**Universidade de Brasília**
**Instituto de Ciências Exatas**
**Departamento de Ciência da Computação**

# Mecanismo de Negociação de Auditor de QoS para Grades baseado em WS-Agreement

Alisson Wilker Andrade Silva

Brasília
2011

**Universidade de Brasília**
**Instituto de Ciências Exatas**
**Departamento de Ciência da Computação**

# Mecanismo de Negociação de Auditor de QoS para Grades baseado em WS-Agreement

Alisson Wilker Andrade Silva

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Alba Cristina Magalhães de Melo

Brasília
2011

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Mauricio Ayala Rincón

Banca examinadora composta por:

Prof.ª Dr.ª Alba Cristina Magalhães de Melo (Orientadora) — CIC/UnB
Prof. Dr.ª Célia Ghedini Ralha — CIC/UnB
Prof. Dr.ª Liria Matsumoto Sato — Poli/USP

**CIP — Catalogação Internacional na Publicação**

Ficha catalográfica elaborada pela Biblioteca Central da Universidade de Brasília. Acervo 990567.

Silva, Alisson Wilker Andrade.

S586mn Mecanismo de Negociação de Auditor de QoS para Grades baseado em WS-Agreement / Alisson Wilker Andrade Silva. –
2011.
xi, 73 f. : il. ; 30 cm.

Dissertação (mestrado) - Universidade de Brasília, Instituto de Ciências Exatas, Departamento de Ciência da Computação, 2011.
Inclui bibliografia.
Orientação: Alba Cristina Magalhães de Melo.

1. Redes e Sistemas Distribuídos. 2. Processamento eletrônico de dados - Processamento distribuído. 3. Controle de qualidade. I. Melo, Alba Cristina Magalhães de. II. Título.

CDU 004.75

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

**Universidade de Brasília**
**Instituto de Ciências Exatas**
**Departamento de Ciência da Computação**

# Mecanismo de Negociação de Auditor de QoS para Grades baseado em WS-Agreement

Alisson Wilker Andrade Silva

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Alba Cristina Magalhães de Melo (Orientadora)
CIC/UnB

Prof. Dr.ª Célia Ghedini Ralha
CIC/UnB

Prof. Dr.ª Liria Matsumoto Sato
Poli/USP

Prof. Dr. Mauricio Ayala Rincón
Coordenador do Mestrado em Informática

Brasília, 21 de julho de 2011

# Dedicatória

Dedico esse trabalho a todos os professores que acreditam numa melhor educação nesse país e que, muitas vezes, enfrentam grandes dificuldades para compartilhar conhecimento e formar os cidadãos de amanhã. Dedico também a cada aluno que luta diariamente para ter acesso a um direito seu, que é uma educação de qualidade, em busca de garantir um espaço nesse Brasil emergente.

# Agradecimentos

Agradeço a Deus pela oportunidade de viver momentos preciosos como este. Agradeço a minha esposa Hellen, pela paciência com que me aturou durante as noites a fio em frente ao computador. Agradeço aos meus pais e aos meus irmãos pelo apoio incondicional em todos os momentos da minha vida. Agradeço a minha orientadora, Doutora Alba Melo, pela orientação sempre presente nos momentos de dúvida. Agradeço a Viviane Malheiros e a José Maria Leocádio, do Serviço Federal de Processamento de Dados, pela colaboração para a conclusão deste trabalho. Agradeço aos meus amigos e parentes, pela compreensão de minha ausência nos momentos de comunhão. Agradeço a todos os professores que participaram da minha jornada de aprendizagem durante os últimos vinte e poucos anos e que, certamente, contribuíram para esse momento de realização.

# Resumo

As plataformas de alto desempenho compostas por recursos computacionais comuns, como as grades computacionais e os sistemas *peer-to-peer*, evoluíram bastante e assumiram um papel de destaque na última década. Porém, o seu uso de forma mais ampla ainda depende do estabelecimento de uma infraestrutura de qualidade de serviço (QoS) efetiva nesses ambientes, possibilitando a utilização comercial dessas plataformas em escala global. Por esse motivo, várias propostas de mecanismos para prover QoS em grades computacionais têm surgido recentemente, nas quais consumidor e provedor monitoram e controlam os recursos da grade a fim de garantir acordos de nível de serviço previamente estabelecidos. Contudo, devido à falta de confiança entre provedor e consumidor com relação ao monitoramento desses acordos, surge a necessidade de se introduzir a figura de um auditor de QoS como uma terceira entidade, que deve ser imparcial e de confiança tanto do provedor como do consumidor, a fim de resolver conflitos de interesse. Como podem existir vários auditores de confiança do provedor e do consumidor, o auditor de QoS também precisa ser negociado e estabelecido, assim como é feito para o estabelecimento do acordo de nível de serviço firmado entre as partes. A presente dissertação de mestrado propõe e analisa um mecanismo de negociação de auditores de QoS nesse contexto. Algumas das características do mecanismo proposto são a baixa intrusividade e o uso de padrões abertos, como o WS-Agreement.

**Palavras-chave:** Computação em Grade, Qualidade de Serviço, Negociação de Auditor

# Abstract

High performance platforms composed of ordinary computing resources, such as grids and peer-to-peer systems, have greatly evolved and assumed an important role in the last decade. Nevertheless, its wide use still depends on establishing an effective quality of service (QoS) infrastructure in those environments, which may enable the commercial use of those platforms in global scale. For this reason, a variety of proposals to support QoS in grids have emerged recently, in which consumer and provider monitor and control grid resources in order to guarantee previously established service level agreements. However, due to the lack of trust between provider and consumer in relation to monitoring those agreements, it becomes necessary to introduce a QoS auditor as a third entity, which must be impartial and trusted by both provider and consumer, in order to solve conflicts of interest. As there may be several auditors trusted by provider and consumer, the QoS auditor also needs to be negotiated and established, just as for the establishment of the service level agreement agreed by the parties. The present master's thesis proposes and analyzes a negotiation mechanism for QoS auditors in this context. Some of the proposed mechanism's characteristics are low intrusiveness and use of open standards, such as WS-Agreement.

**Keywords:** Grid Computing, Quality of Service, Auditor Negotiation

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

A busca por acesso a um alto poder computacional de forma universal e simples tem sido o objetivo de muitos cientistas da computação desde a década de 1960. Já na década de 1970, reconheceu-se que tal poder computacional podia ser obtido a um baixo custo através do agrupamento coordenado de estações de trabalho, ao invés de um único e caro supercomputador (Organick, 1972). Desde essa época, os cientistas perceberam as dificuldades inerentes ao desenvolvimento de sistemas distribuídos, tais como: mensagens perdidas, corrompidas e atrasadas, sincronização de tarefas, coerência de memória, tolerância a falhas, entre outras. Essas dificuldades deram início a uma série de pesquisas importantes na área de computação distribuída, a fim de construir algoritmos robustos que levassem a sistemas distribuídos coerentes, controláveis e de alto desempenho (Chandy e Lamport, 1985) (Lamport, 1978).

Desde então, várias propostas de arquitetura de sistemas distribuídos surgiram como alternativas para prover o alto poder de processamento e armazenamento necessários a diversos tipos de aplicação atuais, como previsão do tempo, renderização de imagens 3D, genômica comparativa, entre outros. Entre essas alternativas, podemos citar os *clusters*, os sistemas *peer-to-peer* (P2P) e as grades computacionais (Foster e Kesselman, 1998) (Schollmeier, 2001). Diferentemente dos clusters, que são plataformas de processamento de alto desempenho compostas por computadores relativamente similares, localizados em um mesmo espaço geográfico e conectados por redes locais de alta performance, as grades computacionais e os sistemas P2P são normalmente compostos por computadores de configurações diversas, distribuídos geograficamente ao redor do mundo e acessíveis através de vários tipos de protocolos de interconexão.

Os sistemas P2P e as grades computacionais geralmente utilizam os ciclos ociosos de máquinas disponíveis na rede para formar a plataforma de processamento de alto desempenho. Assim, em sistemas P2P e grades computacionais, os recursos normalmente não são dedicados e o trabalho de processamento é realizado com base no princípio do melhor esforço (*best-effort*). De acordo com esse princípio, os serviços requisitados nesses ambientes são realizados conforme a disponibilidade momentânea de recursos computacionais compatíveis com a demanda. Ou seja, não há garantias de realização do serviço requerido, uma vez que pode não haver recursos compatíveis disponíveis na plataforma.

Esse fato é agravado pela característica dinâmica e heterogênea dos recursos computacionais disponíveis nessas plataformas. Uma vez que há pouco acoplamento entre os recursos computacionais e pouco ou nenhum controle sobre sua disponibilidade, normal-

mente as entradas e saídas de recursos nessas plataformas ocorrem com muita frequência. Além disso, como há poucas restrições para a participação de um recurso computacional na rede P2P ou na grade computacional, os recursos disponibilizados são geralmente muito heterogêneos, dificultando ainda mais a previsibilidade de realização dos serviços demandados.

