Available online at www.sciencedirect.com
SCIENCE DIRECT
Future Generation Computer Systems 22 (2006) 721–731
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Evaluating architectures for independently auditing service level agreements

Ana Carolina Barbosa, Jacques Sauvé, Walfredo Cirne*, Mirna Carelli

Universidade Federal de Campina Grande, Programa de Pós-Graduação em Ciência da Computação, Av. Aprígio Veloso, 882 – 58.109-970, Campina Grande – PB, Brazil

Received 26 August 2005; received in revised form 25 December 2005; accepted 5 January 2006
Available online 9 March 2006

Abstract

Web and grid services are quickly maturing as a technology that allows for the integration of applications belonging to different administrative domains, enabling much faster and more efficient business-to-business arrangements. For such an integration to be effective, the provider and the consumer of a service must negotiate a service level agreement (SLA), i.e. a contract that specifies what one party can expect from the other. But, since SLAs are just contracts, auditing is key to assure that they hold. However, auditing can be very challenging when the parties do not blindly trust each other, which is expected to be the common case for large grid deployments. We here evaluate six architectures that perform SLA auditing both quantitatively and qualitatively. The quantitative evaluation focuses on the performance penalty that auditing introduces. The qualitative evaluation compares the architectures based on aspects such as intrusiveness, trust, use of extra requests, possibility of preferential treatment, possibility of auditing consumer load, and possibility of auditing encrypted messages. We conclude that no single architecture seems to be the best solution for all cases and indicate where each one is best suited.

Keywords: Service auditing; Independent auditing; Grid; Web services; Service level agreements

1. Introduction

Web services are maturing as a technology that allows for the integration of applications belonging to different administrative domains, enabling much faster and more efficient business-to-business arrangements. Grid services have recently evolved from web services and high-performance grid technology and promise an unprecedented level of service dynamism. In the grid vision, the relationship between suppliers and consumers is very dynamic and the services are transient (i.e. have a lifetime). Since applications integrated via web and grid services typically span multiple administrative domains, contracts called service level agreements (SLAs) are used to establish what a consumer application can expect from a provider and vice versa. An SLA is composed of a set of service level objectives (SLOs), which are evaluated using measurable data, called service level indicators (SLIs).

Since SLAs are simply contracts, they do not provide hard guarantees per se. One must audit them to ensure that they hold. However, since these applications typically belong to different entities, there may be no implicit trust relationship between them. In particular, a consumer may be suspicious of the provider's audit findings and vice versa. When there is no trust between services, auditing can be done by an independent mutually trusted third-party entity.

In a very dynamic service environment, SLA management should be an automatic and dynamic process. This process is composed of the phases of SLA negotiation, definition, auditing, notification of violation and triggering of management actions when SLA non-compliance is detected. Although SLA management is addressed by other works, such as [1,3,5,7,11,13], they do not analyze the SLA auditing phase in detail.

This paper fills this gap, evaluating six architectures for independently auditing computational services. We address issues concerning trust between services; where and how to instrument the services and evaluate their SLAs; about the response time increase from the consumer point of view caused by the introduction of the SLA auditing process, as well as other aspects. Although our results are not tied to web and grid service technology, we believe that automated service auditing is key for the full deployment of such technology. Our efforts extend initial results presented in [2] and are part of the OurGrid

* Corresponding author. Tel.: +55 83 3310 1433; fax: +55 83 3310 1365.
E-mail addresses: carolina@dsc.ufcg.edu.br (A.C. Barbosa), jacques@dsc.ufcg.edu.br (J. Sauvé), walfredo@dsc.ufcg.edu.br (W. Cirne), mirna@dsc.ufcg.edu.br (M. Carelli).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.01.001

project developed in collaboration between the Universidade Federal de Campina Grande (UFCG) and Hewlett Packard (HP) [14].

The paper is organized as follows: Section 2 presents related work; Section 3 presents some issues concerning SLA auditing; Section 4 describes possible architectures for auditing services, and evaluates them qualitatively; Section 5 develops a quantitative performance analysis of these architectures; and, finally, Section 6 concludes the paper.

2. Related work

Service level agreements have been applied to managing web services and, recently, to managing resources and grid services. This section presents some work being developed in this area.

Web service distributed management (WSDM) is an Organization for the Advancement of Structured Information Standards (OASIS) standard that introduces a framework for managing resources through web services [13]. The managed objects are web services representing resources or services. These web services implement management interfaces and are thus responsible for providing management information. In this framework, although there are no constraints on using SLAs, web service management does not consider them and there is consequently no concern with trust in the management information provided by the managed objects. In contrast to the WSDM, our work focuses on the SLA auditing phase.

The web service level agreement (WSLA) framework [5] is an IBM effort for specifying and monitoring SLAs for web services. It consists of an SLA definition language based on an XML scheme and of a runtime architecture that provides several monitoring services that can be outsourced to assure greater objectivity. The IBM project offers SLA management for web services in the phases of SLA negotiation, monitoring and triggering of corrective actions by management tools when an SLA violation is detected. The monitoring process can be performed either by signing parties or by third parties, through investigation and interception of consumer requests. Our work is different because it focuses, details and analyzes the monitoring process, which we call the SLA auditing process. Although the WSLA framework cites many possibilities for performing instrumentation and monitoring of services, it does not explore each of these possibilities in detail. This paper analyzes six possible architectures for SLA auditing between services, their advantages and drawbacks related to diverse factors, such as the trust issue and the impact on service provider performance due to the auditing process.

Other related works are the G-QoSM framework [1], SEQUIN [3], and the RAC Utility Model [11]. The G-QoSM framework addresses SLA management for grid services, including the process of service discovery based on quality of service (QoS) requirements, SLA negotiation, monitoring and execution of management actions in case of SLA non-compliance [1]. It aims to enable grid services to describe their QoS properties and enable their users to select services/resources based on QoS requirements. QoS management is performed based on SLAs in order to provide adaptation to attain the consumers' QoS requirements. SEQUIN defines and implements an end-to-end approach to quality of service, operating across multiple management domains and exploiting a combination of link layer technologies [3]. However, the SLI gathering phase is not detailed. It is only suggested that the infrastructure have monitoring equipment or functionality placed in intermediate positions along the end-to-end path between end-users, as well as at the premises of each end-user. Khana et al. deal with the problem of accepting or rejecting an Internet session request according to SLA requirements [11]. They introduce a RAC (routing and admission control) system using a utility model, which may be used for admission control of new sessions, resource allocation to existing sessions, and dynamic resource reallocation to cope with changes in system sessions and/or network properties. In common, these works do not include in their scope considerations about trust relationships among the parties involved in the SLA. The SLA auditing phase is neither detailed nor analyzed as it is in our work.

