UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Avaliando Arquiteturas de Auditoria de Acordos de Nível de Serviço para *Web Services* e *Grid Services*

Ana Carolina Benjamim Barbosa

Campina Grande – PB
Fevereiro - 2005

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Avaliando Arquiteturas de Auditoria de Acordos de Nível de Serviço para *Web Services* e *Grid Services*

Ana Carolina Benjamim Barbosa

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande - Campus I - como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Jacques Philippe Sauvé
Orientador

Campina Grande – PB
Fevereiro - 2005

BARBOSA, Ana Carolina Benjamim.
B238A

Avaliando Arquiteturas de Auditoria de Acordos de Nível de Serviço para *Web Services* e *Grid Services.*

Dissertação (Mestrado) – Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande – Paraíba, Fevereiro/2005.

81 p. Il.

Orientador: Jacques Philippe Sauvé

Palavras Chave:

1. Sistemas Distribuídos
2. Auditoria de serviços
3. Auditoria independente.

CDU-681.3.066D

Avaliando Arquiteturas de Auditoria de Acordos de Nível de Serviço para *Web Services* e *Grid Services*

Ana Carolina Benjamim Barbosa

Dissertação aprovada em 22 de fevereiro de 2005

Dr. Jacques Philippe Sauvé
Orientador

Dr. Walfredo da Costa Cirne Filho
Componente da Banca

Dr. Dorgival Olavo Guedes Neto
Componente da Banca

Campina Grande, 22 de fevereiro de 2005.

Agradecimentos

Gostaria de agradecer ao meu orientador Jacques, que foi sempre prestativo e compreensível, principalmente nos momentos em que mais precisei, pelos seus ensinamentos e atenção.

Ao professor Walfredo, por ter me convidado a participar do projeto OurGrid, do qual tenho muito orgulho de ter feito parte, pelas nossas discussões e amizade.

A Verlaynne e Mirna, por terem sido minhas companheiras de linha de pesquisa no projeto OurGrid e, principalmente, pela nossa amizade construída em cima de tantos momentos de pressão e de descontração.

Aos meus amigos do Laboratório de Sistemas Distribuídos, pelas trocas de idéias e pela alegria que sempre proporcionaram durante os dias e as noites de trabalho. Principalmente a Roberta, Raquel, Eliane, Esther, Ayla, Daniel Paranhos, Nazareno, Gustavo, Filipe, Loreno, Hugo, Glaucimar, Lauro, Randolph, Lívia, Elizeu e Raissa.

A Aninha, Keka, Zane, Marcelo e Vera pelo apoio e amizade.

A todos os professores que me ensinaram durante o curso.

Aos meus amigos do mestrado, pelos momentos de descontração e de trabalho. Principalmente a Luana, Rodrigo, David, Fabiana, Leidjane, Cidinha e Emerson.

À HP Brasil pelo suporte financeiro.

A meus pais maravilhosos, que sempre fizeram de tudo para me ajudar nos mínimos detalhes, e às minhas amadas irmãs.

A Robson, por seu carinho e companhia em todos os momentos.

Por fim, a Deus, a quem eu devo tudo.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Gerência de nível de serviço de *Web* e *Grid Services* é uma questão importante que precisa ser resolvida para se alcançar a implantação em larga escala de serviços. A necessidade de negociar, medir e auditar a qualidade dos serviços fornecidos (em termos, por exemplo, de desempenho, custo ou segurança) crescerá diretamente com a proliferação de serviços. A gerência de SLA (*Service Level Agreement*) compreende as fases de: (i) negociação e estabelecimento do SLA; (ii) auditoria de SLA (iii) notificação de violações do SLA para as partes envolvidas e interessadas; e (iv) a tomada de ações de gerência com o intuito de fazer correções no serviço para evitar novos descumprimentos ou aplicar penalidades aos serviços não cumpridores do SLA.

Embora existam trabalhos que abordem a automatização da gerência de SLAs para serviços, eles não detalham a fase de auditoria de SLAs. Esta dissertação foca a fase de auditoria de SLAs para *Web* e *Grid Services*.

São apresentadas sete arquiteturas possíveis para a realização do processo de auditoria, assim como uma avaliação qualitativa e uma avaliação quantitativa dessas arquiteturas. A avaliação qualitativa compara as arquiteturas com base em aspectos como: intrusividade, confiança necessária, uso de requisições extras, possibilidade de tratamento preferencial, auditoria de carga do consumidor, e auditoria de mensagens criptografadas. A avaliação quantitativa foca na perda de desempenho causada pela auditoria. Nessa avaliação, são descritos dois fatores que diminuem o desempenho dos serviços visto pelos clientes devido ao processo de auditoria. Um deles é o tamanho do conjunto de amostras das requisições extras obtidas pelo Provedor para calcular os valores dos SLIs (*Service Level Indicators*). Quanto maior o tamanho desse conjunto de amostras maior é o impacto no desempenho do serviço. Um outro fator é o Erro de Interferência de Medição (EIM), ou seja, o erro introduzido no valor medido devido ao próprio método de observação. Esses dois fatores são analisados para as sete arquiteturas apresentadas e os EIMs são analisados mais precisamente através de experimentos realizados com a implementação de algumas dessas arquiteturas.

Abstract

Service level management (SLM) of Web and Grid Services is an important issue that has to be solved in order to achieve large-scale deployment of services. The need to negotiate, measure and audit the quality of the services provided (in terms, for instance, of performance, cost or security) will increase directly with the proliferation of services. SLM comprises the following phases: (i) Service Level Agreement (SLA) negotiation and establishment; (ii) SLA auditing (iii) notification of SLA violations to the involved parties; and (iv) triggering of management actions in order to correct services to avoid new violations or to apply penalties to the assigning parties that did not comply with the SLA.

Although some research efforts address SLM automation to Grid and Web Services, they do not detail the SLA auditing process. This dissertation focuses on the SLA auditing phase.

We here present seven architectures that perform SLA auditing, as well as a qualitative and quantitative evaluation of such architectures. The qualitative evaluation compare the architectures based on aspects such as intrusiveness, trust, use of extra requests, possibility of preferential treatment, possibility of auditing of Consumer load, and possibility of auditing of encrypted messages. The quantitative evaluation focuses on the performance penalty that auditing introduces. In this evaluation, two factors that reduce service performance in the Client view due to auditing are described. One is the sample size of the extra requests sent by Inspector and obtained by the Provider in order to calculate the (Service Level Indicator) SLI values. The greater this sample size, the greater is the impact on service performance. Another factor is the Measurement Interference Error (MIE), that is, the error introduced in the measured value due to the observation method itself. These two factors are analyzed in the seven architectures and the MIEs are analyzed in more details by means of experiments performed with an implementation of some of these architectures.

Capítulo 1
Introdução

Este capítulo motiva a dissertação. Inicialmente, é apresentada a relevância do trabalho aqui apresentado. Em seguida, são mostrados os desafios para a auditoria de acordos de nível de serviço (SLAs) para *Web* e *Grid Services*. Depois são apresentados os objetivos a serem atingidos. O capítulo finaliza com uma descrição da estrutura dos outros capítulos desta dissertação.

1.1 Relevância

Por representarem uma nova tecnologia para aplicações distribuídas independente de plataforma e de linguagem e por fazerem uso de tecnologias padrão da Internet, os *Web Services* estão amadurecendo como uma tecnologia que permite a integração de aplicações pertencentes a diferentes negócios, possibilitando operações entre parceiros diferentes cada vez mais rápidas e eficientes.

Grid Services têm recentemente evoluído a partir de *Web Services* e da tecnologia *Grid* de alto-desempenho e prometem um nível de dinamismo de serviços sem precedentes. Na visão *Grid*, o relacionamento entre fornecedores e consumidores é bastante dinâmico e os serviços em um ambiente *Grid* são transientes, ou seja, possuem um tempo de vida.

Uma vez que aplicações integradas por *Web Services* ou *Grid Services* tipicamente abrangem múltiplos domínios administrativos e participam de uma arquitetura orientada a serviços, em que serviços se descobrem e se comunicam sob-demanda, Qualidade de Serviço (*Quality of Service, QoS*) se torna essencial para garantir que essas interações sejam realizadas com sucesso. Dessa forma, contratos chamados Acordos de Nível de Serviço (*Service Level Agreements, SLAs*) são usados para estabelecer requisitos de QoS, tais como parâmetros que indicam o que uma aplicação cliente pode esperar de uma aplicação provedora de serviços e parâmetros restringindo as requisições que um consumidor pode requerer de um provedor. Um SLA é composto por um conjunto de Objetivos de Nível de Serviço (*Service Level Objectives, SLOs*), que são avaliados usando dados mensuráveis, chamados Indicadores de Nível de Serviço (*Service Level Indicators, SLIs*).

Em contraste com o atual uso de SLAs pelas empresas, em que a especificação, monitoração e gerência de SLAs são realizadas de maneira bastante manual e através da intervenção humana, em um ambiente de serviços dinâmicos, a gerência de SLAs deve ser um processo automatizado e dinâmico. Esse processo é composto pelas fases de negociação, definição, auditoria, notificação de violações do SLA e tomada de ações de gerência quando é detectado o não cumprimento do SLA.

O trabalho desta dissertação considera que a fase de auditoria de SLAs apresenta pontos bastante interessantes a serem analisados e que não são tratados em detalhes por outros trabalhos relacionados, tais como [1], [2], [3] e [4], que abrangem tópicos relacionados à automatização de gerência de SLAs para serviços.

Como SLAs são simplesmente contratos, eles não fornecem fortes garantias por si só. Alguém precisa auditá-los para assegurar que eles sejam obedecidos. Essa é a fase de auditoria de SLAs. Nessa fase, supõe-se que o SLA já foi negociado e definido. Então, são realizadas medições de SLIs para calcular valores para os SLOs definidos no SLA (esse processo é aqui chamado de instrumentação) e, em seguida, os valores calculados são comparados com os valores definidos para os níveis de serviços (SLOs) estabelecidos no SLA (esta é a auditoria propriamente dita).

Como os serviços assinantes do SLA geralmente pertencem a diferentes entidades e podem não ter nenhum conhecimento uns sobre os outros até que necessitem de suas funcionalidades, pode ser que não exista uma relação implícita de confiança entre eles. Em particular, um consumidor pode suspeitar das informações de auditoria fornecidas pelo provedor e vice-versa. Nos casos em que não há confiança entre as partes assinantes do contrato, a auditoria poderia ser feita por uma terceira entidade independente. Nesta dissertação são tratadas questões sobre confiança entre os serviços, sobre onde e como instrumentá-los e fazer a avaliação dos seus SLAs, sobre o aumento no tempo de resposta visto pelo consumidor causado pela introdução do processo de auditoria de SLAs, entre outros aspectos que serão mais bem descritos no decorrer deste trabalho. Levando-se em consideração esses aspectos, são apresentadas algumas arquiteturas possíveis para o processo de auditoria de SLAs entre serviços, além de características, vantagens e desvantagens de cada uma dessas arquiteturas.

1.2 Desafios em Auditar SLAs entre *Web* e *Grid Services*

SLAs precisam ser auditados para dar garantias reais de QoS. Contudo, quando se pensa em auditar um SLA estabelecido entre serviços pertencentes a diversas entidades e sob

diferentes controles administrativos, encontra-se um problema de confiança. Quando os serviços possuem um forte relacionamento de confiança, eles podem acreditar nos SLIs fornecidos por cada um (tais como requisições do cliente por minuto e tempo de resposta do servidor). Infelizmente, em um cenário altamente dinâmico de interação entre serviços, relacionamentos fortes de confiança pré-estabelecidos são improváveis [5].

SLIs de um assinante podem também ser medidos pelo outro assinante. Por exemplo, o cliente pode medir tempo de resposta do serviço e o provedor pode computar a taxa de requisições do cliente. Porém, dessa maneira o problema da confiança não é resolvido. Se o cliente diz que o SLA não foi cumprido porque, por exemplo, o tempo de resposta foi alto demais, o servidor pode discordar apresentando suas próprias medições de tempo de resposta, menores [5].

Outra questão a ser pensada é que os serviços podem não querer modificar seus códigos para calcular SLIs para fins de auditoria, ou mesmo essa mudança pode não ser viável ou apropriada devido à dinamicidade das interações entre eles. Para driblar o problema da confiança e da intrusividade no código dos serviços, uma terceira entidade independente, um Auditor, poderia fazer a instrumentação e a avaliação do SLA. Para isso, tanto o servidor quanto o cliente precisam concordar com o uso do Auditor antecipadamente [5].

Contudo, essa proposta levanta a questão: como o Auditor pode obter SLIs confiáveis? Pedir os SLIs ao cliente e ao servidor implica nos problemas já descritos. Uma idéia é fazer uma entidade chamada Inspetor responsável por investigar o servidor como se ele fosse um cliente. A desvantagem é a sobrecarga gerada, ou seja, o Inspetor pode reduzir o desempenho do servidor fazendo requisições falsas de serviço para calcular os SLIs. Requisições falsas podem provocar efeitos colaterais no serviço, tais como inserção de dados falsos. Outro problema com essa abordagem é a possibilidade de o provedor identificar que a requisição veio do Inspetor e dar tratamento preferencial no atendimento dessa requisição. Além do mais, essa abordagem da investigação somente ajuda a obter SLIs relacionados ao servidor. Como obter SLIs relacionados ao cliente, como a carga submetida, continua em aberto [5].

Um outro desafio ao se auditar SLAs entre serviços consiste na alteração dos valores reais dos SLIs ao se incluir mecanismos para fazer a instrumentação e a auditoria, ou seja, o próprio processo de auditoria pode alterar os valores a serem observados durante o processo de interação entre os serviços assinantes. Essa diferença de valor causado nos SLIs pelo processo de auditoria é aqui denominada Erro de Interferência de Medição (EIM). Através do EIM pode-se analisar qual o aumento no tempo de resposta visto pelo consumidor devido ao processo de auditoria [5].

1.3 Objetivos

O objetivo geral desta dissertação é analisar o processo de auditoria de SLAs entre serviços. Para atingir o objetivo geral, seguem os seguintes objetivos específicos:

- Identificar e analisar algumas arquiteturas possíveis para o processo de auditoria de SLAs;
- Fazer uma avaliação qualitativa, comparando as arquiteturas em relação aos seguintes aspectos: intrusividade, confiança necessária, uso de requisições extras, possibilidade de tratamento preferencial, auditoria de carga do consumidor e auditoria de mensagens criptografadas.
- Fazer uma avaliação analítica sobre o desempenho das arquiteturas identificadas;
- Implementar algumas das arquiteturas com intuito de coletar valores para a análise experimental sobre desempenho;
- Fazer uma avaliação experimental sobre o desempenho das arquiteturas identificadas;

1.4 Estrutura da Dissertação

O texto desta dissertação está dividido em seis capítulos, incluindo esta introdução.

O capítulo 2 apresenta os fundamentos teóricos sobre os conceitos necessários para o bom entendimento da dissertação. Entre eles, são explicados os conceitos de: *Web Services*, *Grid Services*, Gerência de Nível de Serviço (SLM), Acordo de Nível de Serviço (SLA) e *Globus Toolkit*.

O capítulo 3 descreve os trabalhos relacionados, entre eles: WSMF (*Web Service Management Framework*), WSLA (*Web Servive Level Agreement*) *Framework*, G-QoSM (*Grid QoS Management Framework*), GARA (*Globus Architecture for Reservation and Allocation*). Esse capítulo traz também uma comparação desses com o trabalho apresentado nesta dissertação.

O capítulo 4 apresenta sete arquiteturas identificadas para o processo de auditoria de SLAs para serviços, assim como uma avaliação qualitativa das mesmas, mostrando suas vantagens e desvantagens. É apresentada uma comparação das arquiteturas em relação aos seguintes aspectos: intrusividade, confiança necessária, uso de requisições extras, possibilidade de tratamento preferencial, auditoria de carga do consumidor e auditoria de mensagens criptografadas.

O capítulo 5 traz uma análise de desempenho dessas arquiteturas. Primeiramente apresenta uma avaliação analítica, a qual inclui a análise de dois fatores que afetam o desempenho das arquiteturas apresentadas: o tamanho da amostra de SLIs coletada para auditoria e o erro de interferência de medição. Em seguida, faz uma avaliação experimental, a qual explica como foram realizados os experimentos e apresenta os resultados obtidos para a análise de desempenho das arquiteturas apresentadas.

O capítulo 6 apresenta as conclusões e os resultados obtidos com o trabalho realizado e traz propostas para trabalhos futuros.

Por fim, o apêndice A apresenta um artigo com os resultados desta dissertação, escrito pela autora da mesma, Ana Carolina Barbosa, e seus orientadores Jacques Sauvé e Walfredo Cirne. Esse artigo foi submetido para publicação no *Future Generation Computer Systems*, o Jornal Internacional de Computação *Grid*: Teoria, Métodos e Aplicação, em agosto de 2005.

Capítulo 2
Fundamentos

Aqui serão apresentados conceitos teóricos sobre os assuntos necessários para o bom entendimento da dissertação.

O objetivo desta seção é fornecer um melhor entendimento sobre os componentes individuais discutidos nesta dissertação. Esses componentes são: *Web Services*, *Grid Services*, Gerência de Nível de Serviço (SLM), Acordo de Nível de Serviço (SLA) e *Globus Toolkit*.

2.1 *Web Services*

Web Services são aplicações modulares, autocontidas e autodescritas que podem ser publicadas, localizadas e invocadas através de protocolos de aplicação bem definidos, tais como HTTP (*Hyper Text Transport Protocol*) e SMTP (*Simple Mail Transport Protocol*). Eles foram concebidos para o ambiente de negócios. Empresas desejam integrar seus sistemas para automatizar processos de negócios. Porém um fator que torna essa integração muito difícil é a diversidade dos seus sistemas de tecnologia da informação. *Web Services* surgiram para resolver esse problema.

Web services utilizam TCP/IP e XML pelos motivos dados. Primeiro, os protocolos TCP/IP estão consolidados como os protocolos de comunicação em rede. Segundo, XML (*eXtensible Markup Language*) é o padrão para troca de dados [6]. Associadas com essas tecnologias padrão, as seguintes direções também foram consideradas: o conteúdo da *Web* está se tornando cada vez mais dinâmico; largura de banda e armazenamento estão ficando mais baratos; e *thin clients* estão se tornando mais comuns.

Através da observação dessas direções, *Web Services* consideram os seguintes requisitos [6]: devem ser capazes de combinar conteúdos a partir de fontes diferentes; entregar diferentes tipos de conteúdo; lidar com grandes quantidades de dados de maneira inteligente e servir os diversos tipos de dispositivos, plataformas, e navegadores, entregando conteúdo através de vários tipos de conexão.

Web Services não são relacionados diretamente à navegação *Web* para humanos. *Web Services* representam uma nova tecnologia para aplicações distribuídas que permitem a criação de aplicações cliente/servidor. É uma tecnologia independente de plataforma e de

linguagem, uma vez que faz uso de padrões XML, o que facilita a criação de sistemas distribuídos fracamente acoplados, em que o cliente não precisa ter nenhum conhecimento sobre o serviço até invocá-lo. Além disso, a comunicação entre os serviços pode ser feita através do protocolo de transporte padrão *World Wide Web*, o HTTP, o que possibilita a passagem das mensagens entre domínios separados por *firewalls*, uma vez que a maioria dos *proxies* da Internet e *firewalls* não se importa com tráfego HTTP. Tais características facilitam a interação entre aplicações na Internet e entre aplicações em *grids* computacionais [6].

Eles podem realizar desde funções muito simples até procedimentos comerciais ou científicos complexos. Podem se comunicar com outros serviços (e também outras aplicações) sem restrições de *hardware*, sistema operacional ou ambiente de programação, permitindo uma interoperabilidade mais rápida entre sistemas parceiros.

O modelo de *Web Services* é a base para seu melhor entendimento. O modelo define três papéis: provedores, consumidores e registradores (*brokers*) do serviço (ver Figura 1). Provedores fornecem serviços. Consumidores desejam usar serviços. Finalmente, os registradores ajudam os provedores e os consumidores a se encontrarem.

Esse modelo permite três operações importantes: **publicar** (ou **cancelar a publicação**), **localizar** e **comunicar**. A primeira é executada entre os provedores e os registradores do serviço: os provedores do serviço entram em contato com os registradores para **publicar** ou **cancelar a publicação de** um serviço. A operação **localizar** ocorre entre os consumidores e os registradores do serviço. Quando um consumidor deseja usar um serviço, ele entra em contato com os registradores de serviço para localizar o serviço apropriado. Os consumidores de serviços descrevem os tipos de serviços que estão procurando, e os registradores entregam os resultados da busca que melhor se adequam ao pedido. Finalmente, quando o consumidor encontra o serviço apropriado, ele negocia com o provedor do serviço para poder acessar e invocar os serviços a partir do provedor (operação **comunicar**).

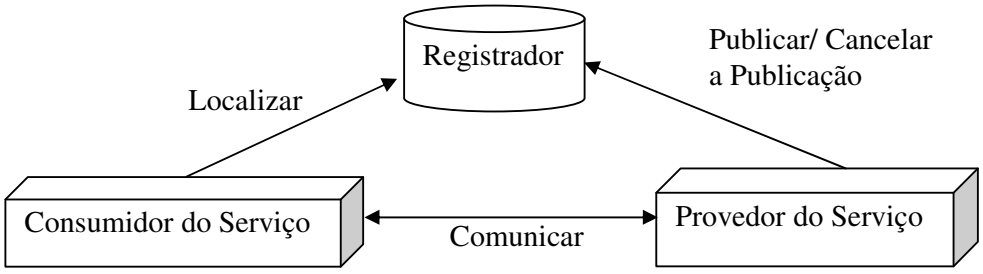

Figura 1: Modelo de *Web Services*

Alguns padrões foram criados para *Web Services*. SOAP (*Simple Object Access Protocol*) é o protocolo padrão para troca de mensagens usado entre *Web Services*. SOAP é independente de transporte. Por exemplo, ele pode funcionar sobre HTTP e SMTP. Ele pode ser usado para trocar qualquer tipo de informações XML [8].

UDDI (*Universal Description Discovery and Integration*) é uma especificação que define um registrador de *Web Services*. Ele possibilita que provedores de serviços publiquem uma descrição de seus serviços disponíveis e recebam requisições de consumidores. Todas as mensagens são enviadas via SOAP. A operação **localiza** (também chamada **descoberta**) ocorre em um registrador baseado em UDDI. A abordagem UDDI para descoberta é disponibilizar um registrador distribuído de serviços através da *Web*. Nesse registrador distribuído, negócios e serviços são descritos através do formato XML comum. Os dados estruturados nos documentos XML são facilmente procurados, analisados e manipulados [7].