Os sistemas P2P e as grades computacionais, porém, têm evoluído bastante e já se apresentam como plataformas confiáveis e robustas para provimento de computação de alto desempenho. Com isso, diversas organizações ao redor do mundo têm se interessado cada vez mais pelo uso comercial dessas plataformas, visando alugar recursos computacionais conectados através de redes P2P e grades computacionais para o provimento de alto poder de processamento. Contudo, o princípio do melhor esforço é evidentemente inadequado para o uso comercial dessas plataformas, uma vez que o usuário que pagar pelo aluguel de recursos de uma grade computacional baseada nesse princípio, por exemplo, não terá qualquer garantia sobre quando, onde e como a sua demanda será executada.

Impulsionadas por esse problema, surgiram diversas pesquisas relacionadas a como prover qualidade de serviço em um ambiente tão heterogêneo e dinâmico quanto as grades e os sistemas P2P (Burchard et al., 2005) (Menascé e Casalicchio, 2004). A maioria dessas pesquisas se utiliza de acordos de nível de serviço (Lamanna et al., 2003) para definir a qualidade de serviço acordada previamente entre consumidor e provedor de serviços. Esses acordos definem, entre outras questões, o que deve ser executado, quem deve executar, com que nível de serviço e as multas por violação da qualidade de serviço acordada.

Para que esses acordos sejam cumpridos, é necessário que os serviços objetos desses acordos sejam monitorados quanto ao seu nível de serviço. Esse monitoramento pode ser realizado pelas próprias partes envolvidas (consumidor ou provedor de serviços) ou por alguma entidade externa de confiança dessas partes. Porém, quando existem problemas de confiança entre consumidor e provedor de serviços, a monitoração dos níveis de serviço por uma entidade externa se torna essencial. Essa entidade externa deve ser de confiança de ambas as partes e funciona como um auditor da qualidade de serviço (Barbosa et al., 2006), verificando o cumprimento do acordo de nível de serviço tanto pelo provedor como pelo consumidor dos serviços.

Existem diversos trabalhos na literatura (Racz e Stiller, 2009) (Hasan e Stiller, 2005) (Tserpes et al., 2008) que abordam o tema de um auditor externo de qualidade de serviço. Em (Hasan e Stiller, 2005), por exemplo, é proposto o uso de um auditor externo para avaliar o serviço de um operador de rede com suporte a qualidade de serviço. Em todos os trabalhos encontrados na literatura, porém, são discutidas apenas as características do auditor externo, considerando que este já está presente e operante no sistema. A nosso conhecimento, portanto, não existem trabalhos na literatura que tratem da negociação do auditor externo de qualidade de serviço.

Uma vez que podem existir diversos auditores de confiança do consumidor ou do provedor com características e capacidades de monitoração diferentes, acreditamos que o auditor externo de qualidade de serviço deve ser negociado previamente entre as partes, assim como é feito para o estabelecimento do acordo de nível de serviço. Essa negociação deve ocorrer logo após o estabelecimento do acordo, visto que o auditor precisa ser capaz de monitorar os termos acordados entre provedor e consumidor.

A presente dissertação de mestrado propõe um mecanismo para negociar um auditor externo de qualidade de serviço em conformidade com premissas e restrições inerentes às

plataformas de grades computacionais que usam acordos de nível de serviço.

As principais contribuições dessa dissertação são:

- Estudo sobre o papel do auditor externo de QoS em grades computacionais e as consequências da sua ausência para a relação consumidor-provedor;
- Definição e avaliação teórica e prática de um mecanismo interoperável e modular para negociar e estabelecer um auditor de QoS externo em ambientes de grade computacional;
- Integração do mecanismo proposto com o principal padrão aberto de acordo de nível de serviço, o WS-Agreement, e com um dos mais usados *middlewares* de grade computacional, o Globus Toolkit.

A dissertação está organizada da seguinte forma. No Capítulo 2 é apresentado o conceito de computação em grade e suas particularidades. A qualidade de serviço aplicada à computação em grade é discutida no Capítulo 3. A auditoria de qualidade de serviço em grades computacionais é apresentada e discutida no Capítulo 4. No Capítulo 5 é apresentado o projeto do mecanismo de negociação de auditor externo para grades computacionais. A implementação do protótipo do mecanismo de negociação de auditor é apresentada no Capítulo 6. No Capítulo 7 são apresentados os resultados teóricos e experimentais do mecanismo proposto. Finalmente, conclusões e trabalhos futuros são apresentados no Capítulo 8.

# Capítulo 2

# Computação em Grade

## 2.1 Visão Geral

(Foster e Kesselman, 1998) definem uma grade computacional como uma infraestrutura de software e hardware que provê acesso a um alto poder de processamento e armazenamento de forma confiável, interoperável, pervasiva e com baixo custo. Para prover esse alto poder computacional, as infraestruturas de grade coordenam recursos computacionais de vários domínios administrativos, a fim de executar aplicações que levariam muito tempo para executar em um único recurso. A combinação desses recursos computacionais de diferentes domínios para realizar uma tarefa é caracterizada por *organizações virtuais*.

O conceito de organização virtual é descrito em (Foster et al., 2001) como um conjunto de indivíduos ou instituições relacionados por regras de compartilhamento de recursos, tais como o que é compartilhado, quem tem permissão para compartilhar e as condições sob as quais um compartilhamento ocorre. Assim, essas organizações virtuais são essencialmente dinâmicas, uma vez que indivíduos podem ser adicionados ou removidos de uma organização pela simples mudança nas regras de compartilhamento que a definem. Além disso, essas organizações também são potencialmente multi-institucionais, uma vez que podemos ter mais de uma instituição aplicando as mesmas regras de compartilhamento dentro de uma grade computacional.

Segundo (Foster e Kesselman, 1998), os maiores desafios nas infraestruturas de grade são o compartilhamento coordenado de recursos computacionais e a resolução de problemas em organizações virtuais dinâmicas e multi-institucionais. É importante observar, porém, que o principal tipo de compartilhamento que ocorre em grades computacionais não é o compartilhamento de arquivos, mas o acesso direto a computadores, software, dados e outros recursos. Esse tipo de compartilhamento é requerido por várias estratégias de resolução colaborativa de problemas na indústria, ciência e engenharia modernas e requer um ambiente bastante controlado de colaboração e compartilhamento entre os recursos computacionais.

Assim, o motivo principal para se buscar uma grade computacional como plataforma de execução é a necessidade de um alto poder de processamento (não fornecido por computadores comuns de mercado) a um preço mais acessível do que o de utilização de um supercomputador. Por isso, as aplicações executadas em uma grade computacional, normalmente, são testes de modelos e simulações de problemas complexos, diagnósticos de condições médicas, previsão do tempo, entre outras. As grades computacionais fornecem

a infraestrutura de software e de hardware para interconectar recursos computacionais e gerenciar o conjunto resultante, de forma a prover o alto desempenho requerido por tais aplicações.

(Foster e Kesselman, 1998) também relacionam algumas características essenciais de uma grade computacional, como o provimento de um serviço confiável, interoperável, de baixo custo e com acesso pervasivo.

Por serviço confiável, assume-se que os usuários precisam de garantias de que receberão um desempenho previsível, contínuo e de alto nível dos diversos componentes que constituem a grade e que, na ausência de tais garantias, provavelmente as aplicações para rodar na grade simplesmente não seriam escritas. É importante observar que as garantias desejadas variam de aplicação para aplicação, e podem incluir segurança, confiabilidade, serviços de software, poder de processamento, latência e banda de comunicação.

Além disso, os usuários precisam de serviços padronizados, acessíveis por interfaces padronizadas, e operando dentro de parâmetros padronizados, assim como a rede elétrica. Ou seja, a grade precisa prover um serviço interoperável, senão o desenvolvimento de aplicações e o acesso pervasivo seriam impraticáveis.

Quanto ao baixo custo, os usuários precisam pagar um preço aceitável pelo acesso à grade, de forma que o custo do acesso não seja superior ao benefício trazido por ele. Ou seja, como todo serviço, a grade computacional também precisa de um atrativo econômico para que seja aceita e usada por um grande número de usuários.

Por fim, os usuários precisam de acesso a todos os serviços a qualquer momento, não importando em que ambiente estejam. Ou seja, a grade precisa ser pervasiva. Isto não implica que os recursos estejam em todos os lugares ou que sejam acessíveis universalmente, mas que, de dentro da grade, todos os serviços estejam sempre acessíveis, não importando o tamanho da mesma.

(Foster e Kesselman, 1998) acreditam que uma grade computacional construída com base nesses princípios seja capaz de produzir um efeito de transformação na maneira como a computação é construída e usada.