The Globus project has an architecture for discovery, reservation and allocation of heterogeneous resources based on QoS, called the General-purpose Architecture for Reservation and Allocation (GARA) [7]. The client application requests the reservation agent to reserve resources aiming to attain particular QoS requirements. After the resource reservation agent finds resources that can attain the requirements, the allocation agent allocates the resources and returns handlers to the client. The client application can perform QoS monitoring of a resource through its handler. Although this architecture allows QoS monitoring of allocated resources, this process is not detailed. In particular, there is no discussion on how non-trusty parties agree on the monitoring result.

3. Issues concerning SLA auditing

SLAs need to be audited to give real QoS guarantees. However, when one thinks of auditing an SLA established between services belonging to diverse entities under different administrative control, one faces a major trust problem. In fact, when the providing and consuming entities have a strong trust relationship, they can believe in each other's SLIs (such as consumer rate of submitted requests and service response time). While current grid deployments are small in scale and are built over human trust relations, the vision is that the grid is going to scale planet-wide and promote a highly dynamic service ecosystem, in which services discover and bind to each other on the fly [6]. In such a scenario, strong pre-established trust relationships are not expected to be the norm. This can create serious problems because SLIs can also be recorded at the other end and reported results from both parties may not match. For example, the consumer can measure service response time and the provider can compute request rate. However, this does not address the real issue, which is trust. If the consumer claims that the SLA was not met because, for instance, response time was too high, the provider can dispute this claim by presenting its own (smaller) response times.

Another issue to be addressed is that the services may not be willing to instrument the code to calculate SLIs for auditing, or this may not be feasible or appropriate due to the interaction dynamicity between services. To circumvent the trust problem and the intrusiveness in service code, we argue that an independent third-party auditor should do the instrumentation and the SLA evaluation. Of course, both provider and consumer must agree to use the auditor beforehand and must trust it. Therefore, we expect to see a few widely known companies providing SLA auditing in the grid.

However, this proposal begs the question: How is the auditor going to obtain trustworthy SLIs? Asking the consumer and provider for SLIs suffers from the same problems just described. One idea is for an entity called an inspector to probe the provider as if it were the consumer. A basic drawback is the overhead generated: that is, the inspector can reduce service performance due to the additional requests issued to obtain SLIs. Also, if the provider identifies a request as coming from the inspector, it may give preferential treatment to the request, meaning that the SLIs obtained by the inspector will not reflect the performance seen by a real customer. Furthermore, probing only helps to get provider-related SLIs; gauging consumer-related SLIs (such as the submitted load) remains an issue.

Another issue in auditing SLAs relates to the change in real SLI values by including mechanisms to instrument and audit the services; in other words, the auditing process itself can affect the values observed during interactions between provider and consumer. This difference in SLI values due to the auditing process is here termed Measurement Interference Error (MIE). By means of the MIE, we can analyze the response time increase viewed by the consumer due to the auditing process.

4. Architectures for independently auditing services

We started this research aiming to determine the best way for an independent third-party SLA auditor to work. We found, however, that there is no simple solution to this problem. Different architectures have different pros and cons, and the solution considered best depends on the criteria judged to be most important. We here qualitatively evaluate six architectures for SLA auditing. In the next section, we quantitatively evaluate their performance. The architectures presented here represent reasonable ways to tackle the problem of SLA auditing: we do not claim that they are new. For example, WSLA mentions that SLA monitoring can be done by third parties, possibly through probing or request interception. Our contribution is to describe the architectures using a common framework and, principally, to evaluate and compare them to each other along dimensions of functionality and performance.

4.1. Naive architecture

A basic solution to deal with SLA auditing introduces a third party – called an auditor – responsible for evaluating the SLA established between the signing parties (see Fig. 1). In this solution, the auditor obtains SLIs from the parties and compares them with the values agreed upon in the SLA to verify contract compliance. SLA evaluation can be automated for SLAs defined through a common language [15]. Thus, a single auditor service can be used to evaluate several SLAs established for diverse services. If the auditor is a grid service, service instances can be used to distribute the processing of SLA evaluations.

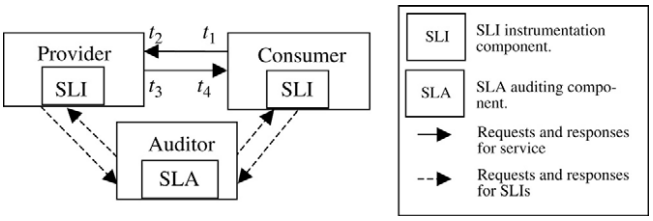

Fig. 1. Auditor evaluates the SLA from SLIs provided by consumer and provider.

Naturally, this solution requires the auditor to trust both the provider and the consumer and is thus not appropriate for the general case we investigate here. We present this architecture as a base solution against which more sophisticated architectures are compared. Observe also that this solution requires the services themselves to provide the SLIs to the auditor, requiring changes in the original consumer and provider code.

4.2. Packet sniffing architecture

A second architecture uses trusted sniffers in the provider and consumer Local Area Networks (LANs) in order to passively capture packets by means of, say, a port mirroring technique. Through this technique, packets coming to LAN switches addressed to the provider or consumer are copied to the sniffers' ports configured to work in promiscuous mode in order to receive packets not addressed to them. This architecture delegates the instrumentation function to a support service, called the inspector, which calculates SLIs based on sniffer information. Therefore the auditor requests SLIs from the inspector (see Fig. 2) rather than from the consumer or provider. In this solution all requests come from a real consumer; sniffers do not introduce new requests and thus do not affect response time to requests; furthermore, the inspector can calculate either provider SLIs or consumer SLIs from sniffer information.

This solution assumes that the sniffers are handed by the inspector to the consumer and provider. Moreover, it must be impossible for both provider and consumer to tamper with the sniffers, which requires hardware support [16] such as tamper-resistant sniffers or the installation of a secure coprocessor on regular sniffers. A secure coprocessor [17,19] provides a tamper-resistant core on which one can store cryptographic keys, process cryptographic algorithms, and check the overall compute system for changes and tampering. It thus can be used to add tamper-resistance to conventional computing systems. The secure coprocessor would also encrypt and digitally sign the sniffer information sent to the inspector, so that the calculated SLIs can be trusted.