A linguagem comum para descrever *Web Services* é chamada WSDL (*Web Services Description Language*). WSDL é uma linguagem XML para descrever interfaces abstratas de serviços, pontos finais de serviços, formato de mensagem e protocolo para acessar serviços.

Juntas, essas tecnologias fornecem toda a infra-estrutura necessária para o funcionamento de *Web Services*: provedores de serviço podem descrever seus serviços (WSDL); consumidores podem descrever o que eles estão procurando (WSDL); registradores podem automaticamente associar provedores e consumidores; depois que uma associação é feita, há maneiras de encontrar os métodos disponíveis para interagir com o serviço, juntamente com as informações necessárias para o acesso.

A diferença principal entre um *Web Service* e uma aplicação acessada via um protocolo padrão TCP/IP é que, uma vez que um *Web Service* é publicado, outros *Web Services* (ou outras aplicações) podem descobri-lo e invocá-lo via sua interface pública.

Web Services resolvem problemas de interoperabilidade importantes no contexto de negócios. Na seção seguinte é mostrado como a comunidade *Grid* pode aproveitar tecnologias de *Web Services* em ambientes *Grid.*

2.2 *Grid Services*

Antes de introduzirmos *Grid Services*, é importante conhecer o contexto em que eles surgiram. O problema mais importante que tecnologias *Grid* necessitam superar é o compartilhamento coordenado de recursos pertencentes a diferentes domínios administrativos [9]. Para facilitar o entendimento desse problema, o conceito de organizações virtuais (OVs) foi introduzido. Uma OV é formada por participantes (por exemplo, seres humanos e

organizações) pertencentes a múltiplas instituições, que podem ou não ter relações de confiança entre si, e que desejam compartilhar seus recursos computacionais para realizar uma tarefa.

Nenhuma tecnologia existente foi capaz de satisfazer todos os requisitos necessários para a formação de OVs. Como resultado, uma arquitetura *Grid* foi introduzida. Essa arquitetura é composta por protocolos que podem ser usados por usuários de uma OV para estabelecer, gerenciar e aproveitar o compartilhamento de recursos [9]. Essa arquitetura define, inicialmente, protocolos e serviços que permitem a interoperabilidade entre os participantes do *grid* para formar relações de compartilhamento. Além do mais, a arquitetura propõe APIs (*Application Programming Interfaces*) e SDKs (*Software Development Kits*) padrão para o desenvolvimento de aplicações mais sofisticadas.

Resumidamente, a arquitetura *Grid* proposta em [9] é formada por cinco camadas (ver Figura 2): *fabric*, *connectivity*, *resource*, *collective* e *application*. Essa arquitetura segue o modelo da ampulheta, assim como o TCP/IP. No pescoço da ampulheta, poucos protocolos foram definidos para servir como uma interface entre os (potencialmente muitos) protocolos de baixo nível e (também muitos) protocolos de alto nível. As camadas *resource* e *connectivity* são o pescoço da ampulheta. A camada *fabric* é responsável pelos recursos que serão compartilhados. A camada *connectivity* define os protocolos de comunicação e de autenticação para permitir a comunicação entre os recursos da camada *fabric* e a verificação das identidades dos usuários. A camada *resource* é usada pela camada *connectivity* para negociar, iniciar, monitorar, controlar e contar as operações de compartilhamento nos recursos individuais. A camada *collective* define os protocolos que capturam as interações entre as coleções de recursos. Os protocolos das camadas inferiores são todos de caráter genérico. Na camada *collective*, protocolos de domínio específico começam a aparecer, definidos para OVs específicas. Finalmente, a camada *application* inclui as aplicações dos usuários que operam dentro da OV. A versão 2.0 do *Globus Toolkit* é uma implementação dessa arquitetura.

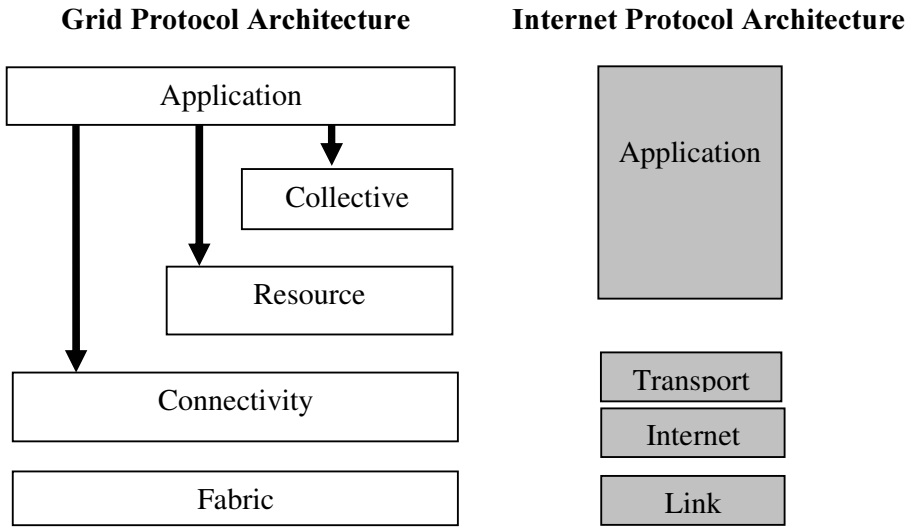

Figura 2: Arquitetura *Grid* [10]

Essa arquitetura aborda o problema da interoperabilidade propondo protocolos padrão. Porém, novos requisitos surgem. Para construir novas aplicações *Grid* é desejável ser capaz de reusar componentes existentes e juntá-los de forma flexível [10]. Protocolos para comunicação *inter-grid* não foram definidos. A solução é uma migração para um modelo orientado a serviço construído sobre conceitos e tecnologias que são herdados a partir das comunidades *Grid* e *Web Services*.

Através da combinação dessas duas tecnologias, o novo conceito de *Grid Service* surgiu e tem sido formalizado com a introdução da arquitetura OGSA (*Open Grid Service Architecture*) [11]. OGSA integra fatores chave de sucesso do *Grid* com mecanismos de *Web Services*, fornecendo uma visão orientada a serviço da arquitetura *Grid.* Em OGSA, tudo – todos os tipos de recursos, programas, redes, etc. – é representado como um serviço.

Essa abordagem herda a arquitetura de camadas descrita em [9], a qual é estruturada em termos de protocolos requeridos para interoperabilidade, enquanto a visão orientada a serviço foca na natureza dos serviços que respondem às mensagens dos protocolos.

OGSA pode ser vista como uma extensão e um refinamento das tecnologias de *Web Services*. De fato, seu conceito chave é o *Grid Service*, um *Web Service* que fornece um conjunto de interfaces bem definidas e que seguem convenções específicas [9]. A visão orientada a serviço simplifica a virtualização. OGSA define semânticas de instâncias de *Grid Service*, porém não coloca restrições nos aspectos de implementação. Ela permite acesso consistente a recursos através de múltiplas plataformas heterogêneas. Em uma visão orientada a serviço, o problema da interoperabilidade é dividido em dois sub-problemas [9]: a definição

das interfaces do serviço e a identificação dos protocolos que podem ser usados para invocar determinada interface. Isso já é resolvido pelas tecnologias de *Web Services*.

OGSA define comportamentos padrão associados a interfaces bem definidas. Em uma visão orientada a serviço, aplicações requerem mecanismos para descoberta de serviços disponíveis. Assim, OGSA define uma representação padrão para elementos de dados sobre o serviço, define interfaces com operações que permitem a recuperação de dados do serviço a partir de instâncias de *Grid Service* e interfaces padrão para registrar informações sobre instâncias de *Grid Service* com serviços registradores.

Padrões para criação dinâmica de instâncias de *Grid Service* são definidos por OGSA. A interface *Factory* define as semânticas que qualquer serviço *factory* tem que fornecer, ou seja, é responsável pela criação das instâncias do serviço entre outras funções. *Grid Services* podem manter estados internos que distinguem uma instância da outra. Como *Grid Services* são *stateful* (ou seja, guardam informações entre uma invocação e outra), potencialmente com longa vida e criados ou destruídos dinamicamente, OGSA define mecanismos para diferenciar instâncias de *Grid Service* que fornecem a mesma interface.

Como se pode ver, estamos falando sobre um ambiente muito dinâmico, em que serviços são criados e finalizados, explicitamente ou não (por causa de falhas, por exemplo). Muitos *Grid Services* são instâncias transientes e têm que ser finalizados quando outros serviços terminam. OGSA define interfaces padrão para propósitos de gerência de ciclo de vida.

Interfaces padrão são também fornecidas para propósitos de notificação. OGSA define interfaces que permitem que *Grid Services* distribuídos notifiquem outros sobre mudanças em seu estado.

Interfaces que tratam problemas como gerência de serviços, autorização e gerência de políticas ainda não são definidas. Supõe-se que a interface de gerência ofereça operações que permitam grandes conjuntos de instâncias de *Grid Services* serem gerenciados.

OGSA tem como objetivo definir uma arquitetura para aplicações *Grid* que seja comum, padrão e aberta, ou seja, especificar interfaces padrão para todos os serviços encontrados em aplicações *Grid*, tais como serviços de gerência de tarefas, de gerência de recursos, de segurança, entre outros [12].

OGSA apenas define os padrões para *Grid Services.* Especificações formais e técnicas mais detalhadas são padronizadas através do padrão OGSI (*Open Grid Services Infrastructure*) [12].

Em janeiro de 2004, o *framework* WSRF (*WS-Resource Framework*) foi proposto como um refatoramento e evolução do OGSI. Seu objetivo é explorar novos padrões para *Web Services* e evoluir o OGSI com base em experiências de implementações anteriores. O WSRF permanece essencialmente com todas as capacidades funcionais do OGSI, mudando algumas sintaxes e adotando uma terminologia diferente em sua apresentação. Além disso, o WSRF divide a funcionalidade OGSI em cinco especificações distintas e componentizáveis [13].

A próxima seção apresenta o conceito de gerência de nível de serviço.

2.3 Gerência de Nível de Serviço – SLM

O objetivo da gerência de nível de serviço é garantir que níveis de serviço apropriados serão oferecidos aos consumidores [14]. O conceito surgiu como uma evolução natural dos sistemas de gerência.

Na metade dos anos 70, os cientistas estavam começando a mostrar que era possível conectar máquinas através de fios e dispositivos de rede como roteadores [15]. Os dispositivos eram gerenciados através de terminais burros e linguagem de texto especial era usada. Gerência de dispositivos ainda existe. Hoje, contudo, um dispositivo pode ser configurado remotamente com o auxílio de interfaces gráficas.

No fim da década de 80, começou-se a perceber que a maneira de gerenciar os dispositivos de rede não era apropriada. Como as redes estavam crescendo e se tornando cada vez mais complexas, surgiu a necessidade de centralizar as funções de gerência de rede [15]. Foi assim que, aproximadamente na década de 90, surgiram os sistemas de gerência de rede (*Network Management Systems* - NMS). Nessa mesma época, os fabricantes começaram a desenvolver produtos que possibilitavam monitorar o tráfego dos enlaces da rede. Esses produtos eram chamados de sistemas de gerência de tráfego (*Traffic Management Systems* - TMS).

Pouco tempo depois, aproximadamente 1992, surgiu o interesse em sistemas de gerência de sistemas (*Systems Management Systems* - SMS). Sistemas eram vistos como classes de computadores, tais como estações de trabalho UNIX, PCs, estações e servidores NT. Tarefas típicas dessas ferramentas de gerência incluíam gerência de configuração, de desempenho (ex. uso de CPU, disco e memória das máquinas) e de segurança [15].

Esses sistemas são complementares e, juntos, poderiam compor uma solução de gerência mais completa. Porém, no início, houve uma certa resistência dos fornecedores a integrarem suas soluções. Até que, em 1993, o conceito de integração e programas parceiros começou a ganhar força na indústria, com a idéia de que nenhum vendedor sozinho poderia

fornecer todas as soluções, e os vendedores de NMS e de SMS resolveram integrar suas soluções [15].

Em 1994, surgiu o conceito de sistemas de gerência de aplicações (*Application Management Systems* - AMS). Os vendedores começaram a perceber que os administradores de negócios estavam mais interessados em gerenciar suas aplicações do que em gerenciar as redes e os sistemas de computadores que as sustentavam. Como resultado, os quatro tipos de sistemas de gerência passaram a coexistir: NMS, TMS, SMS e AMS. Nenhum deles oferecendo uma solução de gerência completa [15].

Em 1996, surgiram os sistemas de gerência empresarial (EMS – *Enterprise Management Systems*) com a finalidade de integrar esses quatro tipos de sistemas complementares, ou seja, EMS = NMS + TMS + SMS + AMS.

Posteriormente, percebeu-se que os executivos tinham interesse em gerenciar a operação global do negócio, não em gerenciar os aspectos da rede, tráfego, sistemas ou aplicações. Assim, serviços foi o termo utilizado para representar essas operações globais sobre a rede empresarial. Gerência de nível de serviço (*Service Level Management* - SLM) é o processo de identificar, definir e gerenciar os serviços da rede. SLM para redes de empresa nasceu em 1998. Nesse momento, sistemas de gerência de nível de serviço uniram-se aos sistemas de gerência empresarial, assim como os NMSs, TMSs, SMSs e AMSs [15].

Outras definições importantes no contexto de SLM são [15]:

Um **processo de negócio** é a maneira que uma organização coordena e organiza tarefas e informações para produzir um objeto de valor. Processos de negócio incluem muitos serviços gerais. Alguns deles dependem da rede da empresa;

Uma **rede da empresa** consiste de dispositivos e linhas de transmissão, sistemas computacionais e aplicações executando nesses sistemas;

Um **serviço** é uma funcionalidade oferecida por uma rede da empresa para o negócio;

Uma **métrica de serviço** é uma variável cujo valor indica a qualidade de um serviço oferecido pela rede da empresa;

Uma **métrica de componente** é uma variável cujo valor indica o desempenho de algum componente da rede da empresa;

Um **mapeamento de métricas de componentes para métrica de serviço** é uma função que recebe métricas de componentes como valores de entrada e fornece como saída o valor de uma métrica de serviço.

Um **nível de serviço** é algum valor de uma métrica de serviço usado para negociar a qualidade de serviço aceitável. Esse valor também é conhecido como objetivo de nível de serviço (*Service Level Objective*, SLO);

Um **acordo de nível de serviço (SLA)** é um contrato entre um provedor e um consumidor que identifica os serviços sustentados pela rede da empresa, as métricas de serviço para cada serviço oferecido, os níveis de serviço e as obrigações do provedor e do consumidor quando os níveis de serviço não são encontrados;

Relatórios de níveis de serviço (*Service Levels Reports*, SLR) são documentos que comparam os níveis de serviço contratados com as métricas de serviço sendo observadas;

Um **agente** é uma entidade de software que monitora, grava e controla valores de métricas de componentes;

Gerência de nível de serviço (*Service Level Management*, SLM) é o processo de: i) identificar serviços, métricas de serviço, níveis de serviço, métricas de componentes e mapeamentos de métricas de componentes para métrica de serviço; ii) negociar e organizar um SLA; iii) implantar agentes para monitorar e controlar métricas de componentes; iv) gerar SLRs e v) melhorar o nível dos serviços sendo oferecidos.

O trabalho desta dissertação é focado na fase de monitoração das métricas no processo de SLM, ao qual denominamos de auditoria de SLAs. Portanto, considera-se que o SLA já foi negociado e estabelecido e não são consideradas ações a serem tomadas para melhorar o nível do serviço caso constate-se que algum SLO não foi cumprido.

2.4 Acordos de Nível de Serviço – SLAs

SLAs são elementos centrais no processo de gerência da qualidade do serviço entregue ou recebido por uma organização de TI (Tecnologia da Informação). É através deles que são definidos os níveis de serviço considerados aceitáveis para o consumidor e que são possíveis de serem alcançados pelo provedor do serviço. SLAs beneficiam os consumidores ao permitir que eles negociem os valores que desejam para os SLOs. Eles também são úteis para o provedor, pois limitam o desejo do cliente e também impõem limitações na carga submetida pelo consumidor. Portanto, ao se estabelecer valores para os SLOs, o provedor se compromete a entregar o nível de serviço que ele é capaz de atender caso o consumidor não ultrapasse a quantidade determinada de requisições e pague o que foi acordado.

Geralmente, um SLA é estabelecido entre um provedor e um cliente que falam línguas bastante diferentes. Ao se definir um SLA, está se documentando as expectativas do

consumidor e as promessas do provedor através de uma linguagem comum, facilitando o entendimento e a clareza entre eles.

Processos são necessários para criar, manter e administrar um SLA [14]. A criação de um SLA começa com o compromisso de se negociar. Geralmente, esse compromisso é feito entre a alta gerência da organização provedora do serviço e a alta gerência da organização consumidora. Em seguida, um time de pessoas é escolhido para negociar os termos do SLA. Ao se escolher um time para negociar um SLA, deve-se considerar os seguintes pontos: deve-se haver uma representação igual de pessoas da organização provedora e da organização cliente para evitar que seja estabelecido um SLA injusto para um dos lados; os membros do time devem possuir algum interesse no serviço fornecido; os membros do time devem ser especialistas sobre o serviço ou sobre os impactos deste no negócio [14].

O processo de negociação do SLA envolve a troca de informações com o objetivo de encontrar uma solução razoável para ambos os negociantes. O grupo de usuários do serviço deve ser capaz de explicar seus requisitos claramente e o impacto no negócio dos vários níveis de serviço. De maneira semelhante, o grupo fornecedor do serviço deve analisar o impacto financeiro e técnico de se entregar os níveis de serviço negociados. É importante que ambas as partes coletem dados, tais como o nível de serviço que está sendo fornecido atualmente, as métricas possíveis de se medir e os níveis possíveis de se atender, antes da negociação do SLA [14].

O passo seguinte à negociação do SLA é sua documentação. Os termos de um SLA propostos em [14] são:

Identificação das partes que participam do acordo;

Prazo do SLA: em geral, dois anos, uma vez que a criação de um SLA requer muito trabalho e que a tecnologia e as condições do negócio mudam rapidamente;

Escopo: define os serviços cobertos pelo SLA;

Limitações: especifica as circunstâncias sob as quais o provedor do serviço não é obrigado a oferecer os níveis negociados. O provedor concorda em oferecer os níveis negociados somente quando nenhuma limitação é excedida. Limitações típicas são: volume de carga (transações por minuto ou por hora, número de usuários, etc), topologia (distribuição de usuários, localização em que os serviços são entregues, etc) e financiamento;

Objetivos de níveis de serviços (*Service Level Objectives,* SLOs): são níveis de serviço negociados para serem fornecidos. As categorias mais populares de SLOs são disponibilidade, desempenho e precisão. É importante não negociar níveis que não podem ser

atendidos. Cada SLO escolhido deve ser alcançável, mensurável, compreensível, aceito por todas as partes, significante, controlável e deve poder ser pago;

Indicadores de nível de serviço (*Service Level Indicators* - SLIs): são as medidas que são de fato realizadas para representar SLOs. É importante sempre considerar a percepção do usuário. Por exemplo, medir disponibilidade de um serviço considerando o ponto de vista dos usuários. Não é suficiente simplesmente monitorar o servidor da aplicação;

Penalidades: indica as conseqüências da não conformidade do SLA. A penalidade deve causar sofrimento ou desconforto para os provedores do serviço;

Serviços opcionais: serviços que não são usualmente oferecidos, mas que podem ser requisitados antecipadamente pelo consumidor;

Exclusões: exclui serviços que não são cobertos pelo SLA, quando necessário;

Relatórios: uma lista de tipos de relatórios que serão gerados e com que freqüência;

Administração: define como o SLA será administrado e quem é responsável por cada processo administrativo necessário;

Análises: definição de períodos regulares em que o SLA será avaliado. Se ambas as partes concordam, avaliações podem acontecer em qualquer tempo. Contudo, avaliações periódicas e obrigatórias devem ser definidas;

Revisões: quando necessário, um novo acordo deve ser escrito. Causas de mudanças são: fusão de empresas, mudança de pessoal administrativo, entre outras;

Aprovações: a assinatura do acordo por ambas as partes.

Resumindo, a gerência de nível de serviço é uma metodologia para negociar métricas de nível de serviço e garantir que elas sejam atendidas. SLAs são instrumentos usados para colocar em prática a gerência de nível de serviço. Eles são contratos entre provedor e consumidor que capturam expectativas sobre níveis de serviço. O valor e a importância da gerência de nível de serviço foram ficando claros; o sucesso do seu uso tem sido documentado em numerosos estudos de caso e sua popularidade está aumentando não somente dentro de organizações de TI, mas também entre provedores de serviço. SLAs podem ser úteis em *Grids* para garantir qualidade nos serviços sendo oferecidos entre domínios administrativos diferentes, mas, para isso, eles necessitam ser muito mais dinâmicos, pois as interações entre os serviços são bastante dinâmicas e mudanças podem ocorrer com uma freqüência bem maior.

2.5 *Globus Toolkit*

O projeto *Globus* une pesquisadores e desenvolvedores em um esforço comum para criar tecnologias fundamentais para o *grid*, o qual permite que pessoas possam compartilhar dados, poder computacional e outras ferramentas *on-line,* de maneira segura, através de fronteiras geográficas, institucionais ou corporativas, sem sacrificar a autonomia local. O projeto é formado por muitas organizações e produz um software de código aberto que é central às atividades de engenharia e de ciência e é substrato para produtos significantes para *grid* oferecidos por companhias líderes de TI [16].

O *Globus Toolkit* (GT) inclui *software* e bibliotecas para descoberta, monitoração e gerência de recursos, além de segurança e gerência de arquivos. Sua última versão, GT3, é a primeira implementação em larga escala da arquitetura OGSA (*Open Grid Services Architecture*). O *toolkit* inclui também *software* para infra-estrutura de informação, gerência de dados, comunicação, detecção de falhas, e portabilidade. É empacotado como um conjunto de componentes que podem ser usados isoladamente ou juntos para desenvolver aplicações.