## 2.2 Vantagens e Desvantagens

Diferentemente dos supercomputadores, que são construídos a partir de vários processadores conectados através de uma rede de interconexão de alta capacidade, as grades computacionais são constituídas de recursos computacionais completos (processador, memória, interfaces de entrada e saída) interligados por redes convencionais de comunicação. Assim, uma das vantagens das grades é que elas podem ser criadas a partir de recursos computacionais comuns de mercado, com arquiteturas e sistemas operacionais tradicionais, ao invés dos sistemas e arquiteturas específicos e caros encontrados nos supercomputadores.

Por outro lado, a falta de uma rede de interconexão de alta capacidade conectando os diversos processadores e unidades de armazenamento se apresenta como um limitador de desempenho para as grades computacionais. Assim, essas plataformas são mais adequadas para a execução de aplicações que podem ser divididas em tarefas paralelizáveis e independentes (*bag-of-tasks applications*), evitando assim o alto *overhead* de uma troca de informações via as redes tradicionais de comunicação (Cirne et al., 2003).

Outra vantagem das grades computacionais é que, como os recursos que as compõem são bastante desacoplados e gerenciados de forma descentralizada, elas proporcionam um

alto nível de escalabilidade. Assim, é relativamente simples ampliar o número de recursos disponíveis na grade, sem aumentar significativamente o esforço de gerenciamento ou degradar o desempenho da plataforma (Kondo et al., 2004).

Contudo, como os recursos computacionais que compõem uma grade computacional pertencem a diferentes domínios administrativos, eles podem não ser inerentemente confiáveis ou estar sempre disponíveis. Na prática, isso significa que recursos que estejam realizando alguma tarefa na grade podem, a qualquer momento, se desconectar desta antes de terminar suas tarefas. Em outras situações, esses mesmos recursos também podem retornar resultados errados maliciosamente (Foster et al., 2001). Assim, os desenvolvedores de aplicações para essas plataformas podem precisar implementar código para proteger suas aplicações contra esses tipos de falhas e comportamentos maliciosos, como, por exemplo, implementar alguma redundância.

Além dos problemas de confiança e disponibilidade, é preciso observar que a grade é normalmente composta por recursos heterôgeneos tanto em hardware como em software. Assim, os recursos potencialmente possuem arquiteturas de hardware diferentes, bem como executam sistemas operacionais diversos. Esse fato requer que desenvolvedores e usuários de aplicações para grades computacionais selecionem os recursos compatíveis com suas aplicações ou criem aplicações multi-plataformas.

O impacto desses problemas de confiança, disponibilidade e heterogeneidade no desempenho e no desenvolvimento de aplicações para essas plataformas é muito importante para decidir se é viável executá-las em uma grade computacional ou se é preferível executá-las em um ambiente mais dedicado e homogêneo, como um *cluster*.

Do ponto de vista dos fornecedores de recursos, a disponibilização de um recurso na grade pode exigir um esforço extra em segurança, uma vez que o recurso será acessado por qualquer pessoa com acesso à grade computacional. Normalmente, esse esforço se traduz na criação e gerenciamento de máquinas virtuais, a fim de limitar o escopo de atuação de uma possível aplicação maliciosa que venha a ser executada na grade (Kertesz et al., 2009).

## 2.3 Execução de Tarefas

A execução de tarefas em grades computacionais normalmente envolve a utilização de um middleware. Um middleware pode ser visto como uma camada de abstração entre o ambiente de execução da grade computacional e as aplicações dos usuários. Essencialmente, o middleware é responsável por abstrair as características específicas de cada recurso da grade, de forma a torná-las transparentes para as aplicações. Assim, o desenvolvimento de aplicações para executar em grades computacionais começa pela escolha do middleware a ser utilizado.

O Globus Toolkit é um middleware de grade computacional apoiado pelo Open Grid Forum (OGF) (ope, 2010), que também criou e mantém a especificação do WS-Agreement (Andrieux et al., 2007). O Globus Toolkit é desenvolvido principalmente nas linguagens Java e C e possui módulos para tratar diferentes aspectos da programação para grades computacionais. Os principais módulos do Globus Toolkit tratam de segurança, gerenciamento de dados, gerenciamento de execução de tarefas e serviços de informação. O módulo de segurança trata de questões como autenticação de usuários e autorização de uso dos serviços da grade. O módulo de gerenciamento de dados trata da localização,

transferência e gerenciamento dos dados distribuídos na grade. O módulo de gerenciamento de execução, por sua vez, refere-se a iniciação, monitoramento, gerenciamento, escalonamento e coordenação das computações remotas executadas na grade. O módulo de serviços de informação é responsável por monitorar e descobrir recursos e serviços na grade computacional. Além desses, o Globus Toolkit também possui um módulo de bibliotecas comuns a todas essas áreas, chamado de *Common Runtime* (Foster, 2005).

Uma vez definido o middleware, o desenvolvedor deve utilizar os recursos disponibilizados por este para dividir a aplicação em pedaços, normalmente chamados de tarefas. Normalmente, a definição de uma aplicação e suas tarefas é realizada através de uma linguagem específica, como a *Job Submission Description Language* (JSDL) (Anjomshoaa et al., 2005). (Amin et al., 2006) apresentam algumas características de uma tarefa, como:

- identidade: um identificador que identifica unicamente uma tarefa;
- especificação: uma descrição especificando os detalhes de uma tarefa, incluindo parâmetros e outros requisitos;
- contexto de segurança: uma credencial de segurança associada à tarefa e que será usada para autenticar a tarefa no ambiente de execução;
- contato de serviço: o endereço de um serviço do ambiente de execução que será usado para executar essa tarefa, como, por exemplo, um serviço de transferência de arquivo; e
- status: o progresso da execução da tarefa, permitindo a aplicação monitorar sua execução (não submetida, submetida, ativa, suspensa, cancelada, falhou ou completada).

Depois de definir as tarefas a serem executadas, estas são submetidas para execução na grade através de um *broker*. O escalonamento das tarefas enviadas pelo *broker* ocorre por intermédio de um sistema gerenciador de recursos disponível na grade computacional. O gerenciador de recursos identifica os recursos disponíveis e os escalona para executar as tarefas compatíveis, de acordo com seus requisitos.

O Torque/PBS (tor, 2011) é um gerenciador de recursos *open source* que provê controle sobre a execução de tarefas em nós distribuídos na grade computacional. O Torque/PBS, portanto, é responsável por aceitar tarefas a serem executadas na grade, adicioná-las a uma fila e submetê-las para execução. O Torque/PBS pode ser integrado ao Globus Toolkit, de forma que, quando o Globus Toolkit recebe uma tarefa a ser executada, ele transfere essa responsabilidade para o Torque/PBS. O uso do Torque/PBS em conjunto com o Globus Toolkit oferece mais controle e flexibilidade sobre a execução das tarefas e sobre os recursos da grade computacional, sendo uma opção *open source* muito utilizada para gerenciar recursos de grades computacionais.

As tarefas submetidas ao gerenciador de recursos são escalonadas e transferidas para os recursos da grade para execução. A partir de então, o usuário pode monitorar a execução das tarefas na grade através do *broker*. Quando uma aplicação falha, os mecanismos de tolerância a falhas, se presentes, entram em ação para contornar o problema. Ao terminar a execução de uma tarefa, os resultados são transferidos para o usuário (Cirne et al., 2003).

Apesar de sua estreita relação com os padrões do OGF, o Globus Toolkit na sua versão 4.0.8 não apresenta suporte nativo à especificação do WS-Agreement. Assim, para obter suporte ao WS-Agreement, o Globus Toolkit precisa ser complementado com a instalação de extensões desenvolvidas por terceiros. Uma dessas extensões é o pacote de serviços Web chamado SLA4D-Grid Negotiation Manager (sla, 2011). O SLA4D-Grid Negotiation Manager é um pacote de serviços que implementa estritamente a especificação do WS-Agreement 1.0 (Andrieux et al., 2007) e que se integra facilmente com o Globus Toolkit. O padrão WS-Agreement para acordos de nível de serviço será detalhado na Seção 3.2.2.

## 2.4 Arquiteturas de Grades Computacionais

### 2.4.1 Arquitetura de Protocolos

Conforme mencionado na Seção 2.1, uma das características importantes de uma grade computacional é a sua interoperabilidade. A interoperabilidade está relacionada com a padronização dos serviços e interfaces disponíveis e tem o papel de viabilizar relações de compartilhamento entre quaisquer entidades, com seus ambientes de programação, linguagens e plataformas específicos. Dessa forma, a interoperabilidade contribui para a criação e expansão dinâmica de organizações virtuais.

Para conseguir essa interoperabilidade,(Foster et al., 2001) propõem uma arquitetura para grades computacionais essencialmente baseada em protocolos padronizados, pelos quais os usuários e recursos de organizações virtuais negociam, estabelecem, gerenciam e exploram relações de compartilhamento. O estabelecimento de protocolos padronizados tem o papel importante de definir como essas interações entre entidades devem ocorrer, bem como a estrutura da informação trocada entre elas.

