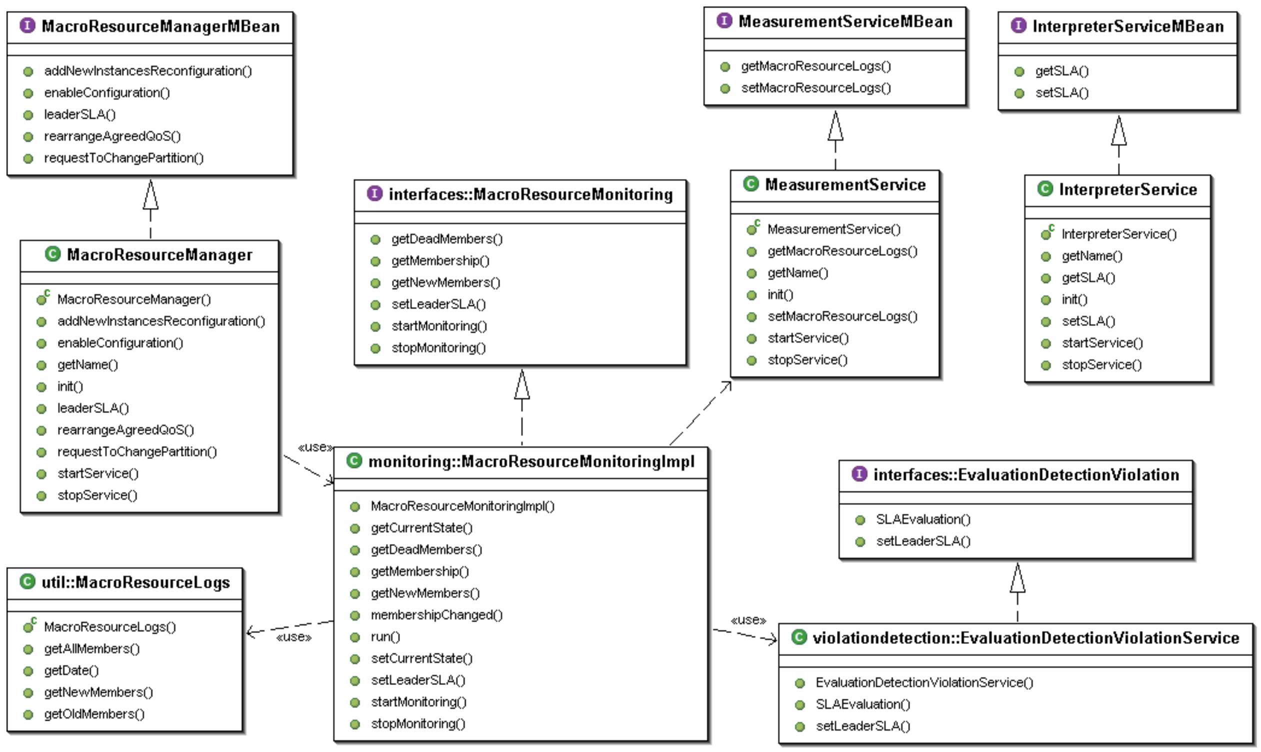


Figure 14: TAPAS Macro Resource Middleware Tier

7.2.3. Application level monitoring

It is necessary to ensure that a hosted application actually meets the QoS requirements, so we need to measure various application level QoS parameters, calculate QoS levels and report any violations. The specification of the auction application that will form the part of September 2004 demonstration (deliverable D15) is given in deliverable D13. The QoS monitoring will be implemented as a third party service, using JMS messaging middleware, in line with the scheme presented in fig. 11. Fig. 14 depicts the overall architecture of the auction application. The various parameters of metric collectors (MeCO), measurement and evaluation and violation detection services will be derived from the SLAng specifications.