Toda organização tem seus próprios modos de operação, e a colaboração entre múltiplas organizações é dificultada pela incompatibilidade de recursos como arquivos de dados, computadores e redes [16]. O *Globus Toolkit* foi concebido para remover obstáculos para essa colaboração. Seus serviços principais, interfaces e protocolos permitem aos usuários acessar recursos remotos como se estivessem localizados dentro de sua própria sala de equipamentos, enquanto, simultaneamente, preserva o controle local sobre quando e quem pode usar os recursos [16].

A versão 4.0 do *Globus* está sendo implementada para suportar as reformas submetidas pela arquitetura WSRF (*WS-Resource Framework*).

A versão utilizada para os experimentos deste trabalho foi a 3.2.1, a última liberação do GT3. Essa ferramenta foi escolhida pelo fato de possibilitar a implementação de *Web Services* e *Grid Services* para compor o sistema de auditoria de SLAs entre os serviços e seus clientes e possibilitar a análise desse processo.

Capítulo 3
Trabalhos Relacionados

Acordos de nível de serviço têm sido aplicados para gerência de *Web Services* e, recentemente, para *Grid Services*. Esta seção apresenta os principais trabalhos sendo desenvolvidos nessa área.

3.1 WSMF: *Web Services Management Framework*

WSMF é uma arquitetura lógica para a gerência de recursos, incluindo *Web Services*, através de *Web Services* [1]. Esse *framework* é baseado na noção de objetos gerenciados e seus relacionamentos. Um objeto gerenciado representa, essencialmente, um recurso e expõe um conjunto de interfaces de gerência através das quais o recurso pode ser gerenciado. Dessa forma, relacionamentos entre objetos gerenciados representam relacionamentos entre recursos.

Cada recurso a ser gerenciado possui um objeto gerenciado (um *Web Service*) que o representa. Esse objeto gerenciado implementa uma ou mais interfaces de gerência, representadas através do elemento *PortTypes* em documentos WSDL. As interfaces de gerência possuem os seguintes componentes: espaço de nomes, atributos, operações e notificações. Cada atributo, operação, ou notificação pertence a uma das seis categorias seguintes: monitoração, descoberta, controle, desempenho, configuração e segurança.

O WSMF funciona da seguinte maneira: um gerente do WSMF descobre um conjunto inicial de URLs com os documentos WSDL dos objetos gerenciados; em seguida, ele descobre mais objetos gerenciados, acessando os relacionamentos dos objetos gerenciados já descobertos. De posse dessa informação, o gerente acessa os objetos gerenciados, seus atributos e suas operações através da interface especificada em seus documentos WSDL. Para monitorar os objetos relacionados, o gerente obtém o estado do objeto (*get state*) e para controlar os objetos, ele altera os estados dos ciclos de vida dos mesmos (pára, inicia, suspende, reinicia). O gerente se cadastra nos *Web Services* para pegar ou receber eventos de notificação, quando seu estado muda, e pode realizar a mesma operação em uma coleção de objetos gerenciados através das interfaces de coleção.

A equipe do WSMF está trabalhando, no momento desta escrita, para adaptar o *framework* à infra-estrutura OGSI. Fazendo um paralelo, um objeto gerenciado do WSMF seria um *Grid Service*, o sistema de eventos seria o sistema de notificações do OGSI, e uma *Collection* do WSMF seria um *ServiceGroup* do OGSI. Os estados dos objetos gerenciados seriam guardados como *service data elements* no *Grid Service*, com informações tais como tempo de vida e mutabilidade. Um SLA seria um *Grid Service* que guarda suas informações como *service data elements*, que poderiam ser atualizadas dinamicamente. WSMF seria uma camada sobre OGSI com interfaces de monitoração e de controle [17].

3.2 *Web Service Level Agreement (WSLA) Framework*

Um outro *framework* para gerência de *Web Services* utilizando SLAs é proposto em [2]. O *framework* WSLA consiste de uma linguagem extensível e flexível baseada em um esquema XML (linguagem WSLA) e de uma arquitetura em tempo de execução formada por vários serviços de monitoração de SLAs [18].

As métricas envolvidas nos SLAs são classificadas em:

- Métricas de recursos – medições realizadas diretamente a partir dos recursos;
- Métricas compostas – métricas criadas pela combinação de recursos ou outras métricas compostas de acordo com um algoritmo específico;
- Parâmetros de SLA – métricas associadas a um limiar e usadas em um SLA no contexto de um consumidor específico;
- Métricas de negócio – métricas que mapeiam parâmetros de SLA em termos financeiros.

Consumidores são sempre envolvidos nos parâmetros de SLA e nas métricas de negócio [2]. Um SLA deve definir claramente como as medições serão feitas para obter valores de parâmetros de SLA.

Alguns requisitos do *framework* WSLA são:

- Uma linguagem formal flexível para especificar uma variedade de SLAs;
- Um terceiro elemento pode ser envolvido na monitoração do SLA, exercendo um papel de suporte;
- As atividades de negociação, de criação e de implantação do SLA deveriam considerar abordagens existentes e sistemas desenvolvidos nas áreas de *e-commerce* e de *business-to-business*;

- Serviços de suporte deveriam receber somente as informações necessárias para realizarem seu trabalho;
- A aceitação de um SLA pode ser pensada em termos de capacidade dos recursos. A capacidade de mapear métricas de recursos para parâmetros de SLA é crucial porque antes de aceitar um SLA, deve ser possível verificar se os recursos disponíveis serão suficientes para cumprir o SLA.

O ciclo de vida de gerência de um SLA compreende cinco fases [19]:

- Negociação e Estabelecimento do SLA – realizada por um serviço de estabelecimento de SLA. Esse serviço permite a um consumidor obter métricas de recursos e métricas compostas fornecidas pelo provedor e combiná-las em parâmetros do SLA. Uma vez que o SLA é especificado, o serviço de estabelecimento solicita a aprovação de ambas as partes, define as partes de suporte e suas tarefas e fornece o documento SLA para o serviço de implantação;
- Implantação do SLA – realizada pelo serviço de implantação. O serviço verifica a validade do SLA e distribui as informações apropriadas para as partes de suporte;
- Medição e Avaliação de nível de suporte – o serviço de medição é responsável por medir os parâmetros do SLA enquanto o serviço de avaliação de condições compara as medições com os parâmetros do SLA. Esses serviços podem ser implementados pelas próprias partes assinantes ou por terceiros, seja porque os assinantes não desejam realizar tais funções ou porque não há confiança entre eles. Quando violações são detectadas, o serviço de gerência é informado pelo serviço de avaliação de condições;
- Ações de Gerência Corretivas – quando o serviço de gerência recebe uma notificação, ele toma as ações apropriadas para corrigir o problema. Em um ambiente comercial, decisões não são tomadas baseando-se somente em aspectos técnicos. Logo, o serviço de gerência consulta a entidade de negócios antes de realizar uma ação. A entidade de negócios aprova ou não a ação proposta pelo serviço de gerência baseando-se em informações de negócios. Ações de gerência corretivas flexíveis e automáticas são difíceis de implementar e ainda é um assunto em aberto na área de gerência;
- Terminação do SLA – Um SLA pode terminar em uma data específica, ou quando certos eventos ocorrem (tais como uma violação do SLA). A terminação é similar ao estabelecimento do SLA.

Esses serviços foram implementados como *Web Services* e compõem o *SLA Compliance Monitor* que é incluso na versão 2.2 do *Emerging Technologies Toolkit* (ETTK)

[20] da IBM, o qual inclui, além de *Web Services*, outras tecnologias emergentes tais como *Grid Services*. Esse trabalho é fruto de vários laboratórios de pesquisa e desenvolvimento da IBM.

3.3 G-QoSM: *Grid QoS Management Framework*

O *framework* "*Grid QoS Management*" (G-QoSM) descrito em [3] possibilita aos usuários especificar, localizar e executar *Grid Services* baseados em restrições de QoS. Além da interface funcional tradicional, o *Grid Service* possui uma interface de gerência, na qual publica informações de QoS, possibilitando aos consumidores conhecer os requisitos de QoS que ele pode fornecer. O *framework* fornece três funções principais: i) suporte para descoberta de recursos e de serviços baseada em propriedades de QoS; ii) provisão de garantias de QoS nos níveis de aplicação, de *middleware* e de rede, e o estabelecimento de SLAs para forçar parâmetros de QoS; iii) suporte para gerência de QoS de recursos alocados baseados em um SLA pré-negociado.

G-QoSM consiste de três componentes principais: i) *Application QoS manager* (AQoS); ii) *Resource Manager* (RM); iii) *Network Resource Manager* (NRM). O AQoS é o componente principal, o qual interage com os clientes, com os RMs, com os NRMs e com os AQoSs de outros domínios. O AQoS negocia os SLAs com os clientes e comunica os parâmetros do SLA ao RM correspondente. Além disso, ele é responsável por garantir a conformidade do SLA nos recursos alocados e fornecer ações no caso de violações do SLA. O RM é considerado uma combinação de *Globus* e de uma extensão ao UDDI chamado UDDIe [21], o qual registra e busca serviços através de informações de QoS publicadas por eles mesmos. O RM hospeda os *Grid Services*, cria um ambiente de execução com especificações de QoS e registra os serviços de acordo com capacidades de QoS. O NRM gerencia os recursos de QoS de um dado domínio baseado nos SLAs negociados naquele domínio e é responsável por gerenciar a comunicação interdomínio, coordenando SLAs através das fronteiras dos domínios. Interações entre esses componentes são mostradas na Figura 3.

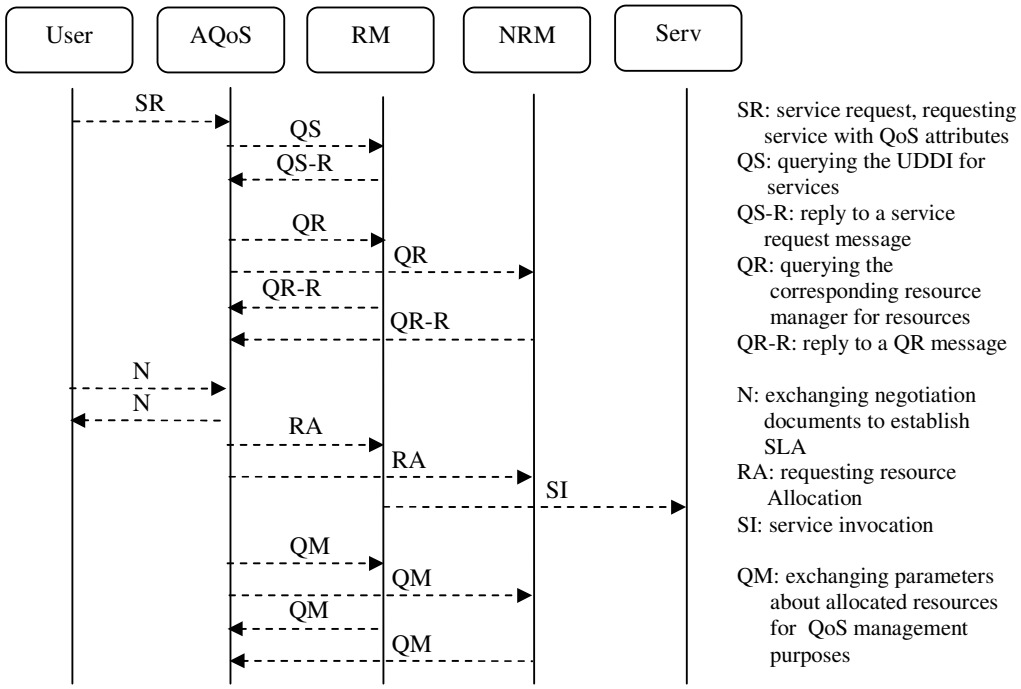

Figura 3: Diagrama de seqüência entre componentes do G-QoSM [3]

Serviços são descobertos com base em propriedades de QoS. Uma vez que o serviço correto é descoberto, o gerente AQoS interage com o RM e com o NRM para encontrar conjuntos de recursos que satisfaçam os requisitos do cliente. Um RM também pode pedir para RMs remotos alocarem recursos. Somente depois que todas essas consultas são respondidas, o AQoS pode negociar um SLA com o cliente. Em seguida, os recursos apropriados são reservados, o serviço é realizado e todos os gerentes AQoS envolvidos trocam parâmetros de QoS para propósitos de gerenciamento.

3.4 GARA: *Globus Architecture for Reservation and Allocation*

GARA (*Globus Architecture for Reservation and Allocation*) [4] é uma arquitetura do *Globus* para permitir reservas e alocações dinâmicas de recursos heterogêneos, localizados em diferentes domínios administrativos, baseados em critérios de QoS. É construída sobre técnicas e conceitos da arquitetura de gerência de recursos do *Globus* (GRAM) [22], como a co-alocação de recursos computacionais para provisão de QoS, estendendo-a para suportar a descoberta, reserva, alocação e gerência de recursos de vários tipos, tais como poder computacional (CPU), redes, sistemas de armazenamento, com independência de controle local.

GARA introduz o conceito de objeto de recurso genérico, o qual pode ser um fluxo de rede, um bloco de memória, um bloco de disco ou um processo, para o propósito de reformular as funções de alocação específicas de computação do GRAM de modo a abranger os diferentes tipos de componentes. Uma função para criar um objeto é usada para criar qualquer um desses componentes, retornando um manipulador específico do objeto, o qual pode ser usado para monitorar e controlar o mesmo. A tarefa de criar um objeto é dividida em duas fases: reserva e alocação. Na fase de reserva, um agente de co-reserva faz a descoberta de uma coleção de recursos que satisfazem requisitos de QoS da aplicação e faz uma reserva, a qual fornece alguma confiança de que a alocação será realizada com sucesso. Nesse momento, nenhum objeto é criado, porém um manipulador para a reserva é retornado, através do qual podem ser feitos a monitoração e o controle da reserva. Na fase de alocação, o manipulador da reserva é passado para o agente de co-alocação que faz a criação dos objetos, ou seja, aloca os recursos, e faz a associação dos objetos com as reservas. As operações de criação de objetos e de reservas são implementadas por um gerente de alocação e de reserva do *Globus*.

3.5 Considerações sobre os Trabalhos Relacionados

O *framework* WSMF é uma proposta da HP para ser um padrão de gerência de recursos através de *Web Services*. No momento desta escrita, o refatoramento do WSMF para o mundo *Grid* é um trabalho em andamento. Os objetos gerenciados são *Web Services* que representam recursos ou os próprios serviços. Tais *Web Services* implementam as interfaces de gerência e, portanto, são os responsáveis por fornecer as informações de gerência. No WSMF, apesar de não haver restrições ao uso de SLAs, a gerência de *Web Services* não os leva em consideração e, portanto, não há preocupação em relação à confiança das informações de gerência fornecidas pelos objetos gerenciados. Ao contrário desse trabalho, que descreve uma solução de gerência de serviços sem uso de SLAs, o trabalho apresentado nesta dissertação é focado na fase de monitoração dos SLAs estabelecidos entre clientes e provedores de serviços.

O *framework* WSLA é um trabalho da IBM para especificação e monitoração de SLAs para *Web Services*. Consiste de uma linguagem de definição de SLAs (WSLA) baseada em um esquema XML e de uma arquitetura em tempo de execução, a qual fornece vários serviços de monitoração que podem ser terceirizados para garantir uma maior objetividade. O *framework* WSLA está incluso no *Emerging Technologies Toolkit*, que inclui tecnologias emergentes como *Web Services* e *grids* computacionais. O projeto da IBM oferece a gerência

de SLAs para *Web Services* desde o momento da negociação do SLA, passando pela sua monitoração até a tomada de ações corretivas através de ferramentas de gerência quando uma violação do SLA é detectada. O processo de monitoração pode ser realizado tanto pelas partes assinantes quanto por terceiros, os quais podem interceptar as requisições dos clientes ou então submeter requisições falsas para fins de instrumentação. O trabalho apresentado nesta dissertação se diferencia deste pelo fato de focar, detalhar e analisar o processo de monitoração também chamado de auditoria de SLAs. Apesar de o *framework* WSLA citar várias possibilidades de realização de instrumentação e monitoração dos serviços, ele não mostra em detalhes cada uma dessas possibilidades. Nesta dissertação são apresentadas várias arquiteturas possíveis para auditoria de SLAs entre serviços, além de vantagens e desvantagens relacionadas a diversos fatores - entre eles a questão da confiança entre as partes assinantes e do impacto no desempenho do provedor do serviço devido ao processo de auditoria – de cada uma dessas arquiteturas.

Já o *framework* G-QoSM trata da gerência de SLAs para *Grid Services*, incluindo o processo de descoberta de serviços com base em requisitos de QoS, a negociação de SLAs, sua monitoração e a execução de ações de gerência em caso de não cumprimento dos SLAs. Porém, seu foco é proporcionar aos *Grid Services* uma forma de descrever suas propriedades de QoS e aos seus usuários selecionar serviços ou recursos com base em requisitos de QoS. A gerência de QoS é realizada com base em SLAs para fins de fazer adaptação para atender aos requisitos de QoS dos clientes, porém as informações de gerência são disponibilizadas através de ferramentas de instrumentação locais, sem a preocupação com relações de confiança entre as partes envolvidas no SLA. A fase de auditoria de SLAs não é detalhada e nem analisada sob diversos aspectos em vários cenários como é realizado nesta dissertação.

O projeto *Globus* tem uma arquitetura para descoberta, reserva e alocação de recursos heterogêneos baseadas em QoS (GARA). A aplicação cliente solicita ao agente de reserva uma reserva de recursos que atendam a determinados requisitos de QoS. Depois que o gerente de reserva de recursos encontra recursos que possam atender a esses requisitos, o agente de alocação aloca os recursos e devolve ao cliente manipuladores para os recursos. A aplicação cliente pode fazer monitoração de QoS nos recursos através de um manipulador para o recurso. Apesar de essa arquitetura possibilitar a monitoração de QoS nos recursos alocados, esse processo não é detalhado e nem o uso de SLAs é especificado.

Capítulo 4

Arquiteturas para Auditoria de SLAs entre *Web* e *Grid Services*

O objetivo inicial desta pesquisa era determinar a melhor forma de auditar SLA de maneira independente. Durante o trabalho, contudo, percebemos que não há uma solução simples para esse problema. Arquiteturas diferentes possuem diferentes vantagens e desvantagens, e qual destas é a melhor depende de qual critério o usuário acha o mais importante. Neste capítulo serão apresentadas sete arquiteturas para auditoria de SLA.

4.1 Arquitetura Básica

Na solução mais básica para lidar com auditoria de SLAs, o Provedor avalia os SLIs calculados pelo Consumidor, que, por sua vez, avalia os SLIs calculados pelo Provedor (ver Figura 4). Neste caso, não há uma terceira entidade envolvida. Contudo, para este tipo de auditoria ser possível, é necessário que haja confiança entre as partes que assinaram o contrato, que é uma situação improvável entre serviços em um ambiente *Grid*. Um outro problema é que os próprios serviços se preocupam com questões não relacionadas ao seu negócio, tais como instrumentação e auditoria.

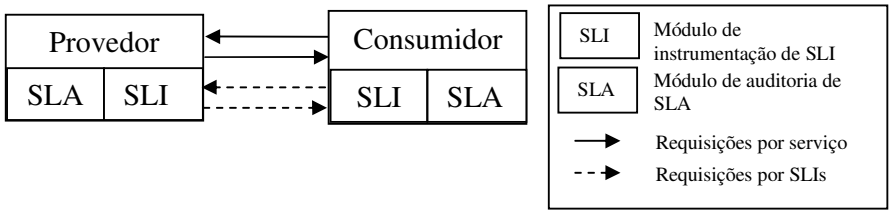

Figura 4: Provedor e Consumidor avaliam seu próprio SLA.

4.2 Arquitetura Auditor Independente

Em uma segunda solução, uma terceira entidade – chamada Auditor – é responsável por avaliar o SLA estabelecido entre as partes assinantes (ver Figura 5). Nessa solução, o Auditor pede os SLIs para os assinantes e os compara com os valores estabelecidos no SLA para verificar o cumprimento do contrato. A avaliação do SLA pode ser automatizada com o auxílio de uma linguagem comum que defina os SLAs [23]. Assim, um único serviço Auditor

pode ser usado para avaliar vários SLAs estabelecidos entre diversos serviços. Se o Auditor for um *Grid Service*, instâncias desse serviço podem ser usadas para distribuir o processamento das avaliações dos SLAs. Uma desvantagem dessa solução é que os próprios serviços precisam fornecer SLIs para o Auditor, causando uma alteração (intrusividade) no código original. Observe que essa solução também requer que os SLIs fornecidos pelos serviços sejam confiáveis.

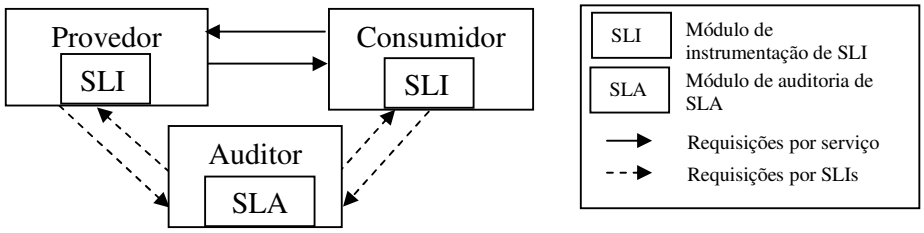

Figura 5: Auditor avalia o SLA a partir dos SLIs fornecidos pelo Consumidor e pelo Provedor.

4.3 Arquitetura Inspetor Independente

Uma terceira solução para lidar com a questão da confiança - sem carregar os serviços com código de instrumentação - delega essa função para um serviço de suporte, chamado Inspetor (ver Figura 6). Esse serviço calcula valores para os SLIs através da interface do Provedor e os entrega ao Auditor, o qual realiza a avaliação do SLA. Nesse caso, as partes assinantes não precisam instrumentar seus códigos para fazer auditoria de SLAs, mas precisam confiar tanto no Inspetor como no Auditor. Como um Inspetor é um Consumidor do Provedor, ele necessita conhecer a interface do Provedor. Logo, cada Provedor pode ter seu próprio Inspetor, o qual sabe como calcular seus SLIs. Um ponto a considerar é que o caminho de rede seguido pelas requisições do Inspetor pode ser diferente do caminho seguido pelas requisições do Cliente. Logo, para lidar com esse problema, poderia haver um Inspetor para cada interação Consumidor-Provedor (SLA) e o Inspetor poderia ficar próximo ao Consumidor ou poderia conhecer uma média do atraso de rede sofrido pelas requisições que seguem o caminho Consumidor-Provedor e adaptar (somar ou subtrair) seu atraso de rede de acordo com essa média. A desvantagem dessa abordagem são as requisições extras geradas pelo Inspetor para o Provedor. Essas requisições extras podem causar efeitos indesejáveis (tais como a modificação da base de dados do Provedor) e impor carga extra, reduzindo o desempenho do Provedor, ou seja, aumentando o tempo com que o Provedor atende a uma requisição. Um outro problema com essa solução é que o Provedor pode ser capaz de identificar as requisições solicitadas pelo Inspetor e conceder tratamento preferencial a elas.