Assim, a construção de serviços para as grades, bem como sua portabilidade e extensibilidade, são facilitadas pela pilha de protocolos padronizados. Sobre essa pilha de protocolos estão as interfaces de programação de aplicação (APIs) e o kits de desenvolvimento de *software* (SDKs), que constituem o *middleware* da grade computacional. Esse *middleware* é importante para que os desenvolvedores possam construir e operar suas aplicações de forma padronizada. Além disso, o *middleware* também facilita o compartilhamento de código e provê portabilidade às aplicações.

Através dos protocolos estabelecidos, é possível criar serviços padronizados para acesso a computação e dados, descoberta de recursos, replicação de dados, entre outros, de forma a ampliar os serviços disponíveis e abstrair detalhes específicos dos recursos da grade para os participantes das organizações virtuais.

A Figura 2.1 apresenta as camadas da arquitetura de protocolos para grades computacionais. Os componentes de cada camada possuem características comuns e são construídos sobre as abstrações das camadas inferiores.

As camadas dessa arquitetura foram definidas de acordo com os princípios do modelo da ampulheta (Council, 1994). De acordo com esse modelo, a parte estreita da ampulheta define um pequeno conjunto de abstrações e protocolos essenciais sobre os quais muitos comportamentos de mais alto nível podem ser construídos, e sob os quais reside uma base de diferentes tecnologias (a base da ampulheta). Na arquitetura de protocolos da Figura 2.1, a parte estreita compreende as camadas Recurso e Conectividade, que facilitam o compartilhamento de recursos isolados.

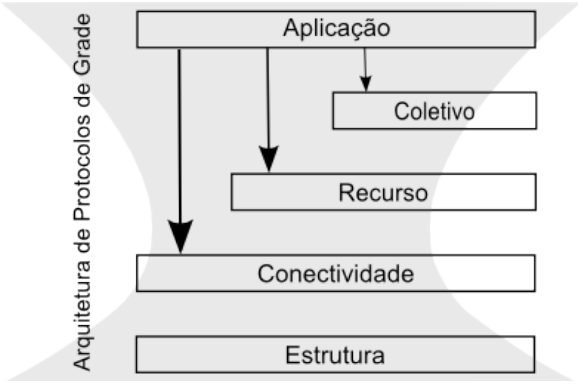


Figura 2.1: Arquitetura de protolocos da grade computacional. Fonte: (Foster et al., 2001)

A camada Estrutura representa os diversos tipos de recursos que podem ser compartilhados em uma grade computacional, como recursos de computação, sistemas de armazenamento, catálogo, recursos de rede, sensores, sistemas de arquivo distribuídos, *clusters*, entre outros. Os componentes dessa camada implementam operações específicas que ocorrem em cada um desses recursos como resultado do compartilhamento provido nas camadas superiores. Em especial, dois tipos de operação são muito importantes: consulta e gerenciamento de recursos. Operações de consulta revelam a estrutura, o estado e as capacidades de um recurso, como arquitetura, sistema operacional, capacidade de armazenamento, entre outros. As operações de gerenciamento, por sua vez, fornecem controle da qualidade do serviço entregue por esses recursos.

A camada Conectividade consiste de protocolos de comunicação e de segurança para atender aos requisitos das operações executadas na grade. Os protocolos de comunicação visam prover a base para a troca de informações entre os recursos da camada Estrutura, enquanto os protocolos de segurança proveem mecanismos de autenticação para verificar a identidade de usuários e recursos.

A camada Recurso é composta por protocolos relacionados com o compartilhamento de recursos isolados. Assim, os protocolos dessa camada usam os protocolos das camadas inferiores para informar e gerenciar os compartilhamentos de cada recurso. Como na camada Estrutura, existem dois tipos de protocolos na camada Recurso: protocolos de informação e de gerenciamento. Contudo, as informações e o gerenciamento dizem respeito ao compartilhamento de um determinado recurso. Por exemplo, através dessa camada é possível consultar a carga atual e a política de uso de um determinado recurso, bem como negociar, iniciar e monitorar o seu compartilhamento.

A camada Coletivo, em complemento à camada Recurso, provê componentes de informação e gerenciamento de compartilhamento dos recursos em coletividade. Assim, os componentes dessa camada capturam e gerenciam informações sobre o estado global dos recursos e suas interações entre si. Como exemplo dos serviços providos por essa camada, podemos citar os serviços de diretório, de monitoração e diagnóstico, escalonamento, gerenciamento de carga, entre outros.

A camada Aplicação comporta as aplicações dos usuários, que executam dentro de uma organização virtual. As aplicações são construídas a partir dos serviços e protocolos disponíveis em quaisquer das camadas inferiores. Além disso, elas podem ser construídas

sobre *frameworks* e bibliotecas, os quais também são considerados como aplicações na arquitetura de protocolos.

Em (Foster et al., 2002), os autores estendem a discussão a respeito da arquitetura de uma grade computacional, argumentando sobre a necessidade não apenas de protocolos padronizados, mas também de serviços padronizados construídos sobre esses protocolos. Essa discussão leva a uma proposta de evolução da arquitetura de protocolos em direção a uma arquitetura de serviços de grade. Essa arquitetura de serviços é discutida na seção 2.4.2.

### 2.4.2 Open Grid Services Architecture (OGSA)

(Foster et al., 2002) apresentam a grade computacional como um conjunto extensível de *serviços de grade*, que podem ser agregados de várias formas para atender às necessidades das organizações virtuais. Esses serviços de grade definem os comportamentos e as interfaces necessárias para apoiar a integração de sistemas distribuídos no contexto de uma grade computacional. As organizações virtuais, por sua vez, passam a ser definidas, em parte, pelos serviços de grade que operam e compartilham.

Com o intuito de viabilizar o desenvolvimento dos serviços de grade de forma padronizada, (Foster et al., 2002) propõem um alinhamento entre estes e os Web Services (Kreger, 2001) (Graham et al., 2001). Este alinhamento facilita o desenvolvimento dos serviços de grade, uma vez que os Web Services estão apoiados sobre padrões abertos e bem difundidos. À definição desse alinhamento e de um conjunto básico de Web Services padronizados para dar suporte às grades computacionais, dá-se o nome de *Open Grid Services Architecture* (OGSA) (Foster et al., 2006).

A OGSA, portanto, pode ser definida como uma arquitetura padrão para a construção de grades computacionais baseadas em Web Services. Ela se apóia em padrões como a *Web Service Description Language (WSDL)*[1] (Christensen et al., 2001) e o *Simple Object Access Protocol (SOAP)* (Gudgin et al., 2007), mas também propõe extensões para atender aos requisitos de interoperabilidade e compartilhamento de recursos heterogêneos, inerentes às grades computacionais. Assim, contrói-se um sistema de serviços de grade distribuídos e baseados em padrões abertos, onde cada serviço de grade é uma instância de um Web Service com características especiais. Algumas dessas características são o suporte a manutenção de estado, invocação segura e confiável, gerenciamento de ciclo de vida, notificação, gerenciamento de políticas, gerenciamento de credenciais, e virtualização.

Para construir esse padrão, o grupo de trabalho de definição da OGSA pesquisou casos de uso de grades, a fim de identificar os requisitos de uma arquitetura de grade computacional moderna (Foster et al., 2006). Os requisitos identificados foram: (i) interoperabilidade e suporte a ambientes heterogêneos e dinâmicos, (ii) compartilhamento de recursos entre organizações, (iii) otimização de uso de recursos, (iv) garantia de qualidade de serviço, (v) execução de tarefas, (vi) serviços de gerenciamento de dados, (vii) segurança, (viii) redução de custos administrativos, (ix) escalabilidade, (x) disponibilidade e (xi) facilidade de uso e extensibilidade.

[1] A WSDL é uma linguagem baseada em XML utilizada para definir a interface pública de um serviço Web. Assim, todas as operações e mensagens que são parâmetros de entrada ou saída de um serviço Web são apresentados na interface WSDL do serviço.

A fim de atender aos requisitos observados, uma grade computacional compatível com o padrão OGSA é descrita em termos de 7 capacidades de serviços (Foster et al., 2006):

1. Serviços de Infraestrutura: compatibilidade com a arquitetura de Web Services;
2. Serviços de Gerenciamento de Execução: instanciação, execução e gerenciamento de unidades de trabalho;
3. Serviços de Dados: gerenciamento de acesso, atualização e transferência de dados;
4. Serviços de Gerenciamento de Recursos: gerenciamento de recursos físicos, lógicos e da infraestrutura da grade;
5. Serviços de Segurança: objetivos, modelos, capacidades funcionais e propriedades dos serviços de segurança;
6. Serviços de Auto-gerenciamento: possuir componentes de hardware e software auto-gerenciáveis, a fim de reduzir custos; e
7. Serviços de Informação: acesso e manipulação de informações sobre aplicações, recursos e serviços da grade.