A drawback of this solution is that sniffers may not be able to capture enough information needed to calculate SLIs if the packets are encrypted, a common situation in inter-domain

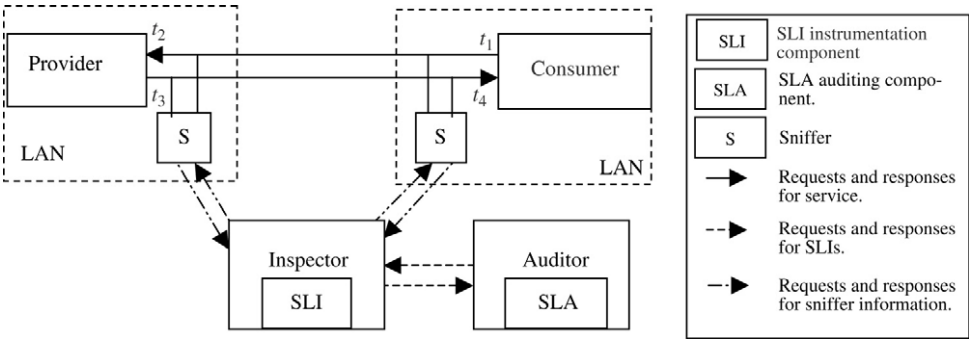

Fig. 2. Auditor evaluates the SLA from SLIs provided by the inspector, which calculates them from sniffer information. Sniffers are represented in the figure by S.

communication. When packets are encrypted, it becomes very hard to infer which packets form a service message. This problem may be avoided in the other architectures because they use service-level instrumentation and thus can see all information needed to measure SLIs. Another drawback is that the consumer and the provider must blindly trust the inspector. This is so because the packet sniffers may capture any traffic in the LAN and can therefore access sensitive information that is not related to the SLA being audited. The consumer and the provider must trust that the inspector will not do so.

4.3. Host decorator architecture

A third architecture has inspectors residing in the provider and in consumer hosts (see Fig. 3). The inspectors implement the provider interface. In fact, inspectors act as decorators to the provider. (A *decorator* is a well-known design pattern based on object composition that is used to add functionality to a particular object as opposed to a class of objects [9].) The consumer requests service from the inspector in its own host, which forwards requests to the inspector in the provider host, which then forwards them to the provider.

Since the inspectors participate in communication at the service level, they may overcome the cryptography limitation seen in the packet sniffing architecture. On the other hand, this may expose sensitive service-related information to the inspectors. A solution would be to encrypt sensitive information with the provider's public key, sending such sensitive information as an attribute of the whole message. In principle, encrypting request/response information may preclude the inspector from evaluating the SLA. However, most SLAs do not seem to care about application data; they typically use response time and availability as SLIs. Therefore, this solution allows for the safe transfer of sensitive data and auditing in many situations.

As in the packet sniffing architecture, key elements (in this case, the inspectors) are placed in provider and consumer sites. As such, inspectors must be made tamper-resistant via, for example, the installation of secure coprocessors [17,19].

4.4. Independent inspector architecture

A fourth architecture deals with the trust issue without burdening the services or its hosts with instrumentation code.

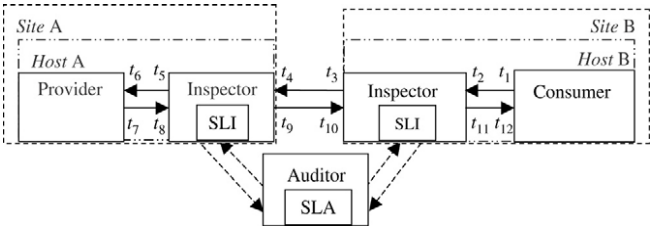

Fig. 3. Auditor evaluates the SLA from SLIs provided by inspectors, which are located in provider and consumer hosts.

It delegates the instrumentation function to a third party inspector, located in a different host (see Fig. 4(A)). This service calculates service SLIs by sending extra requests to the provider and hands them to the auditor, which performs the SLA evaluation. Since an inspector is a consumer to the provider, it needs to know the provider interface. Thus, each provider must have its own inspector that knows how to calculate its SLIs.

A drawback of this approach is the extra requests sent by the inspector to the provider. These extra requests can cause undesirable side effects (such as modifying a provider database) and impose extra load, reducing provider performance. Another problem with this solution is that the provider may be able to identify the requests coming from the inspector and give them preferential treatment. A possible solution for preferential treatment consists of using anonymizer techniques – as described in [18] – to conceal the network address of both consumer and inspector. But extra care should be taken to avoid service-level information giving away which requests come from the inspector (e.g. requests submitted on behalf of a "test" user).

Another point to notice is that the inspector does not measure the consumer SLIs, such as load submitted by the consumer, and thus cannot audit them. A work-around would be to embody the right to make a request in a digital ticket issued by the auditor (see Fig. 4(B)). Here, the auditor digitally signs a number of tickets T and hands them to the consumer. The provider only answers a request if it has a ticket digitally signed by the auditor. The tickets have a useful lifetime to avoid having the consumer accumulate them and overload the provider with requests. Ticket lifetime is defined in the SLA. The consumer asks the auditor for more tickets on an as-needed basis. When receiving a request, the provider verifies that the ticket is signed by the auditor and that the ticket did not timeout. Since the

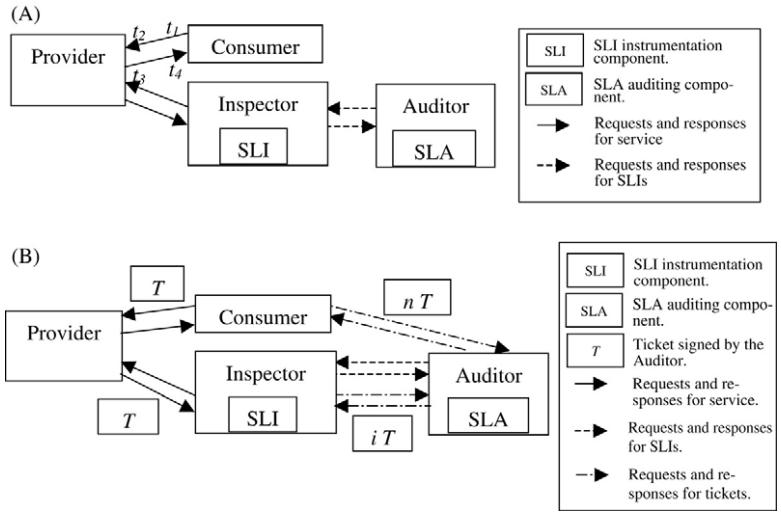

Fig. 4. (A) Auditor evaluates the SLA from SLIs provided by inspector and calculated through extra requests. (B) In order to audit consumer-related SLIs, the auditor can provider the consumer with signed tickets that are required by the provider to deliver service.