Uma possível solução para o problema de tratamento preferencial consiste em usar técnicas para ocultar o endereço de rede, tais como *Anonymizer* [24], para esconder os endereços do Consumidor e do Inspetor. Porém, devem ser tomados outros cuidados para evitar que informações de nível de serviço revelem quais requisições vêm do Inspetor (ex. requisições submetidas em interesse de um usuário "teste").

Ainda um outro problema a ser observado é que o Inspetor não calcula os SLIs do lado do Consumidor, tal como a carga submetida pelo Consumidor e, logo, não pode auditá-los.

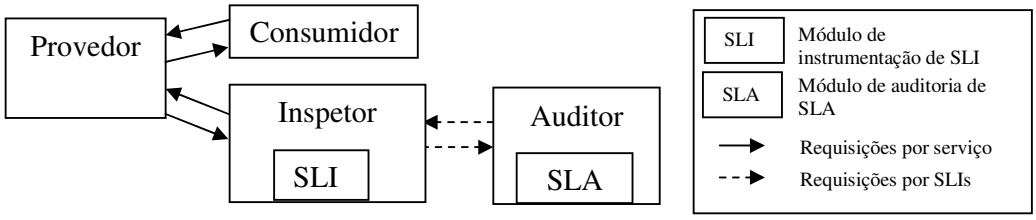

Figura 6: Auditor avalia o SLA a partir de SLIs fornecidos pelo Inspetor e calculados através de requisições extras.

Para resolver o problema dessa arquitetura não poder medir a carga de requisições submetidas pelo Consumidor sugere-se o uso de tíquetes (ver Figura 7). Essa solução consiste de o Auditor conceder uma quantidade de tíquetes T assinados digitalmente para o Consumidor. O Provedor somente atende a uma requisição se esta vier com um tíquete assinado pelo Auditor. Os tíquetes possuem um tempo de vida útil para evitar que o Consumidor os acumulem e sobrecarreguem o Provedor. Quando o tempo de vida expira, o Consumidor pede mais tíquetes ao Auditor. Como o SLA estabelece um número máximo de requisições m permitido ao Consumidor durante um certo período de tempo, o Auditor só pode conceder n tíquetes, tal que $n \leq m$, com o tempo de vida definido dentro do período determinado no SLA. Ao receber uma requisição, o Provedor, além de verificar se ela possui o tíquete assinado pelo Auditor, precisa verificar se o tempo de vida do tíquete ainda não expirou.

Como o Inspetor funciona como um Consumidor, ele também obtém um número de tíquetes i com o Auditor. O Provedor responde tanto as requisições do Consumidor como as do Inspetor, contanto que elas contenham tíquetes assinados pelo Auditor.

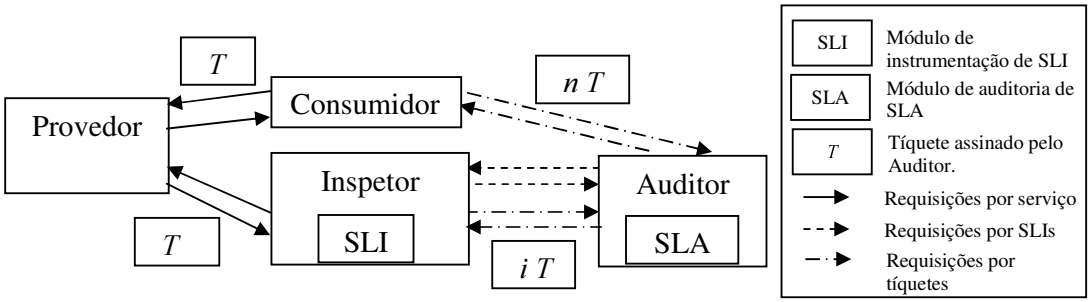

Figura 7: O Auditor assina uma quantidade de tíquetes (*T*) com um tempo de vida determinado para o Consumidor e para o Inspetor poderem fazer requisições ao Provedor.

4.4 Arquitetura Decorador Externo

Uma outra arquitetura pode resolver alguns dos problemas apresentados pelas arquiteturas anteriores; essa arquitetura possui o Inspetor funcionando como um decorador para o Provedor (ver Figura 8). Um decorador é um padrão de projeto bem conhecido que é usado para adicionar funcionalidade extra para um objeto particular de uma classe de objetos [25]. O Consumidor conhece o GSH (*Grid Service Handle* – identificador de rede) do Inspetor ao invés do GSH do Provedor. Dessa forma, todas as requisições vindas do Consumidor passam através do Inspetor antes de chegar ao Provedor. O Inspetor precisa implementar a interface do Provedor além da interface de gerência, a qual fornece os valores dos SLIs. Internamente ao Inspetor, os métodos que calculam os valores dos SLIs são chamados antes ou depois das chamadas aos métodos do Provedor. O Inspetor repassa as requisições do Consumidor para o Provedor, mede a carga submetida pelo Consumidor ao Provedor e calcula os SLIs obtidos a partir do lado do Provedor.

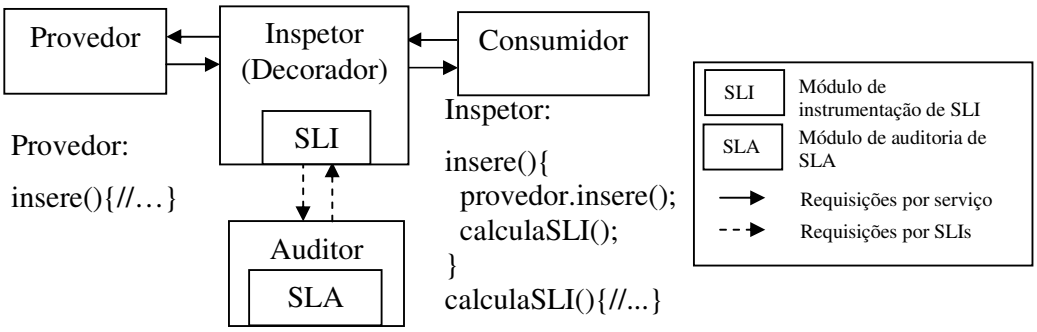

Figura 8: Auditor avalia o SLA a partir de SLIs fornecidos pelo Inspetor funcionando como um decorador do Provedor e calculados sem requisições extras.

Essa solução tem muitas vantagens em relação às tentativas anteriores. Dentre elas, não há requisições extras geradas pelo Inspetor, ou seja, todas as requisições feitas ao Provedor são requisições vindas de um Consumidor real. Nenhuma carga adicional é imposta

ao Provedor e não há efeitos colaterais indesejados (como modificações na base de dados do Provedor). O Provedor não pode identificar as requisições vindas do Inspetor e conceder-lhes tratamento preferencial porque todas as requisições vindas do Consumidor chegam através do Inspetor. Um outro problema resolvido por essa arquitetura é a medição da carga submetida pelo Consumidor, a qual pode também fazer parte das cláusulas do SLA. Uma vez que o Inspetor repassa todas as requisições do Consumidor, ele pode auditá-las através de contagem.

Como acontece com todas as soluções de monitoração e instrumentação, essa solução impõe custos para os serviços. Um custo é a perda de desempenho devido à indireção das requisições causada pela adição de um Inspetor como decorador do Provedor. Um outro custo é um erro na medição do SLI, causado pelo próprio processo de instrumentação. Esses custos são bem mais consideráveis quando o Inspetor é introduzido em um *site* distante que impeça que as requisições percorram o caminho mais curto devido à obrigatoriedade de passarem por ele para fins de auditoria. Contudo, a perda de desempenho pode ser reduzida se, ao invés de se ter apenas um *site* para introduzir todos os Inspetores independentemente da localização dos serviços, existir vários *sites* distribuídos, os quais são escolhidos para introdução dos Inspetores levando-se em consideração a localidade dos serviços que eles vão instrumentar. Claro que a paralelização dos Inspetores também possui um custo adicional de implantação, manutenção e administração dos vários *sites* que os hospedam.

4.5 Arquitetura Decorador Externo com Desvio

Uma alternativa à arquitetura anterior é fazer com que algumas das requisições vindas do Consumidor sigam diretamente para o Provedor e algumas requisições passem pelo Inspetor (ver Figura 9). Essa alternativa tenta reduzir o impacto causado quando todas as requisições são repassadas pelo Inspetor. Enquanto essa solução pode melhorar o desempenho, ela sofre de alguns dos muitos problemas citados anteriormente; por exemplo, o Provedor pode conceder tratamento preferencial às requisições vindas através do Inspetor; ela também traz de volta a dificuldade de auditar as requisições submetidas pelo Consumidor, uma vez que o Inspetor não enxerga todas as requisições. Mais uma vez, poder-se-ia usar os tíquetes para auditar as requisições do Consumidor.

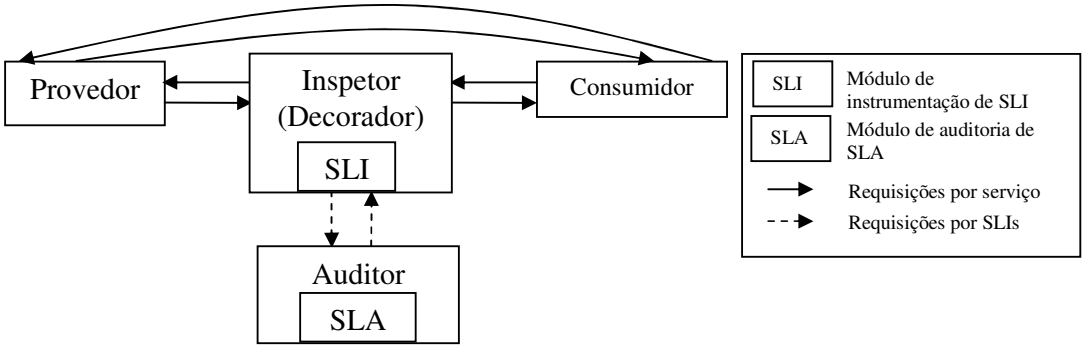

Figura 9: Auditor avalia o SLA a partir de SLIs fornecidos pelo Inspetor funcionando como um decorador para o Provedor. Algumas requisições seguem diretamente para o Provedor para reduzir o impacto no desempenho.

4.6 Arquitetura Sniffers

Uma sexta arquitetura usa *sniffers* confiáveis na LAN do Provedor e na LAN do Consumidor para passivamente capturar pacotes endereçados a eles através da técnica de espelhamento de portas. Através dessa técnica, os pacotes que chegam aos *switches* das LANs endereçados ao Provedor ou ao Consumidor são copiados para as portas dos *sniffers*, os quais são configurados para trabalharem em modo promíscuo para poderem receber pacotes que não são endereçados a eles. O Inspetor calcula os SLIs baseados nas informações dos *sniffers*. Então o Auditor pede os SLIs para o Inspetor (ver Figura 10). Nessa solução: todas as requisições vêm de um Consumidor real; os *sniffers* não influenciam no tempo de resposta às requisições; e o Inspetor pode calcular os SLIs tanto do Provedor quanto do Consumidor a partir das informações dos *sniffers*.

Essa solução assume que os *sniffers* são entregues pelo Inspetor ao Consumidor e ao Provedor. Além do mais, deve ser impossível que o Provedor e o Consumidor possam alterar os *sniffers*, o que requer suporte de hardware [26]. Isso requer a construção de *sniffers* resistentes a alterações ou a instalação de um co-processador seguro em *sniffers* comuns. Um co-processador seguro [27] [28] fornece um núcleo resistente a alterações no qual é possível armazenar chaves criptográficas, processar algoritmos criptográficos e checar alterações e mudanças em todo o sistema computacional. Logo, pode ser usado para adicionar resistência a alterações em sistemas computacionais convencionais. O co-processador seguro também criptografaria e assinaria digitalmente as informações do *sniffer* enviadas para o Inspetor, de modo que os SLIs calculados pudessem ser confiáveis.

Uma das desvantagens dessa solução é que os *sniffers* podem não capturar informação suficiente necessária para calcular os SLIs se os pacotes estiverem criptografados, o que é

uma situação muito comum na comunicação interdomínios. Quando pacotes são criptografados, torna-se muito difícil inferir quais pacotes formam uma mensagem de serviço. Esse problema pode ser evitado nas outras arquiteturas porque, como elas possuem componentes ativos de instrumentação na comunicação, elas vêem toda a informação necessária para a medição dos SLIs. Uma outra desvantagem é que o Consumidor e o Provedor confiam cegamente no Inspetor. Isso porque os *sniffers* podem capturar qualquer tráfego na LAN, podendo dessa maneira acessar informações sensíveis não relacionadas ao SLA sendo auditado. O Consumidor e o Provedor têm que confiar que o Inspetor não agirá dessa forma. Ainda outra desvantagem é a dificuldade de se instalar tais sniffers.

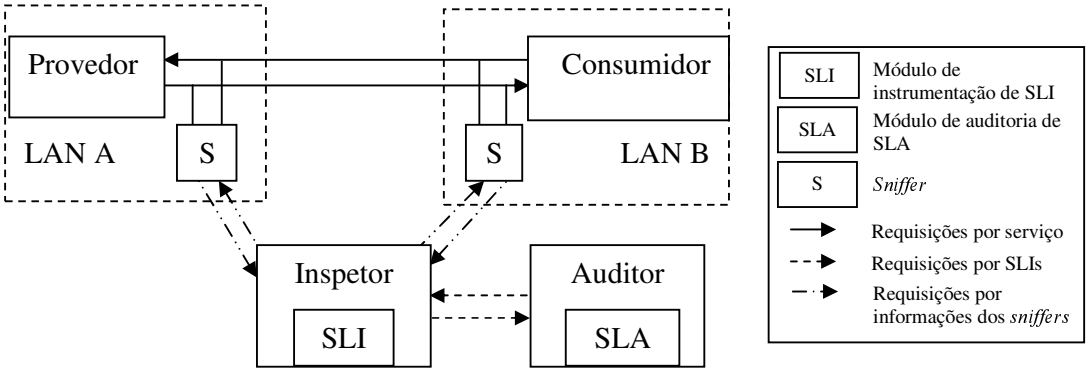

Figura 10: Auditor avalia o SLA a partir de SLIs fornecidos pelo Inspetor, o qual os calcula a partir das informações dos *sniffers*. Aqui *sniffers* são representados com um S.

4.7 Arquitetura Decoradores nos Hosts

Em uma última arquitetura, para cada SLA existe um Inspetor no *host* do Provedor e outro no *host* do Consumidor (ver Figura 11). Esses Inspetores implementam a interface do Provedor. O Consumidor faz uma requisição de serviço através do Inspetor do seu *host* que repassa essa requisição para o Inspetor do *host* do Provedor, o qual, por sua vez, a repassa para o Provedor. Como os Inspetores participam ativamente da comunicação, eles não se deparam com o problema da criptografia encontrado pelos *sniffers* na medição dos SLIs. Por outro lado, isso pode expor informações sensíveis relacionadas ao serviço para os Inspetores. Uma solução seria criptografar as informações sensíveis com a chave pública do Provedor, enviando tais informações sensíveis como um atributo da mensagem completa. Em princípio, criptografar informação de requisição e de resposta poderia impossibilitar o Inspetor de avaliar o SLA. Contudo, na prática, SLAs não tratam de dados da aplicação; eles tipicamente usam SLIs como tempo de resposta e disponibilidade. Logo, essa solução permitiria a transferência segura de dados sensíveis e a auditoria para a maioria das situações.

Todos os problemas resolvidos pela Arquitetura Decorador Externo são também resolvidos por esta, além do problema de desempenho causado pelo atraso sofrido pelas requisições que precisam passar por um Inspetor distante de sua localidade (no caso em que não é feita a paralelização) e sem o custo trazido pela paralelização do serviço Inspetor.

Como na Arquitetura Sniffers, elementos chave (neste caso, os Inspetores) são localizados em *sites* do Provedor e do Consumidor. Como tal, Inspetores devem ser resistentes a alterações, usando, por exemplo, os co-processadores seguros [27] [28].

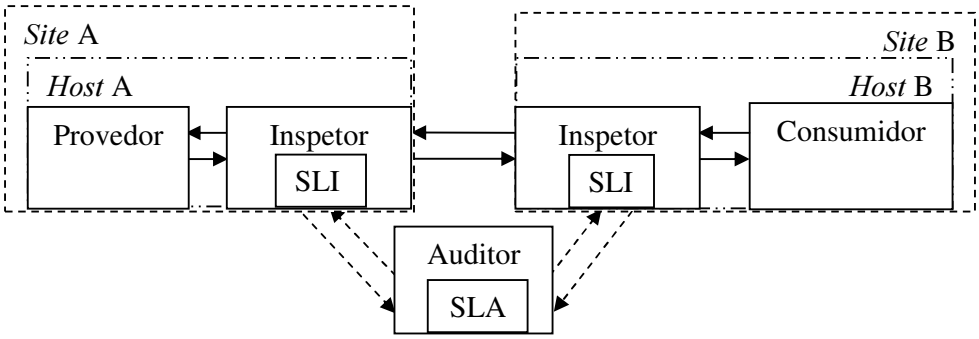

Figura 11: Auditor avalia o SLA a partir de SLIs fornecidos pelos Inspetores, os quais estão localizados nos sites do Provedor e do Consumidor.

4.8 Resumo da Avaliação Qualitativa

A Tabela 1 resume a análise das vantagens e desvantagens de cada arquitetura, fornecendo uma comparação global para os leitores. Como discutido acima, as características exploradas na avaliação são:

Intrusividade: significa a inclusão de código de instrumentação e/ou auditoria no código ou no hardware das partes assinantes do SLA.

Confiança: expressa se é necessário que haja confiança entre os participantes do SLA, e, caso haja, em que processos a confiança é necessária.

Requisições Extras: expressa se a arquitetura em questão faz uso de requisições de serviço extras vindas de um Consumidor falso para fins de auditoria, as quais provocam aumento de carga e efeitos colaterais no serviço (inclusão de dados falsos na base de dados).

Tratamento Preferencial: expressa se o Provedor da arquitetura em questão consegue identificar as requisições vindas de um Inspetor e, com isso, tenha a possibilidade de conceder tratamento preferencial a essas requisições em prejuízo das requisições vindas dos Consumidores reais.

Carga do Consumidor: expressa se a arquitetura em questão consegue medir a carga do Consumidor, ou seja, a taxa de requisições submetidas.

Mensagens Criptografadas: expressa se a arquitetura considerada consegue realizar auditoria nos casos em que a comunicação entre Provedor e Consumidor é realizada através de mensagens criptografadas.

Tabela 1: Comparando as características das sete arquiteturas.

	Intrusividade	**Confiança**	**Requisições Extras**	**Tratamento Preferencial**	**Carga do Consumidor**	**Mensagens Criptografadas**
1. Arquitetura Básica	no código	instrumentação e auditoria	não	não	sim	sim
2. Arquitetura Auditor Independente	no código	instrumentação	não	não	sim	sim
3. Arquitetura Inspetor Independente	nenhuma	nenhuma	sim	sim	uso de tíquetes	sim
4. Arquitetura Decorador Externo	nenhuma	nenhuma	não	não	sim	sim
5. Arquitetura Decorador Externo com Desvio	nenhuma	nenhuma	não	sim	uso de tíquetes	sim
6. Arquitetura Sniffers	nenhuma	nenhuma	não	não	sim	não
7. Arquitetura Decoradores nos Hosts	no hardware	opcional	não	não	sim	sim

Através da Tabela 1, vê-se que as Arquiteturas Básica e Auditor Independente são intrusivas no código dos assinantes, uma vez que a instrumentação (cálculo dos SLIs) e a auditoria dos SLAs, no caso da Arquitetura Básica, e que a instrumentação, no caso da Arquitetura Auditor Independente, são implementadas pelos próprios assinantes. Já a Arquitetura Decoradores nos Hosts não é intrusiva no código dos assinantes, pois a instrumentação é realizada pelo Inspetor, porém é intrusiva no hardware dos assinantes, uma vez que o código do Inspetor é executado nos *hosts* dos assinantes. As outras arquiteturas não são intrusivas nem no código e nem no hardware.

Em relação à confiança necessária entre os assinantes, pode-se ver que a Arquitetura Básica exige que haja confiança nos processos de instrumentação e de auditoria e que a Arquitetura Auditor Independente exige confiança no processo de instrumentação. No caso da Arquitetura Decoradores nos Hosts, há duas opções: (i) existe confiança entre o Provedor e o Cliente e executa-se o código do Inspetor naturalmente em seus *hosts*; (ii) não existe confiança entre o Provedor e o Cliente e, portanto, é necessária a aplicação de técnicas para

manter a integridade e a confidencialidade do código executado e dos resultados obtidos pelo Inspetor nos *hosts* não confiáveis. Para as outras arquiteturas não é necessário que haja confiança entre os assinantes do SLA.

Quanto à existência de requisições de serviço extras vindas de um Consumidor falso para fins de auditoria, a única arquitetura que faz uso desse método é a Arquitetura Inspetor Independente.

A possibilidade de o Provedor identificar as requisições vindas do Inspetor e conceder-lhes tratamento preferencial só existe nas Arquiteturas Inspetor Independente e Decorador Externo com Desvio, já que nessas arquiteturas todas ou algumas requisições, respectivamente, vindas dos Consumidores não passam pelo Inspetor.

A auditoria da carga das requisições submetidas pelo Consumidor pode ser feita nas Arquiteturas Básica, Auditor Independente, Decorador Externo, Sniffers e Decoradores nos Hosts através da própria atividade de auditoria, uma vez que todas as requisições são monitoradas e que tanto o Consumidor como o Provedor são instrumentados. Já na Arquitetura Inspetor Independente, a auditoria da carga do Consumidor não pode ser feita pela atividade de auditoria, uma vez que o Inspetor instrumenta apenas o Provedor. Na Arquitetura Decorador Externo com Desvio, a auditoria da carga submetida pelo Consumidor também não pode ser feita pelo método de auditoria apresentado por essa arquitetura porque apenas uma amostra de requisições é monitorada e instrumentada. Portanto, para efetivar a auditoria da carga do Consumidor nas Arquiteturas Inspetor Independente e Decorador Externo com Desvio foi apresentado, como uma solução possível, o método de tíquetes assinados digitalmente pelo Auditor.