Em (Foster et al., 2006), são descritas cada uma dessas capacidades de serviço em alto nível, em termos de objetivos, abordagem, modelos e propriedades, bem como são apresentadas as relações entre os serviços. Em nível de implementação, porém, os Web Services ainda precisavam de ajustes para atender a alguns requisitos dos serviços de grade, tais quais especificados na OGSA. Por exemplo, os Web Services tradicionais não mantêm estado, enquanto alguns serviços de grade requerem a manutenção de estado durante uma interação consumidor-provedor.

Uma solução para o problema da manutenção de estado em Web Services havia sido proposta pelo *Open Grid Forum (OGF)*[2](ope, 2010) já em 2003, com o nome de *Open Grid Services Infrastructure (OGSI)* (Tuecke et al., 2003). O OGSI definia um modelo de componente, usando uma extensão da WSDL, para introduzir o conceito de instâncias de Web Services com suporte a manutenção de estado, além de notificação assíncrona de mudança de estado, referências a instâncias de serviços, coleções de instâncias de serviços, entre outras funcionalidades.

Mas, apesar dos esforços da comunidade de grade, o padrão OGSI não foi bem aceito pelo grupo de desenvolvimento dos Web Services. Em 2006, então, o Open Grid Forum o substituiu pelo *Web Services Resource Framework* (WSRF)[3](Czajkowski et al., 2004b). O WSRF é, basicamente, uma versão refatorada do OGSI e também mais compatível com os avanços na tecnologia de Web Services da época. Desde então, o WSRF tem sido o padrão de infraestrutura de serviços para grades computacionais compatíveis com a OGSA.

Para esclarecer melhor a relação entre OGSA, WSRF e Web Services, a Figura 2.2 apresenta o posicionamento de cada um desses conceitos na construção de um serviço de grade.

[2]O OGF é a comunidade de usuários, desenvolvedores e fabricantes, também responsável pelo OGSA, que lidera os esforços globais de padronização para grades computacionais.

[3]O mapeamento dos elementos do padrão OGSI para os elementos do padrão WSRF é descrito em (Czajkowski et al., 2004a).

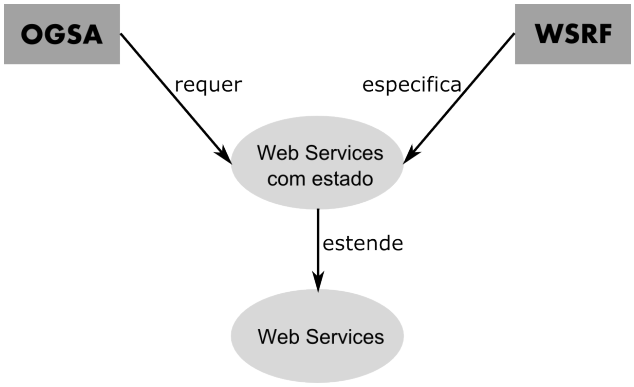


Figura 2.2: Relacionamento entre OGSA, WSRF e Web Services. Fonte: (Czajkowski et al., 2004b)

(I. Foster e Snelling, 2006) descrevem como os serviços de infraestrutura da OGSA são suportados pelo padrão WSRF. Basicamente, a infraestrutura da OGSA é atendida pelo WSRF através de 5 especificações:

1. WS-Resource: define um WS-Resource como o estado a ser mantido e um Web Service através do qual o estado é acessado;

2. WS-ResourceProperties: descreve uma interface para associar recursos a um conjunto de valores tipados;

3. WS-ResourceLifetime: descreve uma interface para gerenciar o ciclo de vida de um WS-Resource;

4. WS-BaseFaults: descreve um mecanismo extensível para tratamento de falhas baseado no padrão *SOAPFaults*; e

5. WS-ServiceGroup: descreve interface para operar sobre coleções de WS-Resources.

O compromisso principal do WSRF é acrescentar manutenção de estado entre invocações aos Web Services. Assim, ele provê um conjunto de operações, definido nas 5 especificações, que os Web Services podem implementar para obter essa capacidade. Na prática, consumidores de serviços se comunicam com *serviços de recursos* para armazenar ou recuperar informações. Quando os consumidores precisam fazer referência a uma informação armazenada previamente, eles informam ao Web Service o identificador do recurso específico onde está a informação. Além disso, o conjunto de operações providas pelo WSRF também permite a atribuição e recuperação de propriedades dos recursos, que podem ser usadas para controlar o estado de cada recurso.

# Capítulo 3

# Qualidade de Serviço em Grades

## 3.1 Visão Geral

O aumento do número de dispositivos conectados a redes de computadores, associado ao crescente número de aplicações desenvolvidas para usar o potencial dessas redes, gera um aumento significativo do tráfego de dados. Por conseguinte, o tráfego intenso por essas redes leva a um problema de congestionamento que, se não for tratado adequadamente, pode inviabilizar o uso de aplicações que requerem grande largura de banda. As consequências de um tráfego de dados congestionado sobre uma aplicação de vídeo-conferência, por exemplo, podem ser observadas na falta de sincronia entre áudio e vídeo e na falta de fluidez do vídeo.

Uma possível solução para o problema é o fornecimento de recursos extras a fim de suportar os picos de tráfego na rede. Assim, é possível obter um alto nível de qualidade sem fornecer nenhum tipo de garantia de desempenho ao usuário final (Maryni e Davoli, 2000). Essa abordagem, porém, tem um custo muito elevado, uma vez que os recursos normalmente são caros e podem acabar sendo subutilizados durante o tráfego normal.

Outra abordagem, denominada *qualidade de serviço*, trabalha com o fornecimento de garantias ao usuário sobre o desempenho da aplicação (Itu, 1994). A qualidade de serviço na área de redes de computadores pode ser tratada através da aplicação de mecanismos de reserva de recursos, a fim de garantir que uma aplicação para a qual um dado recurso (largura de banda, por exemplo) é essencial não seja afetada por outra para a qual aquele recurso não é prioritário (Marchese, 2007). Assim, a qualidade de serviço em redes provê a habilidade de priorizar recursos de forma individual a diferentes aplicações, usuários e fluxos de dados.

Vários protocolos e modelos se propõem a adicionar qualidade de serviço a redes de comunicação por meio da reserva de recursos (R. Braden e Shenker, 1994) (Blake et al., 1998). Um dos principais é o Resource Reservation Protocol (RSVP) (Braden et al., 1997), que foi proposto pelo Internet Engineering Task Force (IETF) (iet, 2010) e está relacionado com a reserva de recursos como largura de banda, tempo de CPU e buffers, por exemplo. Em protocolos desse tipo, a qualidade de serviço é especificada através do preenchimento de campos especiais no cabeçalho dos pacotes de dados transferidos pela rede. Esses campos nos cabeçalhos dos pacotes de dados são interpretados pelos nós da rede e, assim, o fluxo pode ser priorizado em relação a outros pacotes de dados.

A reserva de recursos, porém, não é a única interpretação para qualidade de serviço. Na área de telefonia, por exemplo, a qualidade de serviço foi definida pela International Telecommunication Union (itu, 2010) de forma bastante ampla como "um conjunto de requisitos de qualidade sobre o comportamento coletivo de um ou mais objetos"(ITU, 1995). Esses requisitos de qualidade estão relacionados com o desempenho do serviço prestado em relação a suporte, acessibilidade, integridade, segurança, retenção e operabilidade (Itu, 1994). Nesse contexto, a especificação da qualidade de serviço normalmente é estabelecida entre provedor e consumidor através de contratos de qualidade chamados *acordos de nível de serviço*.

No contexto das grades computacionais, diz-se que um provedor de serviços tem suporte a qualidade de serviço (QoS) quando ele permite ao consumidor definir ou negociar um conjunto de requisitos não-funcionais (vazão, disponibilidade, acessibilidade, entre outros) para o serviço em questão, sob determinadas condições de uso (taxa de requisição e volume de dados de entrada, entre outras), que, uma vez estabelecidos, são garantidos pelo provedor (Sahai et al., 2003). Por exemplo, um serviço de *streaming* de vídeo pode requerer uma grande largura de banda de comunicação mínima, de modo que, se esta não for garantida, o serviço se torna inutilizável. Assim, o consumidor do serviço deve negociar tal requisito com o provedor, que, em contrapartida, negocia o número máximo de usuários simultâneos do serviço. Além disso, são negociadas as formas de monitoramento desses valores e as multas a serem aplicadas tanto em caso de não disponibilização da largura de banda mínima como em caso de número máximo de usuários simultâneos excedido. O nível de qualidade de serviço, portanto, resulta de um acordo entre consumidor e provedor, ambos com direitos e deveres contratuais.