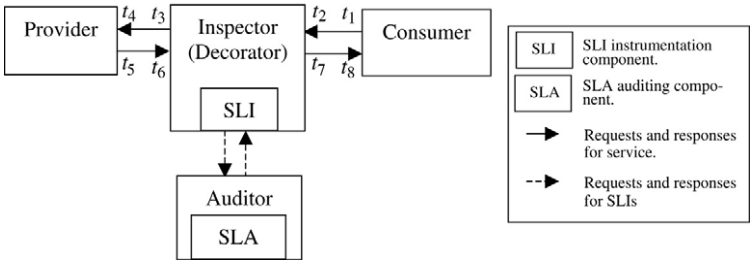

Fig. 5. Auditor evaluates the SLA from SLIs provided by inspector working as a decorator to the provider and calculated without extra requests.

inspector works as a consumer, it also obtains tickets from the auditor.

4.5. External decorator architecture

A fifth architecture can solve some of the problems outlined above. This architecture has the inspector working as a decorator to the provider, but outside both consumer and provider sites (see Fig. 5). In this way, all requests coming from the consumer pass through the inspector before arriving at the provider. The inspector needs to implement the same interface as the provider in addition to a management interface, which provides the SLI values. Inside the inspector, the methods that calculate the SLI values are called before and/or after the calls to the provider. The inspector forwards consumer requests, measures the load submitted by the consumer to the provider (or anything else of interest in the consumer data) and calculates SLIs obtained from the provider side. It is interesting to note that HP's OpenView SOA Manager uses this architecture [20].

This solution has several advantages over previous attempts. Among them, there are no additional requests generated by the inspector; that is, all requests made to provider are requests coming from a real consumer. As such, no additional load is imposed on the provider and no side effects (e.g. database modifications) need be worried about. The provider cannot identify inspector requests and give them preferential treatment because all requests come from consumers through the inspector. Another problem solved by this architecture is the measurement of the load submitted by the consumer, which may also be restricted by SLA clauses. Since the inspector forwards all consumer requests, it can audit them by counting.

As with all instrumentation and monitoring solutions, however, this solution imposes costs for the services. One cost is performance loss due to the addition of an inspector. However, since requests from different consumer are independent, the performance loss can be reduced if many distributed sites are used for inspectors and consumers are allowed to use the closest inspector. Such a strategy has been very successful in building highly scalable content distribution networks [8]. The downside, clearly, is that several inspectors have an additional cost of deployment, maintenance and administration. Another cost is an error in the SLI measurement caused by the instrumentation process itself. These costs are greater when the inspector is introduced at a distant site since requests that could use a shorter route will need to access the inspector for the purpose of SLA auditing.

4.6. External decorator with bypass architecture

An alternative to the previous architecture is to make some requests from the consumer go directly to the provider and some requests go through the inspector (See Fig. 6). This alternative tries to reduce the performance impact caused when all requests are forwarded to the inspector. While this solution

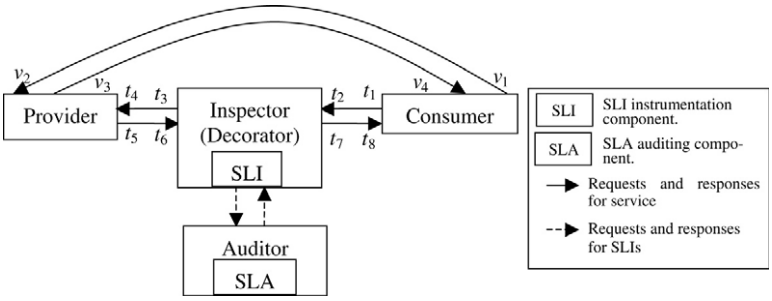

Fig. 6. Auditor evaluates the SLA from SLIs provided by inspector working as a decorator to the provider. Some requests go to provider directly to reduce the performance impact.

Table 1
Comparing the characteristics of the six architectures

	Trust	Intrusiveness	Additional requests	Preferential treatment	Consumer load	Encrypted messages
1. Naive	Yes	Code	No	No	Yes	Yes
2. Packet sniffing	No	Hardware	No	No	Yes	No
3. Host decorators	No	Hardware	No	No	Yes	Yes
4. Independent inspect.	No	No	Yes	Yes	Ticket use	Yes
5. External decorator	No	No	No	No	Yes	Yes
6. Ext. decor. w bypass	No	No	No	Yes	Ticket use	Yes

may improve the performance, it suffers from many problems outlined earlier; for example, the provider can give preferential treatment to the requests coming through the inspector; it also brings back the difficulty of auditing the requests submitted by the consumer, since the inspector does not see all requests.

4.7. Summary of the qualitative evaluation

Table 1 summarizes the qualitative advantages and drawbacks of each architecture, providing an overall comparison to the reader. A quantitative comparison will be undertaken in the next section. As discussed above, the features explored in the evaluation are:

- *Trust*: whether or not trust is necessary between the parties signing the SLA.
- *Intrusiveness*: the need to instrument the service code or to include hardware in the hosts of the parties signing the SLA.
- *Additional requests*: whether or not the architecture uses additional requests coming from a fake consumer (inspector) to perform auditing; this increases load and may produce undesirable effects in the service (inclusion of fake data in the service database, for example).
- *Preferential treatment*: whether or not the provider can identify the requests coming from an inspector and therefore give preferential treatment to these requests, hampering requests coming from real consumers.
- *Consumer load*: whether or not the architecture can measure the consumer load, that is, the submitted request rate.
- *Encrypted messages*: whether or not the architecture can perform auditing when the communication between provider and consumer is encrypted; naturally, SLAs that refer to service attributes will require them to be exposed to enable SLA evaluation.