O processo de auditoria pode ser realizado nas comunicações entre Provedor e Consumidor em que as mensagens são criptografadas nas Arquiteturas Básica, Auditor Independente, Inspetor Independente, Decorador Externo, Decorador Externo com Desvio e Decoradores nos Hosts, nas quais o processo de instrumentação é feito de maneira ativa, ou seja, em que os SLIs são calculados pelas entidades que possuem a informação completa para seus cálculos. No caso da Arquitetura Sniffers, que faz uso de *sniffers* para o cálculo dos SLIs com base em informações que são capturadas passivamente durante seu tráfego pela rede, não é possível realizar a auditoria caso as mensagens venham criptografadas porque os *sniffers* não terão acesso a informações suficientes necessárias para o cálculo dos SLIs.

Capítulo 5
Análise de Desempenho

Neste capítulo é feita uma análise sobre os fatores que afetam o desempenho das várias arquiteturas apresentadas na seção anterior, ou seja, é feita uma análise de quanto desempenho é perdido devido à auditoria de SLAs na visão do cliente.

5.1. Avaliação Analítica

Em termos gerais, dois fatores podem reduzir o desempenho dos serviços em um processo de auditoria. O primeiro é o tamanho da amostra de requisições adicionais feitas ao Provedor para calcular os valores dos SLIs. Quanto maior o tamanho da amostra das requisições adicionais, maior o efeito intrusivo no desempenho do serviço. Um outro fator é o que denominamos de Erro de Interferência de Medição (EIM), que é o erro introduzido no valor medido devido ao próprio processo de medição. Por exemplo, o SLI tempo de resposta calculado pode divergir do valor do tempo de resposta obtido caso a auditoria não fosse realizada. Esses fatores se apresentam de maneiras diferentes nos vários cenários apresentados.

Na **Arquitetura Básica** e na **Arquitetura Auditor Independente**, nas quais o Provedor e o Consumidor calculam os valores dos SLIs, nenhuma requisição adicional é imposta ao serviço no intuito de obter os valores de SLIs e o fator tamanho da amostra não está presente. Entretanto, há um atraso no tempo de resposta do Provedor ocasionado pelo cálculo dos SLIs, o qual é realizado durante a operação requerida pelo Consumidor. Esse atraso na Arquitetura Básica é denominado e_1 e na Arquitetura Auditor Independente é denominado e_2. Como o tempo gasto para efetuar a auditoria de SLAs na Arquitetura Básica não interfere no tempo de resposta a uma requisição feita pelo Consumidor (pois é realizada em um momento posterior à operação requerida pelo Cliente), $e_1 = e_2$. Esses atrasos são considerados desprezíveis, pois o tempo de processamento gasto para realizar a medição de um SLI é tipicamente muito menor do que o tempo de processamento gasto com as próprias requisições. Isso é mostrado na seção 5.2, a qual mostra os experimentos realizados.

Na **Arquitetura Inspetor Independente**, o tamanho da amostra é importante, uma vez que requisições adicionais são feitas ao serviço para fins de auditoria. A carga adicional

imposta ao Provedor por essas requisições afetará o desempenho visto pelos Consumidores, e, conseqüentemente, os valores de SLIs calculados incluirão um EIM e_3. Esse valor é maior do que o e_1 e do que o e_2, uma vez que a carga imposta pelas requisições adicionais (requisições pelo próprio serviço) é normalmente muito maior do que a carga imposta por requisições feitas solicitando valores de SLIs.

Na **Arquitetura Decorador Externo**, em que o Inspetor funciona como um decorador para o Provedor, há somente o fator EIM e_4. Como o Inspetor examina todas as requisições enviadas para o Provedor, não há necessidade de analisar o tamanho da amostra. Os valores dos SLIs vistos pelos Consumidores podem ser substancialmente diferentes dos valores que seriam vistos caso o Inspetor não estivesse presente e os Consumidores fizessem as requisições diretamente ao Provedor. Logo, espera-se um valor alto para o erro e_4. Porém, esse erro pode ser bastante reduzido se considerarmos que haverá vários Inspetores distribuídos por localidade e por tráfego do caminho.

Na **Arquitetura Decorador Externo com Desvio** podem ser analisados os dois fatores. Como somente uma amostra das requisições é auditada através do Inspetor, o tamanho da amostra pode ser analisado para reduzir o impacto no desempenho. Pode-se também analisar o EIM e_5, o qual, semelhante ao e_4, é causado pela indireção sofrida pelas requisições. Entretanto, o erro e_5 é apenas uma fração do erro e_4 porque a indireção é sofrida apenas por uma amostra das requisições, ou seja, $e_5 = \frac{n}{N} e_4$, em que n é o tamanho da amostra (determinado para garantir um dado intervalo de confiança [29]) e N é o tamanho da população (o número total de requisições realizadas durante o período de observação).

Na **Arquitetura Sniffers**, o qual usa *sniffers* para capturar passivamente os pacotes na rede, o fator tamanho da amostra não está presente porque todas as requisições vêm de um Consumidor real e não há EIM ($e_6 = 0$), uma vez que o desempenho do serviço não é afetado pela auditoria. Apesar de a criptografia ser necessária para assinar digitalmente e ocultar as informações dos *sniffers*, ela não influencia no tempo de resposta à requisição, uma vez que só será usada na hora de o Inspetor solicitá-las para realizar o cálculo dos SLIs.

Na **Arquitetura Decoradores nos Hosts**, que possui um Inspetor no *host* do Consumidor e outro no *host* do Provedor, o fator tamanho da amostra não está presente porque não há requisições adicionais. Já o fator EIM e_7 está presente devido ao atraso gasto

durante o processo de criptografia e assinatura digital no caso de não haver confiança entre o Provedor e o Consumidor.

Na próxima subseção, o tamanho da amostra é analisado. A subseção 5.1.2 analisa os erros de interferência de medição das sete arquiteturas em comparação à arquitetura em que a auditoria não é realizada.

5.1.1 Análise do tamanho da amostra

Para reduzir a perda no desempenho do serviço, os SLIs podem ser calculados sobre uma amostra das requisições. Existe um *trade-off* entre o *overhead* imposto pela medição e a exatidão desejada para as medidas realizadas. Deseja-se encontrar o menor tamanho de amostra que forneça determinado nível de confiança. Essa análise se aplica ao terceiro e ao quinto cenários.

Como o objetivo deste trabalho é analisar o desempenho do Provedor em resposta às requisições dos Consumidores, foi escolhido o SLI tempo de resposta como variável aleatória e os eventos da análise são as requisições vindas dos Consumidores.

No SLA, foi definido o SLO tempo de resposta médio, o qual é calculado e avaliado baseado em um conjunto de valores do SLI tempo de resposta durante um intervalo de tempo também definido no SLA. Ou seja, durante esse intervalo de tempo, são coletadas amostras de valores de tempo de resposta, calculada a média do tempo de resposta e esta comparada ao limite do SLO definido no SLA.

Como não é possível calcular uma estimativa perfeita da média da população do tempo de resposta através de uma amostra, o SLA define um nível de confiança $100(1-\alpha)\%$ tal que a probabilidade da média da população do tempo de resposta (μ) estar em um intervalo (c_1, c_2) seja $(1-\alpha)$ [30], ou seja:

$$\text{Probabilidade}\{c_1 \le \mu \le c_2\} = 1-\alpha$$

Esse intervalo é chamado intervalo de confiança para a média da população e, supondo que as observações vêm de uma população distribuída de forma normal, ele é dado usando a distribuição t de *Student* [30]:

$$\left(\overline{T} - t_{[1-\alpha/2;n-1]} s/\sqrt{n}, \overline{T} + t_{[1-\alpha/2;n-1]} s/\sqrt{n}\right)$$

Aqui, $\overline{T}$ é a média da amostra do tempo de resposta, s é o desvio padrão da amostra, n é o tamanho da amostra e $t_{[1-\alpha/2;n-1]}$ é o percentil $\left(1-\alpha/2\right)$ de uma variável *t* com $n-1$ graus de liberdade.

O valor para $t_{[1-\alpha/2;n-1]}$ é um número fixo que depende do nível de confiança e dos graus de liberdade. O SLA pode definir um valor para s ou um valor para o erro máximo aceitável $\varepsilon_{\max}$, que substitui $t_{[1-\alpha/2;n-1]}s/\sqrt{n}$:

$$\varepsilon_{\max} = t_{[1-\alpha/2;n-1]}s/\sqrt{n}$$

O intervalo de confiança é expresso em termos do erro máximo aceitável como sendo $\left(\overline{T}-\varepsilon_{\max},\overline{T}+\varepsilon_{\max}\right)$.

Se o SLA estabelece o erro máximo aceitável e o nível de confiança, pode-se encontrar o tamanho mínimo da amostra n para obter o tempo de resposta médio. De acordo com o Teorema Central do Limite, quando o tamanho da amostra *n* é grande o suficiente, a distribuição da média da amostra é aproximadamente normal e o tamanho da amostra é calculado como sendo:

$$n = \frac{\left(z_{1-\alpha/2}\right)^2 s^2}{\varepsilon_{\max}{}^2}$$

Aqui, $z_{1-\alpha/2}$ é o percentil $\left(1-\alpha/2\right)$ de uma variável normal padronizada (a qual tem média 0 e desvio padrão 1). Note que é necessária uma estimativa preliminar da variância $s_0{}^2$. Para obter essa estimativa, pode-se considerar um conjunto piloto de amostras e usá-lo para calcular o tamanho da amostra em um segundo momento [31].

A cada intervalo de tempo de avaliação do SLO, pode-se fazer uma correção no valor inicial da variância de acordo com a equação abaixo e fazer um teste de hipóteses para aceitar ou rejeitar esse valor:

$$s^2 = \frac{\sum_{i-1}^{n} T_i{}^2 - \left(\sum_{i=1}^{n} T_i\right)^2}{n-1}$$

Nesse caso, o teste de hipóteses pode ser feito através de uma carta de controle. Os testes são:

$$H_0 : s^2 = {s_0}^2$$

$$H_1 : s^2 \neq {s_0}^2$$

Aqui, s^2 é o valor da variância obtido durante a atual avaliação e ${s_0}^2$ é o valor da variância obtido na avaliação anterior.

A carta de controle possui dois limites fixos *a* e *b* dependendo do nível de confiança definido no SLA. Na carta de controle usada para corrigir o valor da variância, a distribuição Qui-Quadrada é usada. Se o valor s^2 obtido estiver entre *a* e *b*, ou seja, $a \leq s^2 \leq b$, H_0 é aceito, H_1 é rejeitado e o valor da variância que será usado na próxima avaliação continuará o mesmo. Se o valor da variância estiver fora da faixa, H_0 é rejeitado, H_1 é aceito e a próxima avaliação usará o valor da variância obtido na avaliação atual.

5.1.2 Análise do Erro de Interferência de Medição (EIM)

O termo Erro de Interferência de Medição (EIM) foi usado para indicar a diferença causada em um valor medido (SLI) devido ao método de observação usado (neste caso, devido à auditoria). Ou seja, ao medir o valor de um parâmetro, introduz-se um instrumento para fazer a medição. O próprio instrumento modifica o valor a ser observado. Aqui analisamos o EIM causado pela auditoria no SLI tempo de resposta. Esse erro se apresenta de forma diferente para as várias arquiteturas apresentadas. Nesta seção, é feita uma análise comparativa desse erro entre as arquiteturas.

Para analisar os erros de interferência de medição em cada arquitetura, foram definidos alguns *timestamps* e alguns intervalos de tempo, calculados os tempos de resposta de cada arquitetura de auditoria e comparados ao tempo de resposta da arquitetura em que não é realizada a auditoria, a qual foi denominada **Arquitetura sem Auditoria** (ver Figura 12).

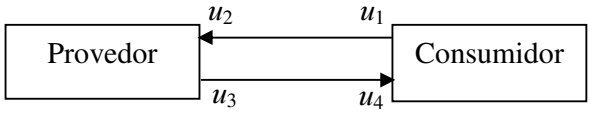

Figura 12: Arquitetura sem Auditoria.

Aqui são considerados os *timestamps* u_1, u_2, u_3, u_4 e os intervalos de tempo δ_p, δ_{C-P}. O intervalo δ_p representa o tempo de processamento da requisição no Provedor e é calculado

como sendo o tempo gasto desde a recepção da requisição pelo Provedor até o momento do envio da resposta, ou seja, $\delta_p = u_3 - u_2$. O intervalo δ_{C-P} representa a soma dos atrasos de ida da requisição e de volta da resposta gastos durante seu tráfego na rede entre o Consumidor e o Provedor:

$$\delta_{C-P} = (u_2 - u_1) + (u_4 - u_3)$$

Como os *timestamps* são medidos no nível de aplicação do *Web Service*, o intervalo de tempo δ_p assume *timestamps* obtidos nesse nível. O tempo gasto nas camadas de protocolos abaixo do nível de aplicação é considerado como parte do atraso em rede.

Nesse caso, quando o Consumidor envia uma requisição diretamente ao Provedor no tempo u_1, o tempo em que o Consumidor recebe a resposta pode ser calculado como:

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

A seguir analisamos os EIMs das sete arquiteturas, com base na Arquitetura sem Auditoria.

Arquitetura Básica:

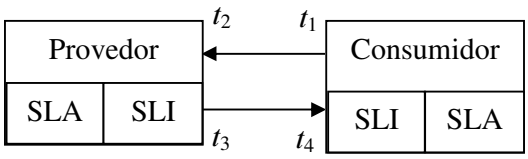

Figura 13: Arquitetura Básica.

Nessa arquitetura (ver Figura 13), os *timestamps* são t_1, t_2, t_3, t_4. Os intervalos de tempo são $\delta_p, \delta_{C-P}, \delta_{SLI}, \delta_{SLA}$. Os dois primeiros são semelhantes aos já mencionados na Arquitetura sem Auditoria. O intervalo δ_{SLI} é o atraso devido à instrumentação dos SLIs e o δ_{SLA} é o atraso devido à auditoria do SLA. O atraso δ_{SLA} não é considerado para o cálculo do erro de interferência de medição (EIM) e_1, porque ele não ocorre durante o atendimento da requisição e sim em um momento posterior e, logo, não influencia no tempo de resposta à requisição visto pelo Consumidor.

O tempo em que o Consumidor recebe a resposta à sua requisição é calculado da seguinte maneira:

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

Aqui, o EIM é a diferença entre o tempo gasto pela requisição na Arquitetura Básica e o tempo gasto na Arquitetura sem Auditoria.

$e_1 = t_4 - u_4$

Em todas as análises, considera-se que o tempo inicial é zero. Portanto, $u_1 = t_1 = 0$

Logo, $e_1 = \delta_{SLI}$.

Arquitetura Auditor Independente:

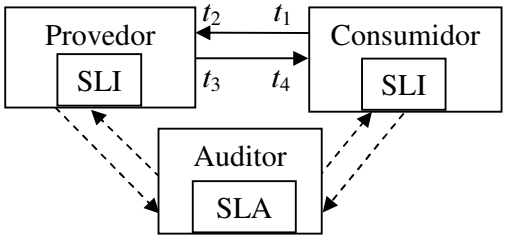

Figura 14: Arquitetura Auditor Independente.

Nessa arquitetura (ver Figura 14), os *timestamps* são t_1, t_2, t_3, t_4. Os intervalos de tempo são $\delta_p, \delta_{C-P}, \delta_{SLI}$.

O tempo em que o Consumidor recebe a resposta à sua requisição é calculado da mesma forma que na Arquitetura Básica:

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

Aqui, o EIM, e_2, é a diferença entre o tempo gasto pela requisição na Arquitetura Auditor Independente e o tempo gasto na Arquitetura sem Auditoria.

$e_2 = t_4 - u_4$

Logo:

$e_2 = \delta_{SLI}$

Arquitetura Inspetor Independente:

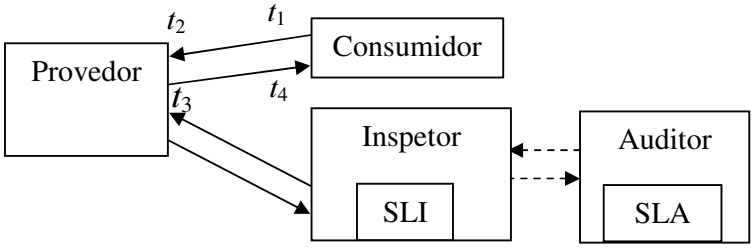

Figura 15: Arquitetura Inspetor Independente.

Nessa arquitetura (ver Figura 15), os *timestamps* são t_1, t_2, t_3, t_4. Os intervalos de tempo são $\delta_p, \delta_{C-P}, \delta_m$. O intervalo δ_m é o atraso marginal, que é o atraso no tempo de processamento no Provedor δ_p provocado pelas requisições adicionais efetuadas pelos Inspetores.

O tempo em que o Consumidor recebe a resposta à sua requisição é calculado da seguinte maneira:

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

Aqui, o EIM é a diferença entre o tempo gasto pela requisição na Arquitetura Inspetor Independente e o tempo gasto na Arquitetura sem Auditoria.

$$e_3 = t_4 - u_4$$

Logo:

$$e_3 = \delta_m$$

O atraso marginal médio $\overline{\delta_m}$ é o aumento no tempo de processamento médio $\overline{\delta_p}$ provocado pelo acréscimo de requisições enviadas pelo Inspetor, ou seja, é a diferença no tempo de processamento médio ao se adicionar um Inspetor. Aqui λ_c é a taxa média de requisições enviadas por todos os Consumidores ao Provedor e λ_i é a taxa média de requisições enviadas pelo Inspetor ao Provedor. O atraso marginal médio é calculado usando a seguinte fórmula:

$$\overline{\delta_m} = \overline{\delta_p}(\lambda_c + \lambda_i) - \overline{\delta_p}(\lambda_c)$$

Para analisar o comportamento do $\overline{\delta_m}$, pode-se fazer uma comparação com um sistema de filas (ver Figura 16). As requisições dos Consumidores e do Inspetor são os fregueses que chegam ao Provedor, onde ficam em fila esperando para serem atendidos pelo serviço. O tempo médio gasto por uma requisição no Provedor $\overline{\delta_p}$ é a soma do tempo médio de espera em fila $\overline{W}$ e do tempo médio de serviço $\overline{X}$.

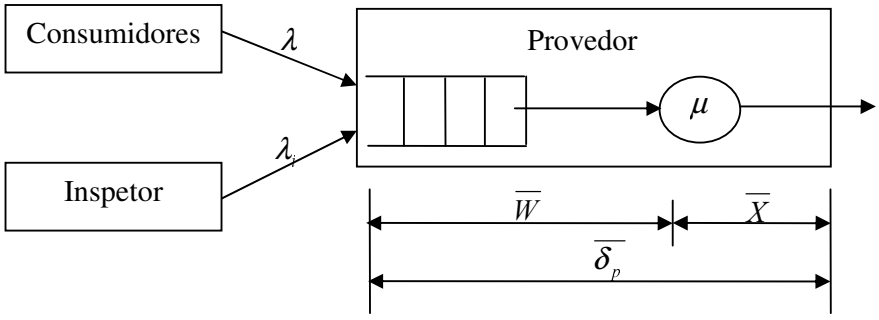

Figura 16: Modelo de filas

Aqui, λ_c e λ_i são as taxas médias de requisições enviadas pelos Consumidores e pelo Inspetor, respectivamente, e são expressas em requisições por unidade de tempo. O valor μ é a taxa média de serviço e é expressa também por requisições por unidade de tempo.

O sistema em questão foi considerado um sistema M/G/1, em que os tempos de inter-chegada são exponencialmente distribuídos, os tempos de serviço seguem uma distribuição geral e há um único servidor [32], no qual o tempo de serviço médio é $\overline{X} = \frac{1}{\mu}$.

O tempo de serviço médio segue uma distribuição com média $\overline{X}$ e segundo momento $\overline{X^2}$. O tempo médio gasto por uma requisição no Provedor $\overline{\delta_p}(\lambda)$ pode ser calculado pela fórmula de Pollaczek-Khinchin [32]:

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

Quando a taxa total de requisições é λ , tem-se:

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

Quando a taxa total de requisições é $\lambda_c + \lambda_i$, tem-se:

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

Aqui ρ_c ou ρ_{ci} é a utilização do servidor e varia de 0 a 1. No caso particular em que os tempos de serviço são constantes (uma aproximação da realidade para serviços de transação), o segundo momento do tempo de serviço é $\overline{X}^2 = \frac{1}{\mu^2}$. Logo, tem-se:

$$e_3 = \delta_m = \delta_P(\lambda_c + \lambda_i) - \delta_P(\lambda_c) = \frac{(\lambda_c + \lambda_i)}{2\mu^2(1-\rho_{ci})} - \frac{\lambda_c}{2\mu^2(1-\rho_c)}$$

Arquitetura Decorador Externo:

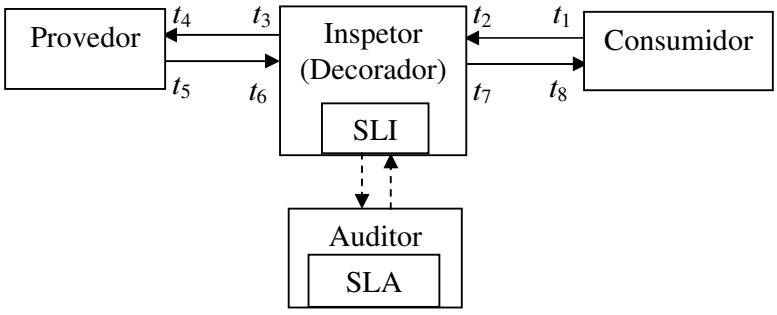

Figura 17: Arquitetura Decorador Externo.