Conforme mencionado anteriormente, na área de telefonia esses acordos são os acordos de nível de serviço. Devido a sua simplicidade e grande utilização na telefonia, o uso desses acordos foi estendido para o contexto das plataformas de grade computacional. Assim, os acordos de nível de serviço também passaram a ser usados como instrumentos de garantia de qualidade nesses ambientes.

Com o surgimento e a adoção de conceitos como *Software as a Service* (SaaS) e *Infrastructure as a Service* (IaaS), a definição e a garantia da qualidade de serviço se tornaram imprescindíveis para a exploração comercial de grades computacionais. Até então, as grades baseavam-se no modelo do melhor-esforço, que não garante nenhuma qualidade ao consumidor e, assim, dificultava a comercialização de serviços nessas infraestruturas. A introdução de qualidade de serviço em grades requer um controle maior da utilização dos recursos disponíveis, a fim de atender aos requisitos de qualidade. Esse maior controle, por sua vez, apresenta diversos desafios e problemas a serem superados.

Para exemplificar os problemas relacionados a uma transação comercial entre um consumidor e um provedor de recursos de grade com suporte a QoS, consideremos a seguinte interação. O consumidor da grade tem um problema e cria uma aplicação para resolver esse problema. Tal aplicação tem requisitos de recursos (memória, processador, banda, armazenamento) e/ou de ambiente (*backup*, privacidade, segurança, disponibilidade, tempo de resposta). O consumidor não tem uma infraestrutura (recursos e ambiente) adequada para executar a aplicação e precisa, então, encontrar a infraestrutura com melhor relação custo/benefício. Assim, ele descreve os requisitos da infraestrutura requerida em um documento digital e o envia a vários provedores de grade. Cada provedor analisa a proposta e pode retornar ou um valor monetário que representa o custo do aluguel da infraestrutura

descrita ou enviar uma contra-proposta ao consumidor. O consumidor recebe a(s) proposta(s) do(s) provedores, as analisa e, possivelmente, envia uma nova contra-proposta. A análise da proposta no consumidor é realizada com base na sua satisfação com a proposta, em comparação com outras propostas recebidas, e, possivelmente, em alguma preferência por um provedor específico. O provedor pode aceitar, recusar ou enviar uma nova proposta ao consumidor. Em caso de aceitação pelo provedor, este deve enviar ao consumidor um contrato ou acordo de serviço, juntamente com um prazo para efetivação do pagamento. O consumidor, finalmente, envia o comprovante de pagamento juntamente com os dados da aplicação no prazo estipulado e o provedor, então, começa a execução da aplicação em paralelo com a monitoração do nível de serviço. Em caso de violação do nível de serviço especificado no acordo, o provedor desconta o valor da multa da dívida do consumidor ou gera um reembolso.

Nessa interação simplificada, que representa uma transação de aluguel de recursos de grade para execução de uma aplicação, podemos identificar diversos problemas: especificação da proposta de serviço de maneira flexível, interoperável, compacta e fácil de usar e entender; cálculo do custo monetário do aluguel da infraestrutura; garantia do nível de serviço em tempo de execução; cálculo da multa em caso de violação do nível de serviço; monitoramento eficiente da QoS em tempo de execução com baixo *overhead*; garantia de que o provedor está monitorando o serviço conforme o acordo de serviço e que está sinalizando o consumidor sobre possíveis violações; segurança e privacidade dos dados.

Essas são apenas algumas das questões para as quais se tem buscado soluções em diversas pesquisas relacionadas com o provimento de QoS em grades computacionais. Cada uma dessas questões está relacionada com um ou mais dos seguintes temas: arquitetura de grade com suporte a QoS, especificação de acordos, negociação de acordos, monitoramento de nível de serviço, auditoria de qualidade de serviço, violação de acordo, segurança, reserva e escalonamento de recursos. Existem diversas pesquisas e propostas de soluções em cada uma dessas áreas. Nas seções seguintes, são apresentadas algumas soluções relacionadas com os problemas de especificação de acordos, negociação de acordos, escalonamento de recursos com base em acordos e monitoramento de acordos.

## 3.2 Acordos de Nível de Serviço

A qualidade de serviço normalmente é estabelecida em um documento contratual chamado Acordo de Nível de Serviço (SLA) (Jin et al., 2002). Esse documento define de forma clara e não-ambígua os requisitos não-funcionais de um ou mais serviços, bem como as condições de uso, a forma de monitoramento e as multas a serem aplicadas a cada violação de contrato. Em linhas gerais, em um acordo de nível de serviço estão representados as expectativas e os entendimentos mútuos entre consumidor e provedor em relação a um serviço. Os termos desse acordo, normalmente, são negociados até que se chegue a um consenso entre as partes e, a partir desse momento, servirá como entrada para o monitoramento da qualidade de serviço e a possível cobrança de multas.

Um acordo de nível de serviço típico possui as seguintes seções de informação:

- Objetivo: descreve o motivo da criação do acordo.
- Partes envolvidas: descreve as partes envolvidas no acordo e seus papéis (consumidor, provedor).

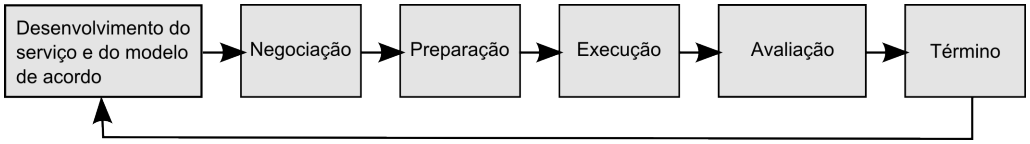


Figura 3.1: Ciclo de vida de um acordo de nível de serviço. Fonte: (Bianco et al., 2008)
0

- Período de validade: define o período de vigência do acordo.
- Escopo: define os serviços e recursos envolvidos no acordo.
- Objetivos de nível de serviço: define os níveis de serviço acordados entre as partes, que normalmente inclui os indicadores de nível de serviço (disponibilidade, tempo de resposta) e suas metas.
- Penalidades: define o que acontece quando uma das partes não cumpre com as metas estabelecidas nos objetivos de nível de serviço.
- Administração: descreve os processos de verificação do níveis de serviço acordados e a responsabilidade organizacional sobre a gestão desses processos.

Um acordo de nível de serviço passa por diversas etapas durante o seu ciclo de vida. A automação dessas etapas é bastante relevante para viabilizar o provimento dinâmico de serviços entre organizações. A Figura 3.1 apresenta o fluxo das etapas do ciclo de vida de um acordo de nível de serviço. Essas etapas são:

- Desenvolvimento do Serviço e do Modelo de Acordo: inclui a identificação das necessidades do consumidor do serviço, a identificação de características e parâmetros de serviço apropriados que podem ser oferecidos pelo ambiente de execução, e a preparação de modelos padrões de acordo;
- Negociação: inclui a negociação de valores específicos para os parâmetros de serviço definidos, os custos para o consumidor do serviço, os custos para o provedor do serviço, e a definição e periodicidade dos relatórios a ser fornecidos ao consumidor;
- Preparação: inclui a preparação do serviço a ser consumido, bem como a configuração dos recursos de execução, a fim de adequá-los aos valores definidos para os parâmetros do acordo;
- Execução: essa etapa representa a execução de fato do serviço, que inclui monitoramento, geração de relatórios em tempo de execução, validação da qualidade de serviço e processamento de violações do acordo;
- Avaliação: inclui a revisão dos níveis de qualidade de serviço que são providos a um consumidor específico, com base na sua satisfação em relação ao serviço; e revisão dos níveis de qualidade de serviço de forma geral, para todos os consumidores, do ponto de vista do provedor de serviços; e
- Término: representa o término da disponibilização do serviço, seja por finalização ou por violação do contrato.

O uso de SLAs como forma de garantir QoS em ambientes de grade requer que a confidencialidade dos acordos seja garantida, uma vez que os SLAs trafegarão por diferentes organizações virtuais provedoras de grade. Além disso, é interessante que haja uma representação única dos SLAs, para que estes sejam trocados e entendidos pelas diferentes organizações virtuais envolvidas no provimento de um serviço.

Atualmente, existem várias propostas de implementação de SLAs para grades computacionais. Duas dessas propostas são o WSLA (*Web Service Level Agreement*) (Keller e Ludwig, 2003), da IBM e o WS-Agreement, desenvolvido pelo *Open Grid Forum* (OGF) (Andrieux et al., 2007).

### 3.2.1 WSLA

O *Web Service Level Agreement* da IBM foi uma das primeiras propostas de linguagem de especificação de acordos de nível de serviço para *Web Services*. Embora a primeira versão da especificação tenha sido publicada como padrão em 2003, o WSLA foi introduzido no *Web Services Toolkit* (WSTK) versão 3.2, em janeiro de 2000.