5. Performance analysis

In this section we are concerned with the performance behavior of the architectures presented previously. That is, we are interested in analyzing how much performance the provider loses due to SLA auditing, as seen by the consumer. In general terms, two factors decrease the performance of the services in an auditing process. The first is what we term the Measurement Interference Error (MIE), which is the error introduced in the measured value due to the measurement process itself. For example, in the external decorator architecture, the calculated response time SLI typically diverges from the value of the response time obtained had auditing not been performed, because the request has to go through by the inspector. Another factor is due to *additional* requests processed by the provider in order to calculate SLI values. The greater the sample size of additional requests, the larger the intrusive effect on the performance of the service. Naturally, one can use statistical techniques to estimate the smallest sample size that assures the desired confidence interval and confidence level in the measurements [4]. Nevertheless, any number of additional requests will affect performance.

These two factors present themselves differently under the various architectures. In the *naive architecture*, in which an auditor obtains SLI values from the parties, no extra requests are performed and the sample size factor does not apply. On the other hand, some additional processing will need to be performed by the provider in order to periodically provide SLI values to the auditor. The MIE in the first architecture, e_1, is considered to be negligible because the processing time spent in performing an SLI measurement is typically very much smaller than the processing time spent in performing service requests. Experimental measurements confirm this, as we shall see in Section 5.2.

In the *packet sniffing architecture*, which uses sniffers to passively capture packets, sampling is not required since all requests come from a real consumer and there is no MIE ($e_2 = 0$), since the performance of the service is not affected by auditing. Although cryptography is needed to digitally sign and hide sniffer information, it does not influence request response time, since it will only be used when the inspector requests information to calculate SLI values.

In the *host decorator architecture*, which has an inspector in the consumer and provider hosts, sampling is not necessary since there are no additional requests. The MIE factor e_3 is present due to the delay spent during cryptography and to digital sign the information when there is no trust between provider and consumer.

In the *independent inspector architecture*, sample size is important, since additional service requests are performed for auditing purposes. The additional load imposed on the provider by these requests will affect the performance seen by consumers, and the SLI values calculated will therefore include an MIE, e_4. This value is larger than e_1 since the load imposed by business requests is typically much larger than the load imposed by a few requests for SLI values.

In the *external decorator architecture*, where the inspector works as a decorator to the provider, we only need worry about the MIE, e_5. Since the inspector will examine all requests sent to the provider, no sampling is needed. The MIE e_5 is different from e_4 because it is not caused by extra requests. The SLI values seen by consumers may be substantially different because requests must go through the inspector. However this error may be substantially reduced if we consider that there may be many inspectors available, distributed by locality, for example.

Finally, in the *external decorator with bypass architecture* both factors matter. Since only a sample of requests is audited through the inspector, we can choose the sample size to reduce the performance impact. We can also analyze the MIE e_6, which, like e_5, is caused by the indirection suffered by the requests. However e_6 is only a fraction of e_5 because the indirection is only suffered by a sample of requests, that is, $e_6 = \frac{n}{N} e_5$, where n is the sample size (determined to ensure given confidence interval and confidence level [4]) and N is the population size (the total number of requests performed over the observation period).

5.1. Analytical evaluation

The Measurement Interference Error (MIE) is the difference caused in a measured value due to the observation method used. In measuring a parameter value, an instrument is introduced to make the measurement. The instrument itself changes the observed value. This error affects the architectures described previously in various ways.

In order to analyze the MIE in each architecture, timestamps and time intervals are defined; the response time in each auditing architecture is calculated and is compared with the response time of the *base architecture*, i.e. when the consumer–provider interaction is performed without auditing (see Fig. 7).

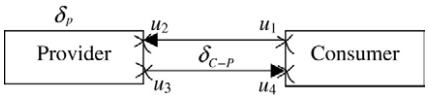

Fig. 7. The base architecture: Consumer–provider communication without auditing.

The timestamps u_1, u_2, u_3, u_4 and the time intervals δ_p, δ_{C-P} are of interest here. The interval δ_p represents the request processing time spent in the provider and is calculated as the difference between the time it answers a request and the time it receives this request, that is, $\delta_p = u_3 - u_2$. Since this time is load dependent, any additional load imposed by the auditing process will have to be carefully taken into account. The interval δ_{C-P} represents the total round-trip network delay between consumer and provider. Since time is measured at the web service application level, the time interval δ_{C-P} is based on timestamps obtained at this level. The time spent in the protocol levels under the application level is thus contained in the network delay.

In the base architecture, when the consumer sends a request directly to the provider at time u_1, the time at which the consumer receives the answer is:

$$u_4 = u_1 + \delta_p + \delta_{C-P}.$$

We now proceed to analyze the auditing architectures presented previously.

In the *naive architecture*, the timestamps are t_1, t_2, t_3, t_4 (see Fig. 1). The time intervals are $\delta_p, \delta_{C-P}, \delta_{\text{SLI}}$, where the interval δ_{SLI} is the delay due to SLI instrumentation. The time at which the consumer receives the answer to its request is:

$$t_4 = t_1 + \delta_{\text{SLI}} + \delta_p + \delta_{C-P}.$$

The MIE e_1 is thus the difference between the time spent by the request in naive architecture and the time spent in the base. Since $t_1 = u_1$, we have:

$$e_1 = \delta_{\text{SLI}}.$$

In the *packet sniffing architecture*, the timestamps are t_1, t_2, t_3, t_4 (see Fig. 2) and the time intervals are δ_p, δ_{C-P}. The time at which the consumer receives the answer to its request is:

$$t_4 = t_1 + \delta_p + \delta_{C-P}.$$

The MIE e_2 is the difference between the time spent by the request in this architecture and the time spent in the base architecture. So:

$$e_2 = 0.$$

In the *host decorator architecture*, t_1 to t_{12} are the timestamps (see Fig. 3) and the time intervals are δ_p, δ_{C-I}, δ_{I-I}, δ_{I-P}, δ_f, δ_r, δ_{crypt}. The interval δ_{crypt} is the total time spent on the forward and reverse paths to encrypt SLI information when there is no trust between provider and

consumer. The intervals $\delta_{C-I}, \delta_{I-I}, \delta_{I-P}$ are respectively the total network delay between the consumer and its inspector, both inspectors, and the provider's inspector and the provider itself. The interval δ_f is the time spent by an inspector to forward the request, whereas the time interval δ_r is the time spent by an inspector to return the answer. Consequently, the time at which the consumer receives the answer to its request is:

$$t_{12} = t_1 + \delta_{C-I} + 2\delta_f + 2\delta_{\text{crypt}} + \delta_{I-I} + \delta_{I-P} + \delta_p + 2\delta_r.$$