Nessa arquitetura (ver Figura 17), os *timestamps* são $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8$. Os intervalos de tempo são $\delta_p, \delta_{C-I}, \delta_{I-P}, \delta_f, \delta_r$. O intervalo δ_{C-I} representa o atraso em rede

total entre Consumidor e Inspetor e δ_{I-P} representa o atraso em rede total entre Inspetor e Provedor. O intervalo δ_f é o tempo gasto pelo Inspetor para repassar (*forward*) a requisição para o Provedor e é calculado como $\delta_f = t_3 - t_2$. O intervalo de tempo δ_r é o tempo gasto pelo Inspetor para retornar a resposta para o Consumidor e é calculado como $\delta_r = t_7 - t_6$.

O tempo em que o Consumidor recebe a resposta à sua requisição é calculado da seguinte maneira:

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

Aqui, o EIM é a diferença entre o tempo gasto pela requisição na Arquitetura Decorador Externo e o tempo gasto na Arquitetura sem Auditoria.

$$e_4 = t_8 - u_4$$

Logo:

$$e_4 = \delta_f + \delta_r + \delta_{C-I} + \delta_{I-P} - \delta_{C-P}$$

Arquitetura Decorador Externo com Desvio:

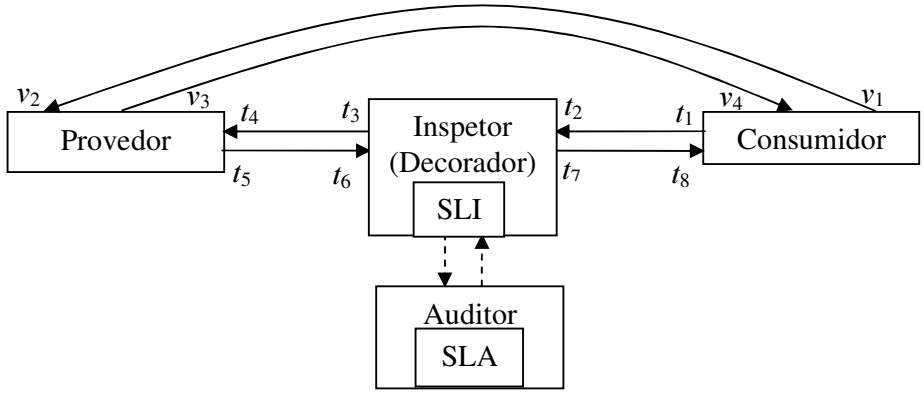

Figura 18: Arquitetura Decorador Externo com Desvio.

Nessa arquitetura (ver Figura 18), os *timestamps* são $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8$ (medidos durante o tráfego de uma amostra n de requisições) e v_1, v_2, v_3, v_4 (medidos durante o tráfego do restante das requisições totais N, ou seja, $N - n$ requisições).

Como foi visto na seção 5.1, o erro de interferência de medição para esta arquitetura pode ser calculado como sendo:

$e_5 = \frac{n}{N} e_4$, ou seja:

$$e_5 = \frac{n}{N}\left(\delta_f + \delta_r + \delta_{C-I} + \delta_{I-P} - \delta_{C-P}\right)$$

Arquitetura Sniffers:

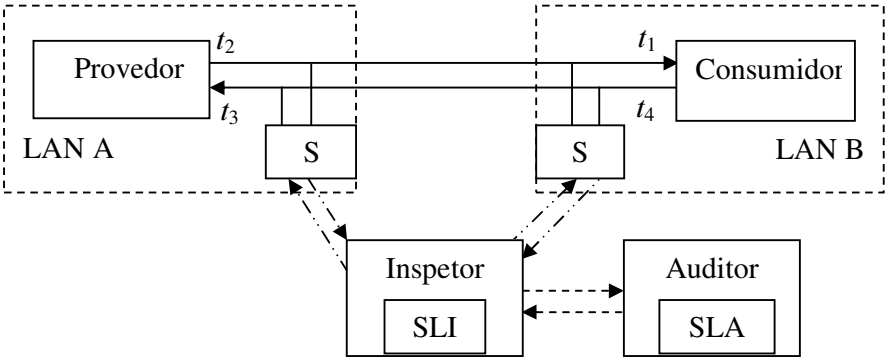

Figura 19: Arquitetura Sniffers.

Nessa arquitetura (ver Figura 19), os *timestamps* são t_1, t_2, t_3, t_4 e os intervalos de tempo são δ_p, δ_{C-P}.

O tempo em que o Consumidor recebe a resposta à sua requisição é calculado da seguinte maneira:

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

Aqui, o EIM é a diferença entre o tempo gasto pela requisição na Arquitetura Sniffers e o tempo gasto na Arquitetura sem Auditoria.

$$e_6 = t_4 - u_4$$

Logo:

$$e_6 = 0$$

Arquitetura Decoradores nos Hosts:

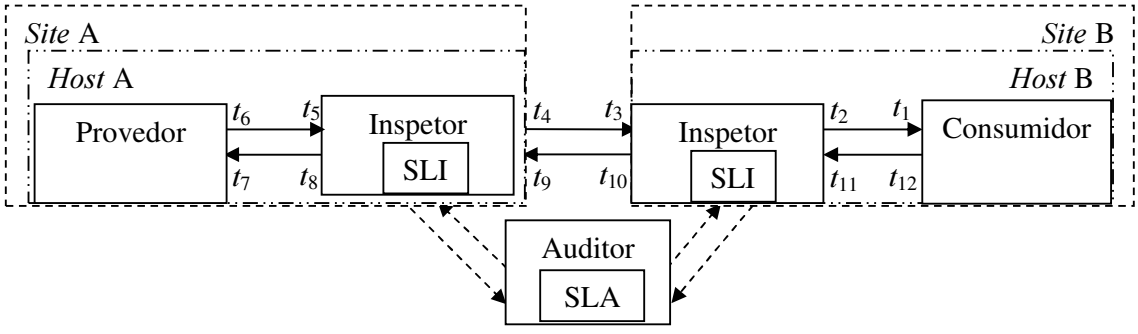

Figura 20: Arquitetura Decoradores nos Hosts.

Nessa arquitetura (ver Figura 20), os *timestamps* são $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}$ e os intervalos de tempo são $\delta_p, \delta_{C-I}, \delta_{I-I}, \delta_{I-P}, \delta_f, \delta_r, \delta_{cript}$. O intervalo δ_{cript} é o atraso devido ao tempo gasto para criptografar as funções e os resultados dos cálculos dos SLIs no

caso em que não há confiança entre o Provedor e o Consumidor. O intervalo δ_{I-I} é o atraso em rede total entre ambos os Inspetores.

O tempo em que o Consumidor recebe a resposta à sua requisição é calculado da seguinte maneira:

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

Os atrasos δ_{C-I} e δ_{I-P} são os atrasos de rede juntamente com o atraso gasto na pilha de protocolos de um *Web Service*, logo:

$$\delta_{C-I} = \delta_{rede(C-I)} + \delta_{WS}$$

$$\delta_{I-P} = \delta_{rede(I-P)} + \delta_{WS}$$

Como um Inspetor fica localizado no *host* do Consumidor e o outro no *host* do Provedor, o atraso de rede entre um Inspetor e o Consumidor ou Provedor do mesmo *host* é zero, ou seja, $\delta_{rede(C-I)} = \delta_{rede(I-P)} = 0$. Logo, a expressão anterior se resume a:

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

Aqui, o EIM é a diferença entre o tempo gasto pela requisição na Arquitetura Decoradores nos Hosts e o tempo gasto na Arquitetura sem Auditoria.

$$e_7 = t_{12} - u_4$$

Como os Inspetores se localizam nos *hosts* do Consumidor e do Provedor, pode-se considerar que o atraso de rede entre os Inspetores da Arquitetura Decoradores nos Hosts é semelhante ao atraso de rede entre o Consumidor e o Provedor da Arquitetura sem Auditoria, ou seja, $\delta_{I-I} = \delta_{C-P}$. Efetuando-se a subtração, chega-se a seguinte fórmula:

$$e_7 = 2\delta_{WS} + 2\delta_f + 2\delta_{cript} + 2\delta_r$$

Observe que a fórmula anterior se aplica para o caso em que não há confiança entre o Provedor e o Consumidor e que é necessário o uso de criptografia. Porém a Arquitetura Decoradores nos Hosts pode ser usada também em casos em que existe confiança e que não é necessário o uso de criptografia. Nesse último caso, a equação do EIM se reduz a:

$$e_7 = 2\delta_{WS} + 2\delta_f + 2\delta_r$$

5.2. Avaliação Experimental

Esta seção tem o objetivo de tornar os resultados analíticos mais concretos através da medição da perda de desempenho das arquiteturas de auditoria em um ambiente real.

Foi implementado um serviço como Provedor de serviço e um cliente que pode incluir ou excluir livros da sua lista de compras. Os serviços foram implementados sob o *Globus 3.2.1* [16] , implementados em Java, implantados em um Pentium 4, 2.66 GHz de CPU, 512 MB de memória RAM e 40 GB de disco rígido. O *Globus 3.2.1* foi escolhido como ambiente de programação, pois oferece APIs para implementação de *Web Services* e de *Grid Services*.

Experimentos foram realizados para obter valores médios para os parâmetros δ_{SLI}, δ_f, δ_r, δ_{WS}, δ_{cript}, usados nas equações apresentadas na seção 5.1.2 para a análise do erro de interferência de medição EIM de cada arquitetura apresentada.

Os valores obtidos para cada um desses parâmetros são valores médios calculados a partir de amostras coletadas em conjuntos de 20 ($n = 20$), nível de confiança de 95%, erro máximo aceitável de 5% ($\varepsilon_{\max} = 5\%$) e distribuição *t de Student*. Sempre as quatro primeiras amostras são desconsideradas porque os tempos de resposta das requisições iniciais são maiores devido ao estabelecimento da comunicação entre o cliente e o provedor do serviço (período transiente inicial).

Enquanto espera-se que os valores dos parâmetros δ_{SLI}, δ_f, δ_r, δ_S , δ_{crypt} permaneçam o mesmo (para a tecnologia computacional atual), o tempo de serviço e a velocidade da rede variam dependendo do serviço e de onde o Provedor, o Consumidor e o Inspetor estão localizados na rede. O tempo de serviço afeta o EIM da Arquitetura Inspetor Independente e a velocidade da rede afeta tanto a Arquitetura Decorador Externo como a Arquitetura Decorador Externo com Desvio. Do mesmo modo, as arquiteturas que dependem do tamanho da amostra (Arquitetura Inspetor Independente e Arquitetura Decorador Externo com Desvio) são afetadas pela variância do SLI (quanto maior a variância maior é o tamanho da amostra necessária para alcançar um dado intervalo de confiança).

Para dar ao leitor uma maior visão das possibilidades, o EIM foi calculado classificando cada um desses parâmetros (tempo de serviço, velocidade da rede e tamanho da amostra) em “baixo” e “alto”. Mais precisamente, foi considerado como baixo um tempo de serviço médio $\overline{X}$ de 10*ms* e como alto um tempo de serviço médio $\overline{X}$ de 1000*ms*; um atraso de rede baixo de poucos microssegundos (típico de uma LAN) e um atraso de rede alto de 200*ms* (típico de uma WAN); e uma baixa taxa de amostras para auditoria $\frac{n}{N} = 5\%$ e uma alta taxa de amostras para auditoria $\frac{n}{N} = 30\%$.

Para calcular o tempo gasto durante o cálculo do SLI δ_{SLI}, o SLI tempo de resposta de uma operação de adição de um livro na lista de compras do Cliente foi escolhido. Esse atraso depende do tipo de SLI a ser calculado, mas geralmente esse cálculo corresponde a uma operação matemática que leva alguns nanossegundos. O tempo medido em *ms* para o cálculo do SLI tempo de resposta foi:

$$\delta_{SLI} = 0{,}02354ms$$

Visto o que foi exposto na seção 5.1.2, os EIMs da **Arquitetura Básica** e da **Arquitetura Auditor Independente** podem ser calculados pelas seguintes equações:

$$e_1 = \delta_{SLI}$$

$$e_2 = \delta_{SLI}$$

$$\delta_{SLI} = 0{,}02354ms$$

Ou seja, $e_1 = e_2 = 0{,}02354ms$, um tempo irrisório, o qual não causa impacto no desempenho do serviço.

O EIM da **Arquitetura Inspetor Independente** é causado pelo atraso marginal provocado pelas requisições extras vindas do Inspetor e pode ser calculado da seguinte maneira:

$$e_3 = \delta_m$$

$$\overline{\delta_m} = \overline{\delta_p}(\lambda_c + \lambda_i) - \overline{\delta_p}(\lambda_c)$$

Em que:

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

e:

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

Portanto:

$$e_3 = \delta_m = \delta_P(\lambda_c + \lambda_i) - \delta_P(\lambda_c) = \frac{(\lambda_c + \lambda_i)}{2\mu^2(1-\rho_{ci})} - \frac{\lambda_c}{2\mu^2(1-\rho_c)}$$

Assumiu-se uma taxa razoável de requisições submetidas pelos Consumidores de metade da taxa de serviço do Provedor, ou seja:

$$\lambda_c = \frac{\mu}{2}$$

Para um provedor rápido que possui $10ms$ como tempo médio de serviço $\overline{X}$ ($\overline{X} = 0{,}01s$) e uma taxa de amostras baixa ($\frac{n}{N} = 5\%$):

Uma taxa de amostras para auditoria de 5% significa que a taxa de requisições submetidas pelo Inspetor λ_i é 5% da taxa de requisições submetidas pelos Consumidores λ_c, ou seja:

$$\lambda_i = 0{,}05\lambda_c$$

Como $\lambda_c = \frac{\mu}{2}$:

$$\lambda_i = 0{,}05\frac{\mu}{2} \therefore \lambda_i = 0{,}025\mu$$

O μ é calculado da seguinte maneira:

$$\overline{X} = \frac{1}{\mu} \therefore \mu = \frac{1}{\overline{X}} = \frac{1}{0{,}01} \therefore \mu = 100$$

Calculando o tempo médio gasto no Provedor pelas requisições dos Consumidores $\overline{\delta_p}(\lambda_c)$:

$$\lambda_c = \frac{\mu}{2} = \frac{100}{2} \therefore \lambda_c = 50$$

$$\rho_c = \lambda_c \overline{X} = 50 \times 0{,}01 \therefore \rho_c = 0{,}5$$

$$\overline{\delta_p}(\lambda_c) = \overline{X} + \frac{\lambda_c \overline{X^2}}{2(1-\rho_c)} = 0{,}01 + \frac{(50 \times 0{,}0001)}{2(1-0{,}5)} = 0{,}01 + \frac{0{,}005}{1} = 0{,}015s \therefore \overline{\delta_p}(\lambda_c) = 15ms$$

Calculando o tempo médio gasto no Provedor por todas as requisições vindas tanto dos Consumidores como do Inspetor ($\overline{\delta_P}(\lambda_c + \lambda_i)$):

$$\lambda_i = 0{,}025\mu = 0{,}025 \times 100 \therefore \lambda_i = 2{,}5$$

$$\lambda_c + \lambda_i = 50 + 2{,}5 \therefore \lambda_c + \lambda_i = 52{,}5$$

$$\rho_{ci} = (\lambda_c + \lambda_i)\overline{X} = 52{,}5 \times 0{,}01 \therefore \rho_{ci} = 0{,}525$$

$$\overline{\delta_P}(\lambda_c + \lambda_i) = \overline{X} + \frac{(\lambda_c + \lambda_i)\overline{X^2}}{2(1-\rho_{ci})} = 0{,}01 + \frac{(52{,}5 \times 0{,}0001)}{2(1-0{,}525)} = 0{,}01 + \frac{0{,}00525}{0{,}95} =$$

$$\overline{\delta_P}(\lambda_c + \lambda_i) = 0{,}0155263s = 15{,}5263ms$$

Calculando o EIM e_3 para um provedor rápido e uma taxa de amostras baixa:

$$e_3 = \delta_m = \delta_P(\lambda_c + \lambda_i) - \delta_P(\lambda_c) = 15{,}5263 - 15 \therefore e_3 = 0{,}5263ms$$

Para um provedor rápido que possui $10ms$ como tempo médio de serviço $\overline{X}$ ($\overline{X} = 0{,}01s$) e uma taxa de amostras alta ($\frac{n}{N} = 30\%$):

Uma taxa de amostras para auditoria de 30% significa que a taxa de requisições submetidas pelo Inspetor λ_i é 30% da taxa de requisições submetidas pelos Consumidores λ_c, ou seja:

$$\lambda_i = 0{,}3\lambda_c$$

Como $\lambda_c = \frac{\mu}{2}$:

$$\lambda_i = 0{,}3\frac{\mu}{2} \therefore \lambda_i = 0{,}15\mu$$

Para um provedor rápido, temos $\mu = 100$, $\lambda_c = 50$, $\rho_c = 0{,}5$ e $\overline{\delta_p}(\lambda_c) = 15ms$.

Calculando o tempo médio gasto no Provedor por todas as requisições vindas tanto dos Consumidores como do Inspetor ($\overline{\delta_P}(\lambda_c + \lambda_i)$):

$$\lambda_i = 0{,}15\mu = 0{,}15 \times 100 \therefore \lambda_i = 15$$

$$\lambda_c + \lambda_i = 50 + 15 \therefore \lambda_c + \lambda_i = 65$$

$$\rho_{ci} = (\lambda_c + \lambda_i)\overline{X} = 65 \times 0{,}01 \therefore \rho_{ci} = 0{,}65$$

$$\overline{\delta_P}(\lambda_c + \lambda_i) = \overline{X} + \frac{(\lambda_c + \lambda_i)\overline{X^2}}{2(1-\rho_{ci})} = 0{,}01 + \frac{(65 \times 0{,}0001)}{2(1-0{,}65)} = 0{,}01 + \frac{0{,}0065}{0{,}7} =$$

$$\overline{\delta_P}(\lambda_c + \lambda_i) = 0{,}0192857s = 19{,}2857ms$$

Calculando o EIM e_3 para um provedor rápido e uma taxa de amostras alta:

$$e_3 = \delta_m = \delta_P(\lambda_c + \lambda_i) - \delta_P(\lambda_c) = 19{,}2857 - 15 \therefore e_3 = 4{,}285ms$$

Para um provedor lento que possui $1000ms$ como tempo médio de serviço $\overline{X}$ ($\overline{X} = 1s$) e uma taxa de amostras baixa ($\frac{n}{N} = 5\%$), o cálculo do EIM é semelhante ao do provedor rápido com uma taxa de amostras baixa. O valor do EIM para um provedor lento e uma taxa de amostras baixa é $e_3 = 52{,}63ms$.

Para um provedor lento, que possui $1000ms$ como tempo médio de serviço $\overline{X}$ ($\overline{X} = 1s$) e uma taxa de amostras alta ($\frac{n}{N} = 30\%$), o cálculo do EIM é semelhante ao do

provedor rápido com uma taxa de amostras alta. O valor do EIM para um provedor lento e uma taxa de amostras alta é $e_3 = 428{,}5ms$.

O EIM da **Arquitetura Decorador Externo** é calculado através da seguinte equação:

$$e_4 = \delta_f + \delta_r + \delta_{C-I} + \delta_{I-P} - \delta_{C-P}$$

Para calcular o tempo gasto no repasse (*forward*) da requisição δ_f e no retorno da resposta δ_r pelo Inspetor, foi implementado um serviço de Inspetor funcionando como decorador do Provedor. Os tempos calculados em *ms* foram:

$$\delta_f = 0ms$$

$$\delta_r = 0{,}5ms$$

O atraso de retorno δ_r foi maior devido ao cálculo do SLI ser feito após a chamada do método do provedor.

Os atrasos $\delta_{C-I}, \delta_{I-P}, \delta_{C-P}$ podem ser estimados considerando-se a soma dos atrasos gastos até o nível de rede (estimados através do comando *ping*) e dos atrasos gastos na pilha de protocolos de um *Web Service* (δ_{WS}):

$$\delta_{C-I} = \delta_{rede(C-I)} + \delta_{ws}$$

$$\delta_{I-P} = \delta_{rede(I-P)} + \delta_{ws}$$

$$\delta_{C-P} = \delta_{rede(C-P)} + \delta_{ws}$$

Para obter valores típicos para o atraso gasto na pilha de protocolos de um *Web Service* δ_{WS}, primeiramente foi implementada a operação *WS-ping* no Consumidor. A operação *WS-ping* é realizada da seguinte maneira: o Provedor e o Cliente são implantados em *hosts* diferentes em uma LAN; o Provedor oferece uma operação bem simples *toUpperCase()* que converte uma *String* minúscula em maiúscula; é medido o tempo desde a requisição da operação *toUpperCase()* pelo Cliente até a obtenção da resposta. A operação *toUpperCase()* foi escolhida, pois o tempo gasto por ela é irrisório.