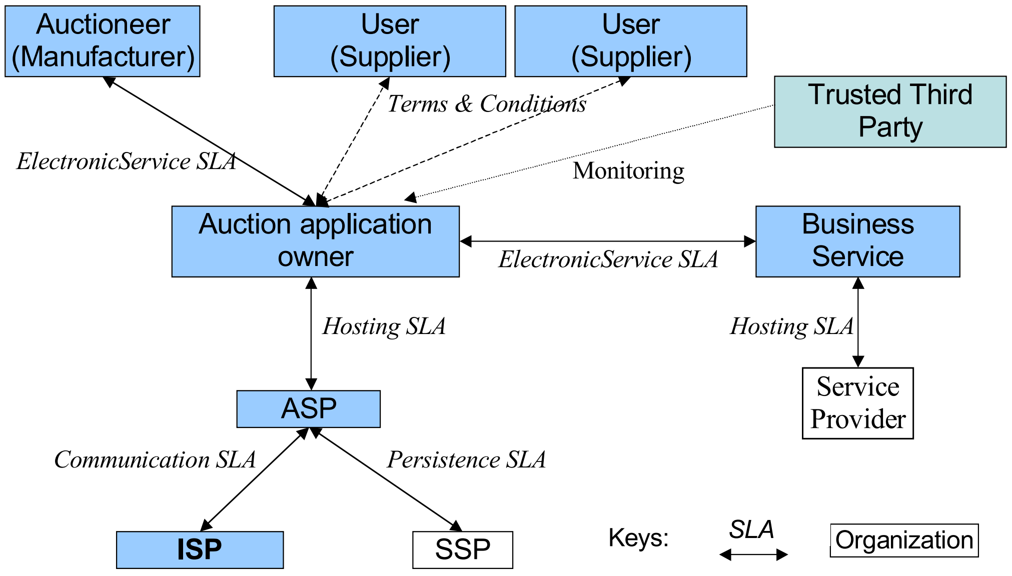


Fig. 15: Auction application

References

[1] Deepak Kakadia, “Tech Concepts: Enterprise QoS Policy Based Systems & Network Management”, Sun Microsystems, available at: http://wwws.sun.com/software/bandwidth/wp-policy/

[2] K. Papagiannaki, N. Taft, Z. Zhang, C. Diot, “Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models”, IEEE Infocom, San Francisco, Apr. 2003.

[3] Marjory S. Blumenthal and David D. Clark, “Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave new World”, ACM Transactions on Internet Technology, Vol. 1, No. 1, Aug. 2001.

[4] J. Skene, D. Lamanna and W. Emmerich, “Precise Service Level Agreements”, proceedings of the 26th International Conference on Software Engineering (ICSE'04), May 2004, Edinburgh, Scotland.

[5] Ludmila Cherkasova Yun Fu, Wenting Tang and Amin Vahdat, "Measuring and Characterizing End-to-End Internet Service Performance", ACM Transactions on Internet Technology, Vol. 3, No. 4, Nov. 2003.

[6] Andrew Moore, James Hall, Christian Kreibich, Euan Harris and Ian Pratt, "Architecture of a Network Monitor", in Proceedings of the Passive and Active Measurement Workshop, La Jolla California, Apr. 6-8, 2003.

[7] David Watson, G. Rober Malan and Farnam Jahanian, "An extensible probe architecture for network protocol performance measurement", Software Practice and Experience, Vol. 34, 2004.

[8] Marcelo Pias and Steve Wilbur, "EdgeMeter: Distributed Network metering", In Proceedings of the IEEE Openarch 2001 conference, short paper session, Anchorage, Alaska, Apr. 2001.

[9] Joseph L. Hellerstein, Mark M. Maccabee, W. Nathaniel Mills III and John J. Turek, "ETE: A Customizable Approach to Measuring End-to-End Response Times and their Components in Distributed Systems", In proceedings of the 19th International Conference on Distributed Computing Systems, Austin, TX, 31 May - 4 Jun 1999.

[10] Keller, A., Ludwig, H., "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", Journal of Network and Systems Management, Special Issue on E-Business Management, Vol. 11, No. 1, Plenum Publishing Corporation, Mar. 2003, (also, IBM Research Report RC22456)

[11] Debusmann, M., Keller, A., "SLA-driven Management of Distributed Systems using the Common Information Model", In proceedings of the 8th International IFIP/IEEE Symposium on Integrated Management (IM 2003), Colorado Springs, CO, Mar. 2003.

[12] Keller, A., Ludwig, H., "Defining and Monitoring Service Level Agreements for dynamic e-Business", In Proceedings of the 16th USENIX System Administration Conference (LISA'02), Philadelphia, PA, Nov. 2002.

[13] Amitava Dutta-Roy, "The cost of quality in Internet-style networks", IEEE Spectrum, Vol. 37, Issue 9, Sep. 2000.

[14] Brent N. Chun, Andy Bavier, "Decentralized Trust Management and Accountability in Federated Systems", In Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), Jan. 05-08, 2004, Big Island, Hawaii.

[15] Keynote Systems, Inc. http://www.keynote.com.

[16] Carlos Molina-Jimenez, Santosh Shrivastava, Ellis Solaiman, and John Warne, "Run-time Monitoring and Enforcement of Electronic Contracts", Electronic Commerce Research and Applications Vol.3, No.2, 2004.