A linguagem de definição do WSLA é baseada em XML (eXtensible Markup Language) e, através dela, é possível definir várias informações relativas a um acordo de nível de serviço, como as obrigações das partes envolvidas no acordo, as medidas a serem tomadas em caso de falha em atingir as metas de qualidade acordadas, as operações de monitoramento e gerenciamento do serviço, entre outras. É possível, inclusive, definir uma terceira entidade que contribua para a medição das métricas, a supervisão de garantias e até o gerenciamento de desvios de garantias de serviço.

Contudo, o WSLA apenas trata da visão acordada comum entre as partes envolvidas. Isto significa que as partes ainda possuem liberdade para definir a política de implementação de um serviço e sua supervisão.

#### Papel e características do WSLA

O WSLA pode ser usado tanto pelo provedor quanto pelo consumidor para configurar seus sistemas a fim de prover e supervisionar os serviços. Este processo, chamado de *implantação*, interpreta o WSLA e toma as ações necessárias, como a criação e parametrização dos serviços e dos sistemas de supervisionamento da qualidade. Essas ações e o processo de implantação como um todo são específicos para cada parte envolvida no acordo e, devido à formalização não-ambígua do WSLA, podem ser completamente automatizados (Keller e Ludwig, 2003).

A Figura 3.2 ilustra o papel do WSLA na relação entre consumidor e provedor de serviços. Na ilustração, tanto o consumidor quanto o provedor possuem instrumentação e sistemas de medição e gerenciamento em seus ambientes. Os sistemas de medição e gerenciamento são alimentados pelo WSLA e cada entidade pode acessar métricas medidas em várias fontes, como as métricas do provedor ou do consumidor.

Um aspecto importante do WSLA é a sua capacidade de lidar com diversas especificidades de domínios e tecnologias diferentes. Essa capacidade só é possível devido à extensibilidade da linguagem XML, sobre a qual o WSLA é construído. A extensibilidade do WSLA permite a criação de tipos específicos de descrição de operações, além de novos tipos de diretiva de medição para sistemas específicos e de funções especiais para compor

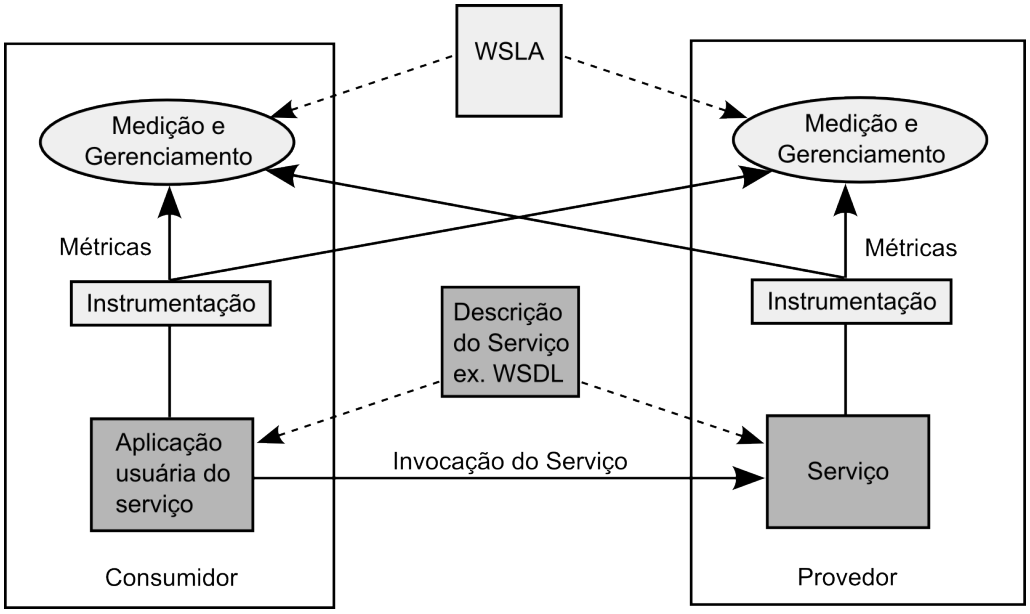


Figura 3.2: O papel do WSLA na relação consumidor-provedor de serviços. Fonte: (Keller e Ludwig, 2003)

métricas agregadas, por exemplo. Assim, o núcleo da especificação WSLA é pequeno, a fim de viabilizar o uso imediato para acordos simples, e, contudo, é bastante abrangente, devido a sua extensibilidade.

Além da simplicidade do núcleo e da extensibilidade, o projeto do WSLA também observou a necessidade de confidencialidade do consumidor do serviço, no sentido de que o acordo deve ser decomponível de forma que uma terceira parte envolvida não precise receber todo o acordo, mas apenas a informação de configuração que for necessária para sua participação.

## Estrutura e uso do WSLA

Para facilitar a configuração automática dos serviços e sistemas de supervisionamento, o WSLA é composto por três seções de informação:

- Partes Envolvidas: descrição das partes envolvidas, seus papéis e as interfaces de operações que elas expõem umas para as outras.
- Definições de Serviço: definição de propriedades ou parâmetros de nível de serviço e de métricas básicas ou compostas pelas quais os parâmetros são medidos.
- Obrigações: formaliza as obrigações de cada uma das partes envolvidas em relação a uma ou mais métricas dos parâmetros de nível de serviço. Essa formalização é descrita em objetivos de nível de serviço. Além disso, define as ações a serem tomadas em caso de violação das garantias de um serviço.

A Figura 3.3 apresenta essa divisão na estrutura de mais alto nível de um WSLA e a Figura 3.4 apresenta um exemplo de definição de um objetivo de nível de serviço. No exemplo da Figura 3.4, o provedor é definido como responsável pelo parâmetro *média de tempo de resposta* e deve mantê-lo abaixo de 5 segundos.

```
<?xml version="1.0">
<wsla:SLA
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:wsla="http://www.ibm.com/wsla"
     name="StockquoteServiceLevelAgreement12345" >
   <Parties>
      ...
   </Parties>
   <ServiceDefinition>
      ...
   </ServiceDefinition>
   <Obligations>
      ...
   </Obligations>
</wsla:SLA>
```