Em seguida à obtenção do valor do tempo gasto durante uma operação *WS-ping*, foi executado o comando *ping* tradicional do Unix entre os dois *hosts*. O valor médio obtido para o tempo gasto na pilha de protocolos de um *Web Service* δ_{WS} é uma média das diferenças entre o tempo de resposta da operação *WS-ping* e o tempo do comando *ping*:

$$\delta_{ws} = 10ms$$

Fazendo-se uma suposição razoável de que o tempo médio gasto por um pacote para fazer o caminho de ida e volta na rede LAN é de alguns microssegundos (aqui considerado

como $0ms$) e de que na rede WAN é de $200ms$, tem-se os seguintes cálculos para o EIM da Arquitetura Decorador Externo:

$$e_4 = \delta_f + \delta_r + \delta_{C-I} + \delta_{I-P} - \delta_{C-P}$$

Para serviços em uma LAN:

$$e_4 = 0 + 0{,}5 + 10 + 10 - 10 = 10{,}5ms$$

Para serviços em uma WAN:

$$e_4 = 0 + 0{,}5 + 210 + 210 - 210 = 210{,}5ms$$

Para o cálculo do EIM da **Arquitetura Decorador Externo com Desvio**, usa-se a seguinte fórmula:

$$e_5 = \frac{n}{N} e_4$$

Para serviços em uma LAN e uma taxa de amostras baixa ($\frac{n}{N} = 5\%$):

$$e_5 = \frac{n}{N} 10{,}5ms = 0{,}05 \times 10{,}5 \therefore e_5 = 0{,}525ms$$

Para serviços em uma LAN e uma taxa de amostras alta ($\frac{n}{N} = 30\%$):

$$e_5 = \frac{n}{N} 10{,}5ms = 0{,}3 \times 10{,}5 \therefore e_5 = 3{,}15ms$$

Para serviços em uma WAN e uma taxa de amostras baixa ($\frac{n}{N} = 5\%$):

$$e_5 = \frac{n}{N} 210{,}5ms = 0{,}05 \times 210{,}5 \therefore e_5 = 10{,}525ms$$

Para serviços em uma WAN e uma taxa de amostras alta ($\frac{n}{N} = 30\%$):

$$e_5 = \frac{n}{N} 210{,}5ms = 0{,}3 \times 210{,}5 \therefore e_5 = 63{,}15ms$$

Como foi visto anteriormente, o EIM para a **Arquitetura Sniffers** é zero:

$$e_6 = 0$$

O EIM para a **Arquitetura Decoradores nos Hosts** para o caso em que não há confiança entre as partes assinantes é calculado através da seguinte fórmula:

$$e_7 = 2\delta_{WS} + 2\delta_f + 2\delta_{cript} + 2\delta_r$$

Os atrasos $\delta_{WS}, \delta_f, \delta_r$ já foram medidos para a Arquitetura Decorador Externo e são:

$$\delta_f = 0ms;\ \delta_r = 0{,}5ms;\ \delta_{ws} = 10ms$$

Para se ter uma idéia do tempo gasto com criptografia, foram implementados dois serviços no *Globus*, um serviço não seguro e um serviço seguro que usa criptografia para enviar as mensagens criptografadas. Daí foi implementado um cliente para cada serviço e medido o tempo de resposta a uma requisição em cada um desses clientes. O atraso causado pela criptografia δ_{cript} é a diferença entre o tempo de resposta proporcionado pelo serviço seguro e o tempo de resposta proporcionado pelo serviço não seguro. O tempo medido em *ms* foi:

$$\delta_{cript} = 130{,}3ms$$

De posse dos valores para os atrasos, o valor do EIM da Arquitetura Decoradores nos Hosts, no caso em que não há confiança entre os serviços, pode ser calculado como:

$$e_7 = 20 + 2\delta_f + 2\delta_{cript} + 2\delta_r$$

$$e_7 = 20 + 0 + 260{,}6 + 1 = 281{,}6ms$$

O EIM da Arquitetura Decoradores nos Hosts para o caso em que existe confiança entre Provedor e Consumidor, pode ser calculado como sendo:

$$e_7 = 20 + 0 + 1 = 21ms$$

A Tabela 2 resume os valores calculados para os erros de interferência de medição das sete arquiteturas:

Tabela 2: Analisando o Erro de Interferência de Medição EIM das sete arquiteturas.

Arquiteturas		**Pequena Amostragem (5%)**	**Grande Amostragem (30%)**
1. Básica		0,02354*ms*	0,02354*ms*
2. Auditor Independente		0,02354*ms*	0,02354*ms*
3. Inspetor Independente	**Provedor Rápido (10*ms*)**	0,5263*ms*	4,285*ms*
	Provedor Lento (1000*ms*)	52,63*ms*	428,5*ms*
4. Decorador Externo	**LAN (0*ms*)**	10,5*ms*	10,5*ms*
	WAN (200*ms*)	210,5*ms*	210,5*ms*
5. Decorador Externo com Desvio	**LAN (0*ms*)**	0,525*ms*	3,15*ms*
	WAN (200*ms*)	10,53*ms*	63,15*ms*
6. Sniffers		0*ms*	0*ms*
7. Decoradores nos Hosts	**Com Confiança**	21*ms*	21*ms*
	Sem Confiança	281.6*ms*	281.6*ms*

Através da Tabela 2, percebe-se que o erro de interferência de medição das Arquiteturas Básica, Auditor Independente e Sniffers são irrisórios ou nulos. O maior EIM do estudo (428,5*ms*) ocorre na Arquitetura Inspetor Independente quando ela demanda uma

grande amostragem a partir de um provedor lento. Contudo, quando a amostragem é pequena e especialmente quando o provedor é rápido essa arquitetura apresenta um bom desempenho. O EIM para a Arquitetura Decorador Externo depende da proximidade entre os serviços. Logo, apresenta um desempenho muito melhor em uma LAN do que em uma WAN. Também como esperado, o EIM reduz com o uso do desvio na Arquitetura Decorador Externo com Desvio. Já a Arquitetura Decoradores nos Hosts apresenta um EIM bem mais alto quando não há confiança entre os serviços do que quando há confiança, uma vez que necessita de duas rodadas de criptografia em seus dois Inspetores no primeiro caso.

Capítulo 6
Conclusões e Trabalhos Futuros

Como *Web Services* e *Grid Services* são tecnologias para criação de aplicações distribuídas independentes de linguagem de programação e de plataforma e que permitem a comunicação entre softwares pertencentes a domínios administrativos diferentes, eles vêm se tornando tecnologias cada vez mais utilizadas para a interação entre aplicações de diversas empresas com negócios diferentes.

Com a proliferação desses serviços, que podem se comunicar com vários outros serviços desconhecidos até o momento da sua descoberta e que podem trocar de parceiros dinamicamente, a necessidade de estabelecer e garantir contratos definindo níveis de QoS e a gerência desses contratos de maneira automática e dinâmica se tornam essenciais e relevantes para permitir maiores compromissos entre os serviços que desejam estabelecer comunicações uns com os outros. Esses contratos são chamados de acordos de nível de serviço ou SLAs.

A gerência de SLA compreende as fases de: (i) negociação e estabelecimento do SLA; (ii) auditoria de SLA, que inclui a instrumentação dos serviços para obtenção de valores para os índices de nível de serviço ou SLIs e a avaliação dos objetivos dos níveis de serviço ou SLOs; (iii) notificação do não cumprimento do SLA para as partes envolvidas e interessadas; e (iv) a tomada de ações de gerência com o intuito de fazer correções no serviço para evitar novos descumprimentos ou aplicar penalidades aos serviços não cumpridores do SLA.

Embora existam trabalhos que abordem a automatização da gerência de SLAs para serviços, tais como [1], [2], [3] e [4], eles não detalham a fase de auditoria de SLAs.

Esta dissertação focou a fase de auditoria de SLAs para *Web* e *Grid Services*. Foram apresentadas sete arquiteturas possíveis para a realização do processo de auditoria, assim como vantagens, desvantagens e uma análise de desempenho dessas arquiteturas. As arquiteturas foram comparadas em relação a alguns aspectos como: intrusividade, confiança necessária, uso de requisições extras, possibilidade de tratamento preferencial, auditoria de carga do consumidor e auditoria de mensagens criptografadas. Na análise de desempenho, foram descritos dois fatores que diminuem o desempenho dos serviços visto pelos clientes devido ao processo de auditoria. Um deles é o tamanho do conjunto de amostras das requisições extras obtidas pelo Provedor escolhidas para calcular os valores dos SLIs. Quanto

maior o tamanho desse conjunto de amostras maior é o impacto no desempenho do serviço. Um outro fator apresentado foi o Erro de Interferência de Medição (EIM), ou seja, o erro introduzido no valor medido devido ao próprio método de observação. Esses dois fatores foram analisados para as sete arquiteturas apresentadas e os EIMs foram analisados mais precisamente através de experimentos realizados com a implementação de algumas dessas arquiteturas.

Como resultado desta dissertação, pode-se concluir que cada arquitetura tem suas vantagens e desvantagens e é mais apropriada para uma determinada situação.

No caso em que existe confiança entre as partes assinantes do SLA e essas partes não fazem questão de implementar código referente à instrumentação e auditoria, então a Arquitetura Básica e a Arquitetura Auditor Independente são ótimas opções. No caso em que existe confiança, porém os serviços não desejam introduzir funções de instrumentação e auditoria nos seus códigos, mas não se incomodam em ter introduzido código de terceiros para fins de auditoria em seus *hosts*, a Arquitetura Decoradores nos Hosts pode ser escolhida, porém com uma perda de desempenho de aproximadamente 300ms (em atuais ambientes computacionais típicos).

Já no caso em que não existe confiança entre as partes assinantes, a Arquitetura Básica e a Arquitetura Auditor Independente não são apropriadas. A Arquitetura Decoradores nos Hosts deve implementar mecanismos que garantam que nem os códigos de terceiros e nem os resultados obtidos sejam modificados pelos *hosts* das partes assinantes que os hospedam e requer a instalação de *hardware* especial no Provedor e no Consumidor, o que pode ser um grande obstáculo para uma larga implantação. As Arquiteturas Inspetor Independente, Decorador Externo, Decorador Externo com Desvio e Sniffers também podem ser escolhidas no caso de não haver confiança entre as partes.

A Arquitetura Inspetor Independente é atrativa porque não interfere no caminho real do serviço entre o Consumidor e o Provedor. Por outro lado, ela cria problemas relacionados à adição de carga ao Provedor, aos efeitos colaterais na base de dados do Provedor e à possibilidade de tratamento preferencial às requisições que são usadas para avaliar o SLA. Ela pode ser escolhida caso as requisições falsas do Inspetor não causem efeitos colaterais indesejáveis. Para que essa arquitetura funcione é necessária a implantação de um método que não permita que o Provedor possa identificar se as requisições vêm do Consumidor real ou do Inspetor e um mecanismo como o uso de tíquetes para o cálculo da carga do Consumidor.

A Arquitetura Decorador Externo é uma boa opção para casos em que não há confiança entre os serviços assinantes, uma vez que os serviços não precisam introduzir

funções de auditoria nos seus próprios códigos e nem nos seus próprios *hosts*. Nessa arquitetura não existem problemas causados por requisições falsas e nem a possibilidade de tratamento preferencial. Porém, ela pode impor um pesado aumento no tempo de resposta devido à auditoria dependendo de onde (na rede) estão o Consumidor, o Provedor e os Inspetores. Caso o serviço de auditoria possa ser distribuído em diversos *sites*, a indireção da rede é menor e, conseqüentemente, o EIM é menor e o desempenho melhora.

Caso o serviço de auditoria não possa ser distribuído e que se deseje um EIM menor, a Arquitetura Decorador Externo com Desvio pode substituir a escolha pela Arquitetura Decorador Externo. Na Arquitetura Decorador Externo com Desvio, o EIM é reduzido com a auditoria apenas de uma amostra de requisições. Porém, ela reintegra (em menor escala) os problemas encontrados na Arquitetura Inspetor Independente. Para utilizar essa arquitetura, é necessária a implantação de mecanismos para evitar que o Provedor possa conceder tratamento preferencial às requisições vindas do Inspetor e um mecanismo como o uso de tíquetes para o cálculo da carga do Consumidor.

A arquitetura Sniffers é uma boa opção para os casos em que as mensagens que trafegam entre as partes assinantes não são criptografadas, pois devido à sua natureza passiva não provoca aumento no tempo de resposta às requisições. Além disso, não é necessário que haja confiança entre as partes assinantes. Já no caso em que a comunicação entre os serviços é feita com criptografia, essa arquitetura pode não ser apropriada pelo fato de não conseguir informações suficientes para calcular valores para alguns SLIs.

A Arquitetura Decoradores nos Hosts é uma excelente opção para a auditoria de partes que confiam umas nas outras e não desejam introduzir funções de auditoria nos seus códigos. O EIM nesse caso é razoável e não há problemas de medição da carga do Consumidor, de tratamento preferencial ou de requisições falsas (efeitos colaterais). Já quando não há confiança entre as partes, é necessário o uso de técnicas para garantir a confidencialidade do código das funções de auditoria e dos resultados destas funções. Foi mostrado que com o uso da criptografia, o EIM sobe bastante.

Logo, não há a melhor solução para todos os cenários. Porém, com os resultados desta pesquisa, acredita-se que a grande maioria das necessidades de auditoria de serviços poderia ser fornecida por uma companhia que disponibilizasse uma Arquitetura Decorador Externa largamente distribuída. O fornecimento de um serviço em escala largamente distribuída (como os exemplos do Akamai [33] e Google [34]) da Arquitetura Decorador Externo permitiria ao Auditor implantar o Inspetor próximo ao Consumidor ou ao Provedor, minimizando dessa forma o impacto de desempenho causado pela auditoria. Uma vez que a

única desvantagem da Arquitetura Decorador Externo é a perda de desempenho, essa abordagem poderia ser uma solução bastante competitiva.

Os trabalhos futuros sugeridos são: (i) analisar a escalabilidade das arquiteturas apresentadas, no sentido de reduzir o trabalho e o tráfego de mensagens para garantir a auditoria de um maior número de SLAs entre serviços; (ii) analisar técnicas para evitar que os códigos dos Inspetores na Arquitetura Decoradores nos Hosts e os valores dos SLIs obtidos sejam modificados pelos seus hospedeiros [26]; (iii) investigar o uso da auditoria de SLAs para atribuir valores de credibilidade aos Provedores e Consumidores de serviço como técnica de auxílio na escolha dos parceiros de negócios; e (iv) analisar as outras fases da gerência de SLAs para serviços (negociação e estabelecimento de SLAs, notificação de violações e tomada de ações de gerência) nas arquiteturas propostas nesta dissertação.

Referências Bibliográficas

[1] Catania N., Murray B., et al. **Web Services Management Framework – Overview (WSMF - Overview) Version 2.0**. Public draft release, 16 de Julho de 2003.
http://devresource.hp.com/drc/specifications/wsmf/WSMF-Overview.jsp

[2] 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", Volume 11, Número 1, Plenum Publishing Corporation, Março de 2003.

[3] Rashid J. Al-Ali, Omer F. Rana and David W. Walker. **G-QoSM: Grid Service Discovery Using QoS Properties**. Computing and Informatics Journal, Special Issue on Grid Computing, 21(5), 2002.

[4] Foster I., Kesselman C., Lee C., Lindell R., Nahrstedt K., Roy A. **A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation**. International Workshop on Quality of Service, 1999.
http://www-fp.globus.org/documentation/papers.html

[5] Barbosa A. C., Sauvé J., Cirne W., Carelli M. **Independently Auditing Service Level Agreements in the Grid**. Proceedings of the 11th HP OpenView University Association Workshop (HPOVUA 2004). Junho de 2004.

[6] Tidwell D., **Tutorial: Web services: the Web's next revolution**. Dezembro de 2004.
http://www-105.ibm.com/developerworks/education.nsf/webservices-onlinecourse-bytitle/BA84142372686CFB862569A400601C18?OpenDocument.

[7] Sotomayor B. **The Globus Toolkit 4 Programmer's Tutorial**. Fevereiro de 2005.
http://gdp.globus.org/gt4-tutorial/

[8] **Web Service Concepts – a Technical Overview**.
http://www.hpmiddleware.com/downloads/pdf/web_services_tech_overview.pdf

[9] Foster I., Kesselman C. and Tuecke S., **The Anatomy of the grid: Enabling Scalable Virtual Organizations**. International Journal of High Performance Computing Applications, 2001.
http://www.globus.org/research/papers/anatomy.pdf

[10] De Roure D., Mark A. Baker, et al., **The Evolution of the Grid**. Novembro de 2004.
http://www.semanticgrid.org/documents/evolution/evolution.pdf

[11] Foster I., Kesselman C., Nick J., Tuecke S. **The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration**. Open Grid Service Infrastructure WG, Global Grid Forum, 22 de Junho de 2002.
http://www.globus.org/research/ogsa.pdf

[12] Sotomayor B. **The Globus Toolkit 3 Programmer's Tutorial**. Janeiro de 2005.
http://gdp.globus.org/gt3-tutorial/

[13] Czajkowski K., Ferguson D., Foster I., et al. **From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution.** Versão 1.1, 05 de Março de 2004.
http://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf

[14] Sturm R., Wayne M., Jander M. **Foundations of Service Level Management**. Sams, 2000.

[15] Lewis L. **Service Level Management for Enterprise Networks**. Artech house, 1999.

[16] Argonne National Laboratory. **The Globus Project**. Janeiro de 2005.
http://www.globus.org

[17] Murray B., Srinivasan L., TreadWell J. **Managing the Grid with WSMF**. Globus World 2004, 20 de Janeiro de 2004.
www.globusworld.org/program/slides/2c.pdf

[18] Página do **Projeto WSLA**. Fevereiro de 2005.
http://www.research.ibm.com/wsla/about.html

[19] Dan A., Davis D., et al., **Web Services on demand: WSLA-driven Automated Management**, IBM Systems Journal, Special Issue on Utility Computing, Volume 43, Número 1, páginas 136-158, IBM Corporation, Março de 2004.
http://www.research.ibm.com/journal/sj/431/dan.pdf

[20] IBM: Alpha Works Web Site. **Emerging Technologies Toolkit**. Fevereiro de 2005.
http://www.research.ibm.com/wsla/ettk.html

[21] ShaikhAli A., Rana O., et al. **UDDIe: An extended registry for web services**. Workshop on Service Oriented Computing: Models, Architectures and Applications. Orlando, Estados Unidos, 2003.

[22] Czajkowski K., Foster I., et al. **A resource management architecture for metacomputing systems**. The 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[23] Sahai A., Durante A., Machiraju V., **Towards Automated SLA Management for Web Services**. Research Report HPL-2001-310 (R.1), Hewlett-Packard (HP) Labs Palo Alto. 26 de Julho de 2002.
http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf

[24] Syverson F., Goldschlag M., Reed G., **Anonymous Connections and Onion Routing.** Proc. IEEE Symp. on Security and Privacy, Oakland, Maio de 1997.

[25] Gamma E., et al. **Design Patterns: Elements of Reusable Object-Oriented Softwar***e*. Addison-Wesley, 1995; ISBN: 0201633612.

[26] Sander T., Tschudin C., **Protecting Mobile Agents Against Malicious Hosts**. *G*. Vigna ed., Mobile Agent Security, LNCS, Fevereiro de1998.

[27] Smith S., Weingart S. **Building a High-Performance, Programmable Secure Coprocesor.** IBM Research Report RC 21102, Fevereiro de 1998.

[28] Yee B. **Using Secure Coprocessors.** Tese de doutorado, Carnegie Mellon University, 1994.

[29] Devore J. **Probability and Statistics for Engineering and the Sciences.** Quarta Edição, Wadsworth Publishing Company, 1995.

[30] Jain R., **The Art of Computer Systems Performance Analysis – Techniques for Experimental Design, Measurement, Simulation and Modeling**. John Wiley & Sons, Inc., Littleton Massachusetts, 1992.

[31] Soares F., Farias A., Cesar C., **Introdução à Estatística**. Livros Técnicos e Científicos Editora S.A., Rio de Janeiro, 1991.

[32] Kleinrock L. **Queueing Systems, Vol I: Theory.** Wiley, New York, 1975.

[33] Site do **Akamai**. 2005.
http://www.akamai.com

[34] Brin S., Page L. **The anatomy of a large-scale hypertextual Web search engine.** Ashman and Thistlewaite [2], páginas 107-117. Brisbane, Austrália. 1998. http://citeseer.ist.psu.edu/brin98anatomy.html

[35] Site do **OurGrid**. 2005.
http://www.ourgrid.org

Apêndice A

Evaluating Architectures for Independently Auditing Service Level Agreements

Ana Carolina Barbosa, Jacques Sauvé, Walfredo Cirne
Universidade Federal de Campina Grande (UFCG)

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 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 part 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 parts do not blindly trust each other, which is expected to be the common case for large grid deployments. In fact, no single architecture seems to be the best solution for all cases. We here present six architectures that perform SLA auditing, as well as a quantitative and qualitative evaluation of such architectures. The quantitative evaluation focus on the performance penalty that auditing introduces. The qualitative evaluation compare the architectures based on aspects such as intrusiveness, trust, use of extra requests, possibility of preferential treatment, possibility of auditing of Consumer load, and possibility of auditing of encrypted messages.[1]

A.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 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

[1] Submetido para publicação no Future Generation Computer Systems, o Jornal Internacional de Computação Grid: Teoria, Métodos e Aplicação, em agosto de 2005.

Service Level Agreements (SLAs) are used to establish what a Consumer application can expect from and demand to a Provider. An SLA is composed of a set of Service Level Objectives (SLOs), which are evaluated using measurable data, called Service Level Indicators (SLI).

Since SLAs are simply contracts, they do not provide hard guarantees per se. One must audit them to ensure that they hold. However, as 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 could 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] [2] [3] [4], they do not analyze the SLA auditing phase in detail.

This paper fills this gap, presenting and evaluating six architectures for independently auditing Web and Grid services. We address issues concerning trust between services; where and how 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. Our efforts are part of the OurGrid project developed in collaboration between UFCG and HP (Hewlett-Packard) [34].

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 shows a quantitative performance analysis of these architectures; and, finally, Section 6 concludes the paper.

A.2. Related Work

Service Level Agreements have been applied for managing Web Services and, recently, for managing resources and Grid Services. This section presents some work being developed in this area.

The WSMF framework [1] is an HP proposal for managing resources through Web Services. At the moment of this work, the efforts to adapt WSMF to the Grid world are still work in progress. The managed objects are Web Services representing resources or services. These Web Services implement management interfaces and, thus, they are responsible for

providing management information. In this framework, although there are not constraints in using SLAs, Web Service management does not consider them and, thus, there is not concern with trust in the management information provided by the managed objects. Contrary to the WSMF, our work focuses on the SLA auditing phase.

WSLA framework [2] is an IBM effort for specifying and monitoring of SLAs for Web Services. It consists of an SLA definition language (WSLA) based on an XML scheme and of a runtime architecture, which 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 the 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 presents 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.

Another related work is G-QoSM framework [3], which addresses SLA management for Grid Services, including the process of service discovery based on QoS requirements, SLA negotiation, monitoring and execution of management actions in case of SLA non-compliance. 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. However data are obtained from local monitoring tools, with no consideration 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 GARA (General-purpose Architecture for Reservation and Allocation) [4]. 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.

A.3. Issues Concerning in SLA Auditing

SLAs need to be audited to give real Quality of Service (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 services have a strong trust relationship, they can believe in each other's SLIs (such as Consumer's rate of requests and Provider's response time). Unfortunately, in a highly dynamic service-binding scenario (such as a grid) 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. 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, the response time was too high, the Provider can dispute this claim by presenting its own (smaller) response times [5].

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 [5].

However, this proposal begs the question: How is the Auditor going to obtain trustworthy SLIs? Asking Consumer and Provider for SLIs poses 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 reality. Furthermore, probing only helps to get Provider-related SLIs; gauging consumer-related SLIs (such as the submitted load) remains an issue [5].

Another issue in auditing SLAs relates to the change of 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 named 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 [5].