The delays δ_{C-I} and δ_{I-P} are the network delays (δ_N) together with the time spent in the web service protocol stack (δ_{WS}), so:

$$\delta_{C-I} = \delta_{N(C-I)} + \delta_{\text{WS}}$$
$$\delta_{I-P} = \delta_{N(I-P)} + \delta_{\text{WS}}.$$

Since an inspector is located in the consumer host and the other is in the provider host, the network delay between an inspector and the consumer or between an inspector and the provider located in the same host is negligible, that is, $\delta_{N(C-I)} = \delta_{N(I-P)} = 0$. So, the previous expression becomes:

$$t_{12} = t_1 + 2\delta_{\text{WS}} + 2\delta_f + 2\delta_{\text{crypt}} + \delta_{I-I} + \delta_p + 2\delta_r.$$

The MIE e_3 is the difference between the time spent by the request in this architecture and the time spent in the base architecture. Since the inspectors are located in consumer and provider hosts, we consider that the network delay between the inspectors in this architecture is similar to the network delay between consumer and provider in the base architecture, that is, $\delta_{I-I} = \delta_{C-P}$. We thus get:

$$e_3 = 2\delta_{\text{WS}} + 2\delta_f + 2\delta_{\text{crypt}} + 2\delta_r.$$

In the *independent inspector architecture*, the timestamps are t_1, t_2, t_3, t_4 (see Fig. 4). The time intervals are $\delta_p, \delta_{C-P}, \delta_m$, where δ_m is the marginal delay, which is the increase in processing time in the provider due to the additional requests sent by an inspector.

The time at which the consumer receives the answer to its request is:

$$t_4 = t_1 + \delta_p + \delta_{C-P} + \delta_m.$$

The MIE e_4 is the difference between the time spent by the request in this architecture and the time spent in the base architecture. So:

$$e_4 = \delta_m.$$

The marginal delay δ_m is the increase in processing time δ_p due to the addition of requests sent by the inspector, that is, it is the difference in the mean processing time in adding an inspector. Here λ_c is the mean rate of requests sent by all consumers to the provider and λ_i is the mean rate of requests sent by the inspector to the provider.

$$\delta_m = \delta_p(\lambda_c + \lambda_i) - \delta_p(\lambda_c).$$

In order to analyze the behavior of δ_m, we use a queuing model in which the server represents the provider service taken as a black box (see Fig. 8). The consumer and inspector requests

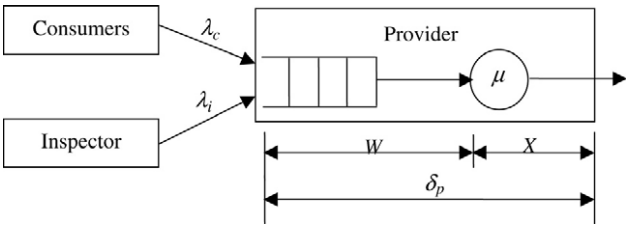

Fig. 8. Queuing model.

are the clients reaching the provider, where they queue waiting for service. The average time δ_p spent by a request in the provider is the sum of the average time spent in the queue W and the average service time X.

Here, λ_c and λ_i are the mean rates of requests sent by consumers and by the inspector, respectively, and are expressed in requests per time unit. The value μ is the mean rate of service and can also be expressed in requests per time unit. We consider an M/G/1 queue in which the inter-arrival times are exponentially distributed, the service times have a general distribution and there is a single server [12], where the mean service time is $\overline{X} = \frac{1}{\mu}$. The service times follow a distribution with average $\overline{X}$ and second moment $\overline{X^2}$. The mean time spent by a request in the provider $\delta_p(\lambda)$ can be calculated by the Pollaczek–Khinchin formula:

$$\delta_p(\lambda) = \overline{X} + \frac{\lambda \overline{X^2}}{2(1-\rho)}.$$

When the total rate of requests is λ_c, we have:

$$\delta_p(\lambda_c) = \overline{X} + \frac{\lambda_c \overline{X^2}}{2(1-\rho_c)} \quad \text{and} \quad \rho_c = \lambda_c \overline{X}.$$

When the total rate of requests is $\lambda_c + \lambda_i$, we have:

$$\delta_P(\lambda_c + \lambda_i) = \overline{X} + \frac{(\lambda_c + \lambda_i)\overline{X^2}}{2(1-\rho_{ci})} \quad \text{and} \quad \rho_{ci} = (\lambda_c + \lambda_i)\overline{X}.$$

Here ρ_c and ρ_{ci} are the server utilization and vary from 0 to 1. In the particular case where service times are constants (a close approximation to reality for transaction services), the second moment of service time $\overline{X^2}$ is $\overline{X}^2 = \frac{1}{\mu^2}$. We therefore have:

$$e_4 = \delta_m = \delta_P(\lambda_c + \lambda_i) - \delta_P(\lambda_c)$$
$$= \frac{(\lambda_c + \lambda_i)}{2\mu^2(1-\rho_{ci})} - \frac{\lambda_c}{2\mu^2(1-\rho_c)}.$$

In the *external decorator architecture*, the timestamps are t_1 to t_8 (see Fig. 5). The time intervals are $\delta_p, \delta_{C-I}, \delta_{I-P}, \delta_f, \delta_r$. The interval δ_{C-I} is the total network delay between the consumer and the inspector and δ_{I-P} is the total network delay between the inspector and the provider. The interval δ_f is the time spent by the inspector to forward the request to the provider and is $\delta_f = t_3 - t_2$. The time interval δ_r is the time spent by the inspector to return the answer to the consumer and is $\delta_r = t_7 - t_6$. The time at which the consumer receives the answer to its request is therefore:

$$t_8 = t_1 + \delta_f + \delta_p + \delta_r + \delta_{C-I} + \delta_{I-P}.$$

The MIE e_5 is the difference between the time spent by the request in this architecture and the time spent in the base architecture. So:

$$e_5 = \delta_f + \delta_r + \delta_{C-I} + \delta_{I-P} - \delta_{C-P}.$$

As shown in Fig. 6, in the *external decorator with bypass architecture*, the timestamps are t_1 to t_8 (measured during the sample traffic of n requests) and v_1, v_2, v_3, v_4 (measured during the traffic of the rest of the $N-n$ requests). As we have already shown the measurement interference error for this architecture is:

$$e_6 = \frac{n}{N} e_5.$$

And hence:

$$e_6 = \frac{n}{N}\left(\delta_f + \delta_r + \delta_{C-I} + \delta_{I-P} - \delta_{C-P}\right).$$

5.2. Experimental evaluation

This section makes the analytical results more concrete by gauging the performance penalty of auditing architectures in a real environment. We have implemented a bookstore service that can include and exclude books from the purchase list. The purchase list is kept in memory (we avoided a database because its cache would increase the variability of the measurements). Services were implemented under Globus 3.2.1 [10] deployed on several Pentium 4s, 2.66 GHz of CPU, 640 MB of RAM. Globus 3.2.1 was chosen as programming environment because it offers APIs to implement both web services and grid services.