A.4. Architectures for Independently Auditing Services

We started this research aiming to determine the best way for an independent third-party SLA auditor work. We found, however, that there is no simple solution for this problem. Different architectures have different pros and cons, and what solution is the best one depends what criteria the user finds more important. We here present and qualitatively evaluate six architectures for SLA auditing. In the next Section, we quantitatively evaluate their performance.

A.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 Figure 1). In this solution, the Auditor asks for SLIs from the parties and compares them with the values agreed to in the SLA to verify contract compliance. SLA evaluation can be automated for SLAs defined through a common language [23]. 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.

Naturally, this solution requires the Auditor to trust both the Provider and the Consumer and thus it is not appropriate for the general case we investigate here. We present this architecture as a base solution; against with we should compare more sophisticated architectures. Observe also that this solution require the services themselves to provide the SLIs to the Auditor, requiring changes in the original Consumer and Provider code.

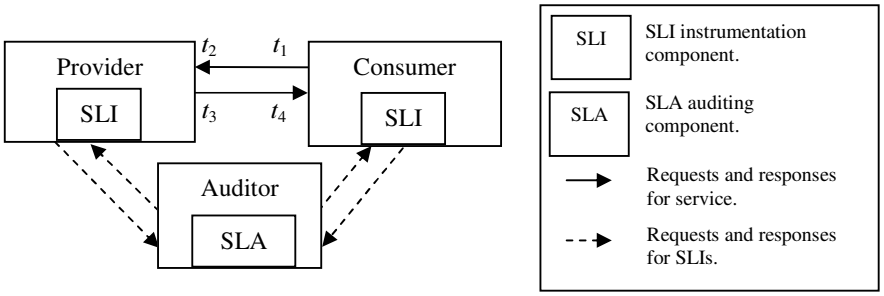

Figure 1: Auditor evaluates the SLA from SLIs provided by Consumer and Provider.

A.4.2 Packet Sniffing Architecture

A second architecture uses trusted sniffers in the Provider's and Consumer's LANs in order to passively capture packets addressed to them by the 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, which are 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 sniffed information. Therefore the Auditor requests SLIs from the Inspector (see Figure 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.

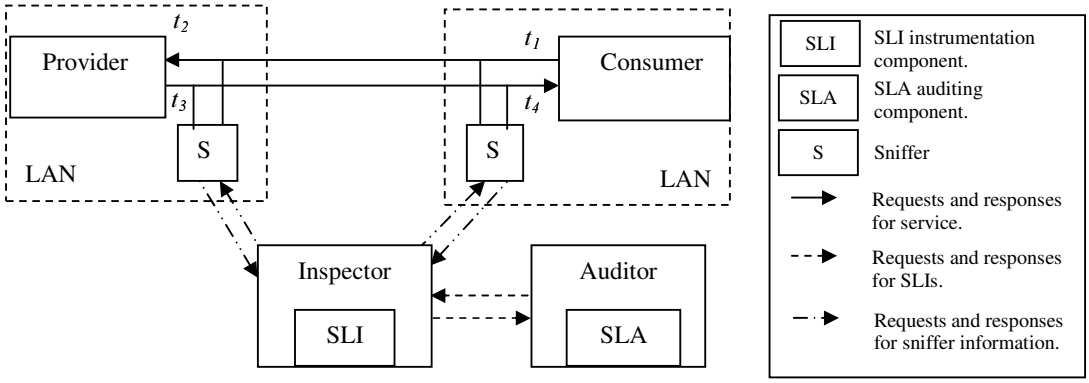

Figure 2: Auditor evaluates the SLA from SLIs provided by Inspector, which calculates them from sniffer information. Sniffers are represented in the picture by S.

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, what requires hardware support [26]. This requires building tamper-resistant sniffers or installing of a secure coprocessor on regular sniffers. A secure coprocessor [27] [28] 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 of capture enough information needed to calculate SLIs if the packets are encrypted, which is a common

situation in the inter-domain 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 may see all needed information to measure SLIs. Another drawback is that the Consumer and the Provider must blindly trust the Inspector. This is because the packet sniffers may capture any traffic in the LAN, therefore accessing 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.

A.4.3 Host Decorators Architecture

A third architecture has Inspectors residing in the Provider and in Consumer hosts (see Figure 3). The Inspectors implement the Provider interface. In fact, Inspectors act as decorators to the Provider. (A *decorator* is a well-known design pattern that is used for adding additional functionality to a particular object as opposed to a class of objects [25].) The Consumer requests service to the Inspector in its own host, which forwards requests to the Inspector in the Provider host, which then forwards requests 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 hard, 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, in practice, SLAs do not care about application data; they typically use SLIs as response time and availability. Therefore, this solution should allow for safe transfer of sensitive data and auditing for most 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 [27] [28].

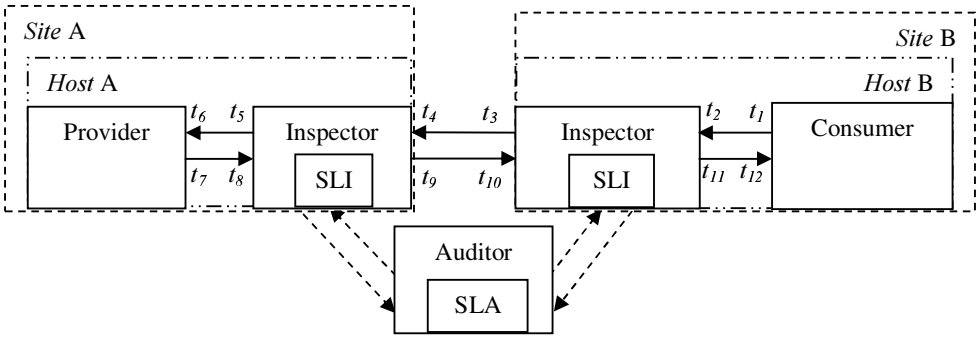

Figure 3: Auditor evaluates the SLA from SLIs provided by Inspectors, which are located in Provider and Consumer hosts.

A.4.4 Independent Inspector Architecture

A fourth architecture deals with the trust issue without burdening the services or its hosts with instrumentation code. It delegates the instrumentation function to a third party Inspector, located in a different host (see Figure 4A). This service calculates service SLIs send extra requests to the Provider and hands them to the Auditor, which performs the SLA evaluation. Since an Inspector is a Provider's Consumer, it needs to know the Provider interface. Thus, each Provider must have its own Inspector, which knows how 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's database) and impose extra load, reducing the Provider's 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 those described in [24], to conceal the network address of both Consumer and Inspector. But extra care should be taken to avoid that service-level information gives 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 Figure 4B). 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 a per-need 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 Inspector works as a Consumer, it also obtains tickets from the Auditor.

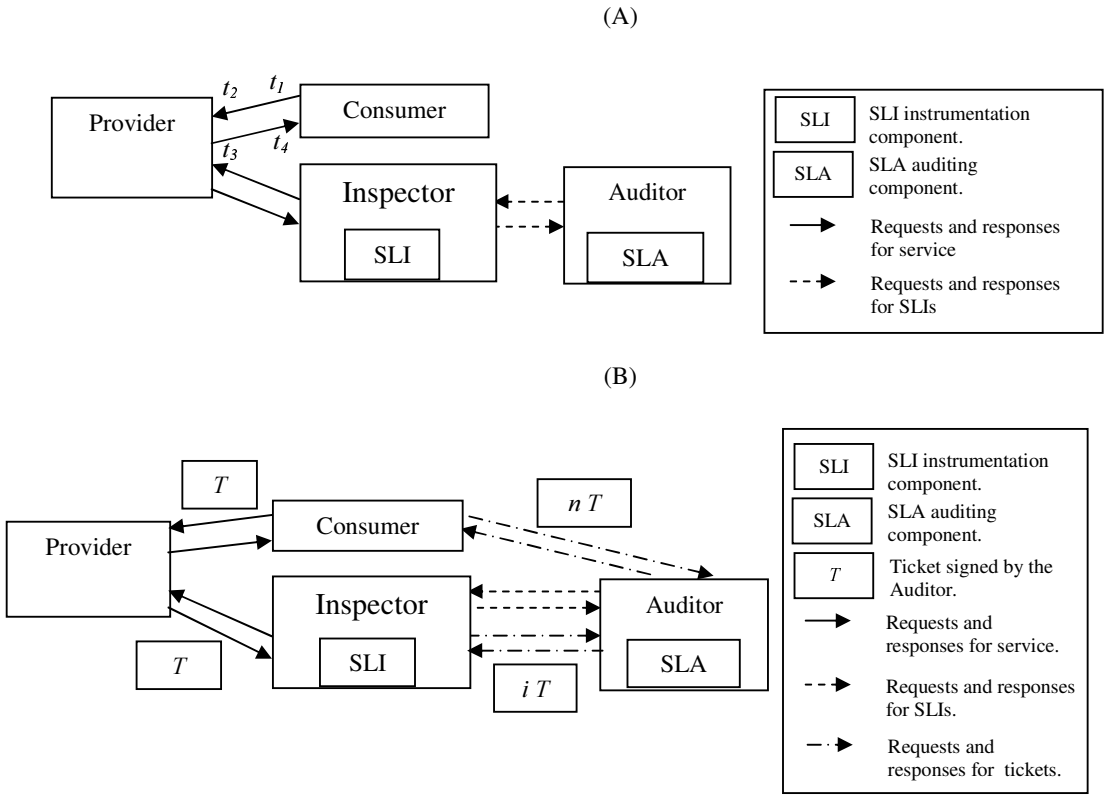

Figure 4: (A) Auditor evaluates the SLA from SLIs provided by Inspector and calculated through extra requests (B) In order to audit Consumer-related SLIs, the Auditor can provider the Consumer with signed tickets that are asked by the Provider to deliver service.

A.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 not outside both Consumer and Provider sites (see Figure 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 Provider's method calls. The Inspector forwards the Consumer's requests, measures the load submitted by the Consumer to the Provider (or anything else of interest in the Consumer data) to the Provider and calculates SLIs obtained from the Provider's side.

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 the Inspector's requests and give them preferential treatments 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 Consumers' 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. The performance loss can be reduced if many distributed sites are used for Inspectors and Consumers are allowed to use the closest Inspector. 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 in a distant site since requests that could use a shorter route will need to access the Inspector for the purpose of SLA auditing.

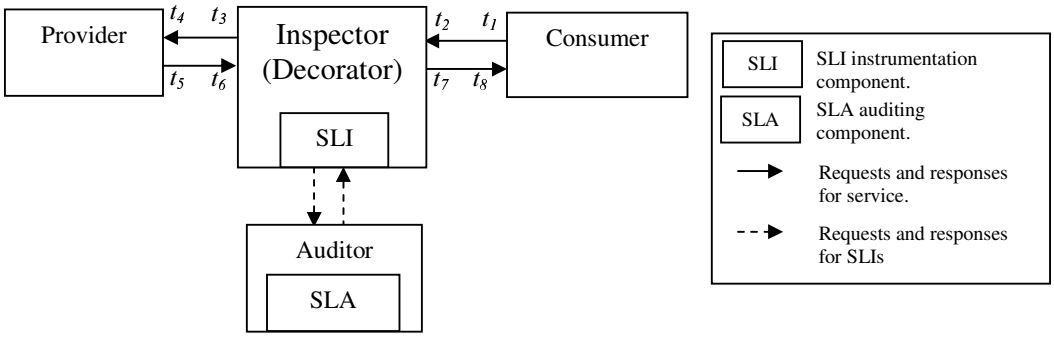

Figure 5: Auditor evaluates the SLA from SLIs provided by Inspector working as a decorator to the Provider and calculated without extra requests.

A.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 Figure 6). This alternative tries to reduce the performance impact caused when all requests are forwarded to the Inspector. While this solution 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.

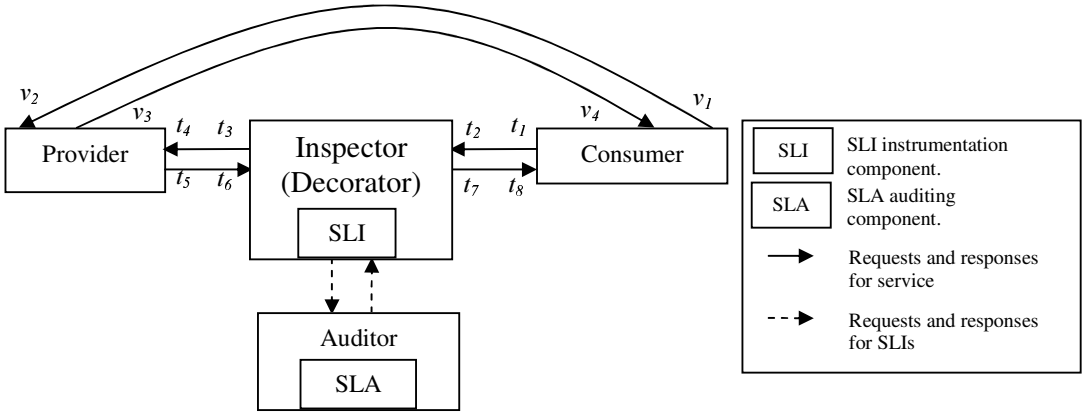

Figure 6: Auditor evaluates the SLA from SLIs provided by Inspector working as a decorator to the Provider. Some requests go to Provider directly to reduce the performance impact.

A.4.7 Summary of the Qualitative Evaluation

Table 1 summarizes the analysis of the advantages and drawbacks of each architecture, providing an overall comparison to the readers. As discussed above, the features explored in the evaluation are:

Trust: whether or not trust is necessary between the parties signing the SLA and, if so, in which processes trust is necessary.

Intrusiveness: the inclusion of instrumentation in the code or 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 realize auditing; this increases load and may incur 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 exposing them to evaluation.

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

A.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, the calculated response time SLI can diverge from the value of the response time obtained had auditing not been performed. 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 statistics techniques to estimate the smallest sample size that assures the desired confidence interval in the measurements [29]. But nevertheless whatever additional requests are used will affect performance.

These factors two 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 a 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 A.5.2.

In the **Packet Sniffing Architecture**, which uses sniffers to passively capture packets, the sample size factor is not present because all requests come from a real Consumer and there is no MIE, since the performance of the service is not affected by auditing. Although the 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 Decorators Architecture**, which has an Inspector in the Consumer and the Provider hosts, the sample size factor is not present because there are no additional requests. The MIE factor 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 a MIE, e_4. This value is larger than e_1 since the load imposed by business requests is typically much larger that 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, we do not need to analyze the sample size. 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 can 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 analyze 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 assure a given confidence interval [29]) and N is the population size (the total number of requests performed over the observation period).

A.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 Figure 7).

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, $\delta_p = u_3 - u_2$. The interval δ_{C-P} represents the total network delay between Consumer and Provider. Since time is measured at the Web Service application level, the time interval δ_p is based on timestamps obtained in this level. The time spent in the protocol levels under the application level is thus contained in the network delay.

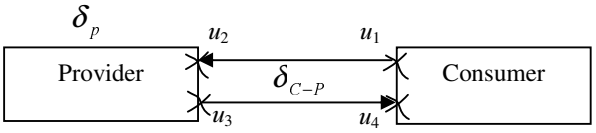

Figure 7: The Base Architecture: Consumer-Provider communication without auditing.

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 Figure 1). The time intervals are $\delta_p, \delta_{C-P}, \delta_{SLI}$, 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:

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

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 Architecture. So:

$$e_1 = \delta_{SLI}$$

In the **Packet Sniffing Architecture**, the timestamps are t_1, t_2, t_3, t_4 (see Figure 2) and the time intervals are δ_p, δ_{C-P}. The time at which the Consumer receives the answer to its request is:

$t_4 = t_1 + \delta_p + \delta$ _

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 Decorators Architecture**, $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}$ are the timestamps (see Figure 3) and the time intervals are $\delta_p, \delta_{C-I}, \delta_{I-I}, \delta_{I-P}, \delta_f, \delta_r, \delta_{cript}$. The interval δ_{cript} is the delay due to the time spent to encrypt functions and results of SLIs calculations 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:

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

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

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

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

Since an Inspector is located in the Consumer host and the other is in 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:

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

The MIE e_3 is the difference between the time spent by the request in this architecture 3 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:

$e_3 = 2\delta_{WS} + 2\delta_f + 2\delta_{cript} + 2\delta_r$

In the **Independent Inspector Architecture**, the timestamps are t_1, t_2, t_3, t_4 (see Figure 4). The time intervals are $\delta_p, \delta_{C-P}, \delta_m$, where δ_m is the marginal delay, which is the delay 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 , we use a queuing model (see Figure 8). The requests of Consumers and Inspector 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.

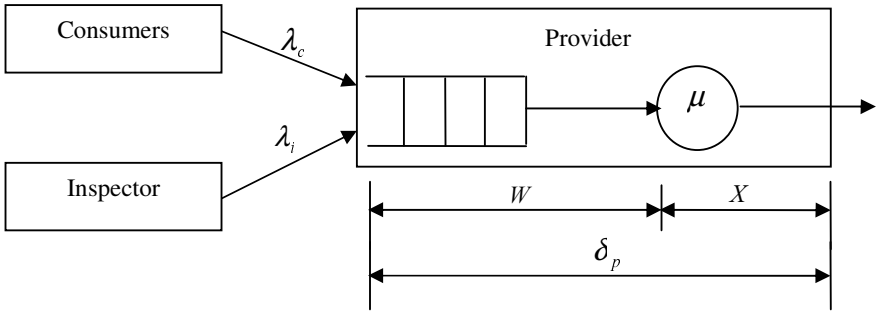

Figure 8: Queuing Model

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 considered a system M/G/1, in which the inter-arrival times are exponentially distributed, the service times have a general distribution and there is a single server [32], 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:

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

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

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

Here ρ_{ci} is the server utilization and varies 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 $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8$ (see Figure 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 Consumer and Inspector and δ_{I-P} is the total network delay between Inspector and Provider. The interval δ_f is the time spent by Inspector to forward the request to Provider and is $\delta_f = t_3 - t_2$. The time interval δ_r is the time spent by Inspector to return the answer to 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 5 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 Figure 6, in the **External Decorator with Bypass Architecture**, the timestamps are $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8$ (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 total requests N, that is, $N-n$ requests). As we have already shown the measurement interference error for this architecture is:

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

And hence:

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

A.5.2. Experimental Evaluation

This section intends to make the analytical results more concrete by gauging the performance penalty of auditing architectures in a real environment. We implemented a bookstore service that can include and exclude books from the purchase list. Services were implemented under Globus 3.2.1 [16] deployed in a Pentium 4, 2.66 GHz of CPU, 512 MB of RAM and 40 GB of HD. The Globus 3.2.1 was chosen as programming environment because it offers APIs to implement both Web Services and Grid Services. Experiments were performed to obtain mean values for δ_{SLI}, δ_f, δ_r, δ_{WS}, δ_{crypt}, used in the equations presented in Section 5.1 to describe the MIE of each architecture (See Table 2). The values obtained for each parameter are mean values calculated from samples collected in groups of 20 ($n = 20$), confidence level of 95%, acceptable maximum error of 5% ($\varepsilon_{\max} = 5\%$). The first four samples were eliminated because the response times of initial requests are greater due to the communication establishment between the service Provider and the Consumer (initial transient period).

Table 2: Values for the parameters obtained by experiments

Measured Parameter	Value
δ_{SLI}	$0.02354ms$
δ_f	$0ms$
δ_r	$0.5ms$
δ_{WS}	$10ms$
δ_{crypt}	$130.3ms$

While one can expect that δ_{SLI}, δ , δ_r, δ_S , δ_{crypt} will remain the same (for the current computer technology), service time and network speed will vary depending on the service itself and where the Provider, the Consumer and the Inspector are placed in the

network, respectively. And service time affects the MIE of the Independent Inspector Architecture, where as 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 affect by the SLI variance (the greater the variance, the greater the sampling needed to achieve a given confidence interval).

In order to give the reader a broader view of the possibilities, we calculated the MIE valuing each of these parameters (service time, network speed, and sample size) in "low" and "high". More precisely, we assume a low service time $\overline{X}$ of 10*ms* and a high service time $\overline{X}$ of 1000*ms*; low network delay of a few microseconds (typical of a LAN) and a high network delay of 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 a (legitimate) arrival rate of half of the service time ($\lambda = \frac{\mu}{2}$). Table 3 summarizes the results.

Table 3: Analyzing the Measurement Interference Error of the architectures.

Architectures		**Small Sample (5%)**	**Large Sample (30%)**
1. Naïve		0.02354*ms*	0.02354*ms*
2. Packet Sniffing		0*ms*	0*ms*
3. Host Decorators		281.6*ms*	281.6*ms*
4. Independent Inspector	**Fast Provider (10*ms*)**	0.5263*ms*	4.285*ms*
	Slow Provider (1000*ms*)	52.63*ms*	428.5*ms*
5. External Decorator	**LAN (0*ms*)**	10.5*ms*	10.5*ms*
	WAN (200*ms*)	210.5*ms*	210.5*ms*
6. Ext Decorator with Bypass	**LAN (0*ms*)**	0.525*ms*	3.15*ms*
	WAN (200*ms*)	10.53*ms*	63.15*ms*

From Table 3, one can see that the MIE for the Naive and Packet Sniffing Architectures is negligible. The Host Decorators Architecture presents somewhat large MIE, which is due to the need of two rounds of cryptography in both Inspectors it uses. The Independent Inspector Architecture incurs in the highest overhead of the study (428.5*ms*) when it demands a large sample from a slow Provider. However, if the sampling is small and (especially) if the Provider is fast, such architecture performs well. The MIE of 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.

A.6. Conclusions

This paper addresses the auditing of service level agreement for Web and Grid Services. In particular, we focus on the case that Provider and Consumer do not blindly trust each other, which is envisioned to be the common case for large grid deployments. We presented six possible architectures for the auditing process, as well as their advantages, drawbacks and performance analysis.

The Naive Architecture was provided as a bottom line, as it does assume trust between Provider and Consumer. Moreover, it requires changes in the code of Provider and Consumer. Such intrusive changes on the code of Provider and Consumer can be avoided by using the Host Decorators Architecture, at the expense of adding an approximate 300*ms* 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 Decorators Architectures require the installation of special hardware in both Provider and Consumer, what 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 added load to the Provider, side-effects on the Provider database, and possible preferential treatment 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 then network) Consumer, Provider and Inspectors are. The External Decorator with Bypass Architecture attempts to address this potential performance problem, but reintroduces (in 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 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 External Decorator Architecture single issue is performance, this approach would likely create a very competitive solution.