We first validate our analysis by measuring the MIE introduced by the naive architecture and the external decorator architecture in a 100 Mbps switched Ethernet, and comparing the obtained values against the MIE predicted by the analysis. We chose these architectures for the simplicity in instrumenting them, whereas the environment was chosen for its ease of control. Recall that we need δ_{SLI}, δ_f, δ_r, δ_{C-P}, δ_{C-I}, and δ_{I-P} in order to evaluate the naive architecture MIE e_1 and the external decorator architecture MIE e_5. Table 2 depicts these values (δ_{C-P}, δ_{C-I} and δ_{I-P} have the same value because consumers, providers and inspectors were running on identical machines connected through the same networking).

Table 3 then contrasts the measured MIE against the expected MIE. As one can observe, the differences between measured MIEs and those predicted by the analysis are negligible.

The methodology used to perform the experiments and obtain numerical values can be summarized as follows:

- All programs were written in Java and measures obtained with 1 μs resolution. Since Java only supports measurements with a 1 ms resolution, time intervals were obtained by timestamping a loop performing a particular task 1000 times and then factoring out the loop overhead.
- To calculate the time spent in the forward (δ_f) and in the return of the requisition (δ_r) by the inspector, we implemented an inspector service working as a provider's decorator. The return delay δ_r is slightly greater than the forward delay δ_f in this case because the SLI is calculated immediately after of the provider method call, as part of the return procedure.

Table 2
Measured values for the parameters needed to compute e_1 and e_5

Measured parameter	Value (ms)
δ_{SLI}	0.007
δ_f	0.477
δ_r	0.482
$\delta_{C-P}, \delta_{C-I}, \delta_{I-P}$	5.124

Table 3
Comparing measured and predicted MIE

Architectures	Response time (ms)	Observed MIE (ms)	Predicted MIE (ms)
0. Base	8.116	N/A	N/A
1. Naive	8.120	0.004	0.007
5. External decorator	14.300	6.184	6.083

Table 4
Measured values for δ_{crypt} and δ_{WS}

Measured parameter	Value (ms)
δ_{crypt}	130.000
δ_{WS}	5.016

- The values obtained for each parameter are mean values calculated from samples collected in groups of 20, confidence level of 95%, confidence interval of 5%.
- The first four samples were consistently eliminated throughout the experiments. The reason is that, when using languages such as Java, class loading issues make things slower at first; also, in general, any dynamic system exhibits transients that should be removed when calculating long-term averages.

Naturally, one can expect to see considerable variation of MIE depending on computer and network speed, as well as the service time itself. Service times affect the MIE of the independent inspector architecture, whereas network speed affects both external decorator and external decorator with bypass. Likewise, the architectures that depend on sampling (independent inspector and external decorator with bypass) are going to be affected by the SLI variance (the greater the variance, the greater the number of samples needed to achieve a given confidence interval). In order to give the reader a broader view of the possibilities, we calculated the MIE with each of these parameters (service time, network speed, and sample size) assuming "low" and "high" values. More precisely, we assume a low service time $\overline{X} = 10$ ms and a high service time $\overline{X} = 1000$ ms; low network delay $\delta_N = 1$ ms (typical of a LAN) and a high network delay $\delta_N = 200$ ms (typical of a WAN); and low sample size $\frac{n}{N} = 5\%$ and a high sample size $\frac{n}{N} = 30\%$. Also, for the independent inspector architecture, we assumed a reasonably loaded server, with an arrival rate of half of the service rate ($\lambda = \frac{\mu}{2}$), a typical value for servers that are not lightly loaded but have not yet reached saturation.

Besides these assumptions and values presented in Table 2, we also measured δ_{crypt} and δ_{WS} to be able to compute the MIE formula of the host decorator architecture. The result

Table 5
Analyzing the measurement interference error of the architectures

Architectures		Small sample (5%) (ms)	Large sample (30%) (ms)
1. Naive		0.007	0.007
2. Packet sniffing		0.000	0.000
3. Host decorators		271.950	271.950
4. Independent inspector	Fast provider (10 ms)	0.526	4.285
	Slow provider (1000 ms)	52.631	428.572
5. External decorator	LAN (1 ms)	6.975	6.975
	WAN (200 ms)	205.975	205.975
6. Ext decorator with bypass	LAN (1 ms)	0.349	2.093
	WAN (200 ms)	10.299	61.792

is presented by Table 4. To calculate the value for δ_{crypt}, a Globus service was implemented with and without security. The secure service was implemented using encryption as the Grid Security Infrastructure (GSI) security level. Response time was measured for a client using both services and the value of δ_{crypt} is simply the difference between the two mean values. In order to measure δ_{WS}, a ping web service (WS-ping) was implemented. This is essentially the same as a ping command but goes through web service interfaces and protocol stack. By subtracting the time of a common ping command (network time) from the WS-ping time, one gets the time spent in the web services protocol stack.

Table 5 joins all results, summarizing the expected MIE for different conditions and giving the reader an overall idea of how the architectures behave on different points of the design space. One can see that the MIE for the naive and packet sniffing architectures is negligible. The host decorator architecture presents somewhat larger MIE, which is due to the need for two rounds of cryptography in both inspectors it uses. The independent inspector architecture incurs the highest overhead of the study when it demands a large sample from a slow provider. However, if the sample size is small and (especially) if the provider is fast, the architecture performs well. The MIE of the external decorator architecture depends on the proximity between the service endpoints. Therefore, it performs much better in a LAN than in a WAN. Also as expected, using bypass further reduces its MIE.

6. Conclusions

This paper addresses the auditing of service level agreements for web and grid services. In particular, we focus on the case where provider and consumer do not blindly trust each other, which is envisioned to be the common case for large grid deployments. We evaluated six possible architectures for the auditing process, showing their advantages, drawbacks and analyzing their performance.

The naive architecture was provided as a baseline, as it does assume trust between provider and consumer. Moreover, it requires changes in the provider and consumer code. Such intrusive changes on the provider and consumer code can be avoided by using the host decorator architecture, at the expense of adding approximately 300 ms of performance penalty (on current typical computing environments).

When there is no trust between the signing parties, the naive architecture is not applicable. The Packet sniffing and the host decorator architectures require the installation of special hardware in both provider and consumer, which is likely to be a great obstacle for wide deployment. The independent inspector architecture is very attractive because it does not interpose anything in the real consumer-to-provider service path. On the other hand, it does create problems related to the additional load imposed on the provider, side-effects on the provider database, and possible preferential treatment given to the requests that are used to evaluate the SLA. The external decorator architecture is immune to all these problems, but it may impose a heavy performance penalty depending on where (on the network) consumer, provider and inspectors are. The external decorator with bypass architecture attempts to address this potential performance problem, but reintroduces (on a smaller scale) the problems faced by the independent inspector architecture.

Therefore, there is no “best” solution for all scenarios. However, we believe that the vast majority of service auditing needs could be catered for by a company that deploys a widely distributed external decorator architecture. A wide multi-site deployment (as done, for example, by Akamai and Google) of the external decorator architecture could allow the auditor to place the inspector near either the consumer or the provider, therefore minimizing the performance impact of auditing. Since the single issue with the external decorator architecture is performance, this approach can possibly create a very competitive solution.

Acknowledgments

We are grateful to the OurGrid research team for the many fruitful conversations about the issues involved in grid research in general and management research in particular. We are also grateful to Chico Souza and Antônio Silva, Statistics professors who helped with the performance analysis. Our gratitude also goes to Katia Saikoski for her helpful comments. Thanks also to the anonymous reviewers who greatly helped to improve the quality and presentation of the paper. This work was partially developed in collaboration with HP-Brazil R&D and partially funded by CNPq/Brazil grant 302317/2003-1.

References

[1] R. Al-Ali, O. Rana, D. Walker, G-QoSM: Grid service discovery using QoS properties, in: Grid Computing, Computing and Informatics Journal 21 (5) (2002) (special issue).

[2] A.C. Barbosa, J. Sauvé, W. Cirne, M. Carelli, Independently auditing service level agreements in the grid. in: Proceedings of the 11th HP OpenView University Association Workshop, HPOVUA 2004, June 2004.
[3] C. Bourasa, M. Campanella et al., QoS and SLA aspects across multiple management domains: The SEQUIN approach, Future Generation Computer Systems 19 (2) (2003) 313–326.
[4] J. Devore, Probability and Statistics for Engineering and the Sciences, fourth edn, Wadsworth Publishing Company, 1995.
[5] A. Keller, H. Ludwig, The WSLA framework: Specifying and monitoring service level agreements for web services, in: E-Business Management, Journal of Network and Systems Management 11 (1) (2003) (special issue). Plenum Publishing Corporation.
[6] I. Foster, The Grid: Computing Without Bounds, Scientific American, April 2003.
[7] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, A distributed resource management architecture that supports advance reservations and co-allocation, in: International Workshop on Quality of service, 1999.
[8] S. Gadde, J. Chase, M. Rabinovich, Web caching and content distribution: A view from the interior, in: Proc. of the Fifth Int. Web Caching and Content Delivery Workshop, May 2000.
[9] E. Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, ISBN: 0201633612, 1995.
[10] The Globus Alliance site, http://www.globus.org/.
[11] S. Khana, K.F. Li et al., Optimal Quality of service routing and admission control using the Utility Model, Future Generation Computer Systems 19 (7) (2003) 1063–1073.
[12] L. Kleinrock, Queueing Systems, Vol I: Theory, Wiley, New York, 1975.
[13] OASIS. OASIS web services Distributed Management (WSDM) Technical Committee web Page, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
[14] The OurGrid site, http://www.ourgrid.org.
[15] A. Sahai, A. Durante, V. Machiraju, Towards automated SLA management for web services, Research Report HPL-2001-310 (R.1), Hewlett-Packard (HP) Labs Palo Alto, July 26, 2002. http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf.
[16] T. Sander, C. Tschudin, Protecting mobile agents against malicious hosts, in: G. Vigna (Ed.), Mobile Agent Security, in: LNCS, February 1998.
[17] S. Smith, S. Weingart, Building a high-performance, programmable secure coprocessor, IBM Research Report RC 21102, February 1998.
[18] F. Syverson, M. Goldschlag, G. Reed, Anonymous connections and onion routing, in: Proc. IEEE Symposium on Security and Privacy, Oakland, May 1997.
[19] B. Yee, Using secure coprocessors, Ph.D. Thesis, Carnegie Mellon University, 1994.
[20] HP OpenView SOA Manager site, http://www.managementsoftware.hp.com/products/soa/index.html.

Ana Carolina Barbosa holds an M.Sc. degree in Informatics from the Universidade Federal de Campina Grande (UFCG), Brazil. Her thesis is concerned with auditing of service level agreements for grid services.

She was researcher on the OurGrid Project in the Distributed System Laboratory at the UFCG and in the Technology and Music Group at the Universidade Federal da Paraíba, Brazil.

Now, she is an information analyst for the Brazilian Government. Her interests are in the areas of network management, system development, computational grids, computational system security and auditing.

Dr. Jacques Sauvé received a Ph.D. in Electrical Engineering at the University of Waterloo, Canada. He is a consultant in several technology fields, with special emphasis in Computer Networks and Business-Driven IT Management. He was Vice-President of Development for Light-Infocon for several years. He is currently professor in the Computer Science Department at the Federal University of Campina Grande, Brazil, where his efforts are concentrated in the areas of advanced architectures for Information Systems and IT Management. Dr Sauvé has published 10 books and many papers in international journals and conferences.

Walfredo Cirne is faculty at the Computer Science Department of the Universidade Federal de Campina Grande, in Brazil. Dr. Cirne holds a Ph.D. from the University of California San Diego, in the USA. Since 1997, his research focuses on grid Computing. Before that, we worked on Computer Networks and Machine Learning. Currently, Dr. Cirne leads OurGrid, a project developed in cooperation with Hewlett Packard that aims to provide an open, free-to-join grid solution for bag-of-tasks applications. Further information and publications of Dr. Cirne can be found at http://walfredo.dsc.ufcg.edu.br/index_en.html.

Mirna Carelli is a Computer Science undergraduate student at Universidade Federal de Campina Grande. She has been working on the OurGrid Project at the Distributed Systems Lab since 2004 and her interest areas include service oriented architecture, artificial intelligence and software engineering.