Figura 3.3: Exemplo da estrutura de alto nível de um WSLA.

```
<ServiceLevelObjective name="g1">
  <Obliged>provider</Obliged>
  <Validity>
     <Start>2001-11-30T14:00:00.000-05:00</Start>
     <End>2001-12-31T14:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Predicate xsi:type="wsla:Less">
      <SLAParameter>AverageResponseTime</SLAParameter>
      <Value>5</Value>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Figura 3.4: Exemplo de definição de objetivos de nível de serviço em um WSLA.

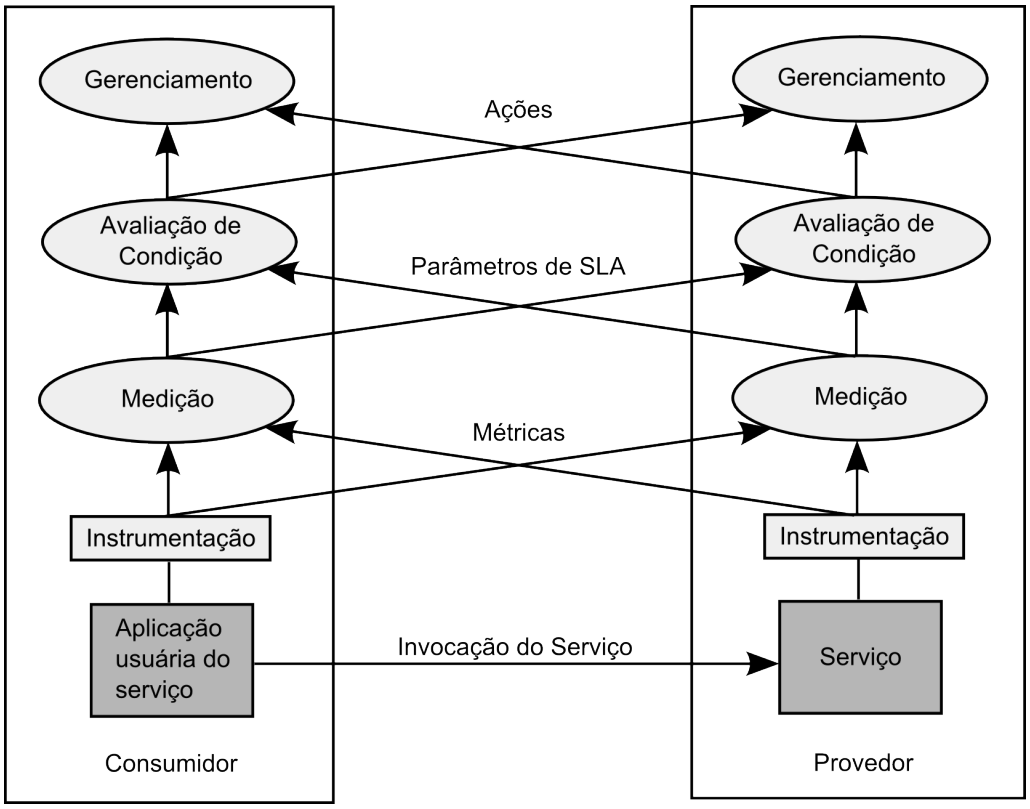


Figura 3.5: Interações em tempo de execução para gerenciamento da qualidade de serviço.

Em tempo de execução, tanto consumidor como provedor usam a instrumentação de seus serviços para capturar as métricas estabelecidas no WSLA. Essas métricas são passadas para o sistema de medição, que as mapeiam em parâmetros de nível de serviço e as repassam para uma avaliação de adequação às definições do WSLA. De acordo com a sua adequação ao WSLA, ações serão tomadas pelo sistema de gerenciamento, a fim de corrigir ou prevenir alguma inadequação. Dentre as ações que podem ser tomadas, podemos citar a reserva ou priorização de recursos bem como a cobrança de multas por violação de acordo. Essa interação é ilustrada na Figura 3.5.

Apesar da especificação do WSLA estar disponível para *download*, poucos estudos de caso foram relatados fora da IBM. Assim, o WSLA parece não ter sido amplamente adotado (Bianco et al., 2008). Por outro lado, o WS-Agreement do OGF tem crescido bastante nos últimos anos, com evoluções contínuas e diversos estudos de caso. O WS-Agreement, que incorporou e estendeu vários conceitos do WSLA, é apresentado na Seção 3.2.2.

### 3.2.2 WS-Agreement

O WS-Agreement é uma especificação desenvolvida pelo *Open Grid Forum* que define uma linguagem para formalização de acordos de nível de serviço. Assim como no WSLA, a linguagem é baseada em XML e definida através de *XML schemas*. Os *XML schemas* definem a estrutura geral do documento que representa o acordo. Além da linguagem de especificação, o WS-Agreement também define um protocolo para estabelecimento de acordos dinamicamente. Esse protocolo é baseado na tecnologia de serviços Web (Andrieux et al., 2007).

A especificação do WS-Agreement é composta por três partes: um esquema para especificar acordos, um esquema para especificar modelos de acordo e um conjunto de operações para gerenciar o ciclo de vida de um acordo. O modelo de acordo pode ser usado para facilitar a descoberta de provedores compatíveis com um determinado acordo. Quanto ao gerenciamento do ciclo de vida, são disponibilizadas operações para criar, invalidar e monitorar estados de acordos. É importante observar que cada uma dessas partes pode ser combinada com as demais, a fim de prover um esquema de qualidade de serviço personalizado e adaptado à situação (Bianco et al., 2008).

#### Estrutura do WS-Agreement

A Figura 3.6 apresenta uma visão geral da estrutura de um WS-Agreement. A estrutura é composta pelas seguintes partes:

- um ID único obrigatório para o acordo seguido de um nome opcional;
- contexto do acordo, o que inclui a identificação das partes envolvidas, vários metadados sobre o acordo (tempo de expiração, ID do modelo de acordo) e atributos definidos pelo usuário;
- os termos do acordo, que podem ser de serviço ou de garantia;
- termos de serviço que identificam e descrevem os serviços que estão sujeitos ao acordo. Os elementos para descrever um serviço dependem das necessidades do

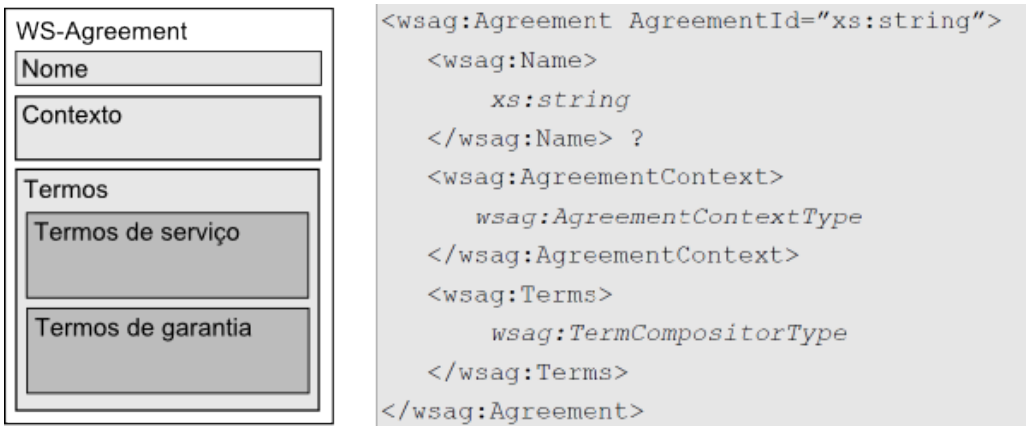


Figura 3.6: Estrutura geral de um acordo WS-Agreement. Fonte: (Andrieux et al., 2007)

domínio específico e, por isso, são personalizáveis. Termos de serviço incluem as propriedades mensuráveis dos serviços, que podem ser usadas para compor objetivos de nível de serviço. Nesse sentido, os termos de serviço são similares aos parâmetros de nível de serviço do WSLA; e

- termos de garantia que especificam os níveis de qualidade de serviço que as partes envolvidas acordam. Esses termos de garantia representam os objetivos de nível de serviço sobre os termos de serviço. Alguns exemplos de termos de garantia são a disponibilidade mínima de um serviço e o tempo máximo de resposta que o provedor deve fornecer. Em alguns casos, termos de garantia podem se aplicar ao consumidor, como por exemplo quando este deve fornecer alguma informação da qual o provedor depende para realizar o seu trabalho.

## Estrutura do modelo de WS-Agreement

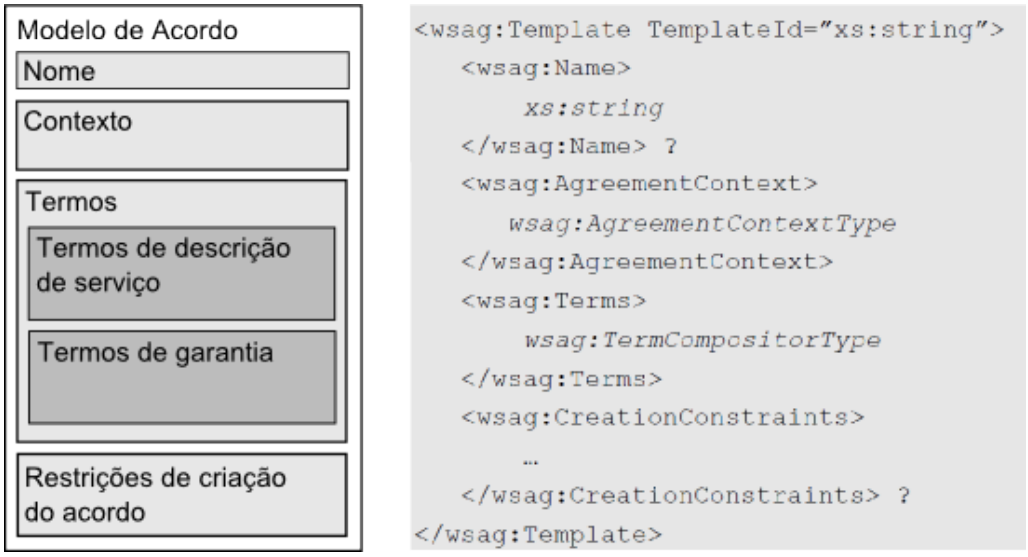


Figura 3.7: Estrutura geral de um modelo de acordo WS-Agreement. Fonte: (Andrieux et al., 2007)

Para criar um acordo, o consumidor primeiramente envia uma proposta de acordo ao provedor. Porém, para que o provedor interprete corretamente essa proposta de acordo, ele precisa divulgar previamente os modelos de acordo que ele espera receber de um consumidor. Dessa forma, os consumidores podem criar propostas de acordo compatíveis com os modelos de acordo interpretados por aquele provedor específico. É importante observar que a estrutura da proposta de acordo enviada ao provedor é a mesma do acordo final.

A Figura 3.7 apresenta uma visão geral da estrutura de um modelo de acordo WS-Agreement. A estrutura de um modelo de acordo é quase a mesma de um acordo, exceto pela presença da seção "Restrições de Criação do Acordo"ao final do modelo. Esta seção é opcional em um modelo de acordo e o seu objetivo é definir restrições para os possíveis valores de termos do acordo final. Em outras palavras, essas restrições possibilitam a definição de uma faixa de valores que os termos podem assumir.

É importante salientar que a definição de restrições de criação no modelo de acordo não garante que um provedor que atenda a essas restrições irá aceitar o acordo. Apesar do provedor poder definir um modelo de acordo apresentando as restrições de criação que ele espera aceitar, a decisão final pela aceitação ou rejeição de um acordo pode depender da disponibilidade dos seus recursos.

## Estados do WS-Agreement

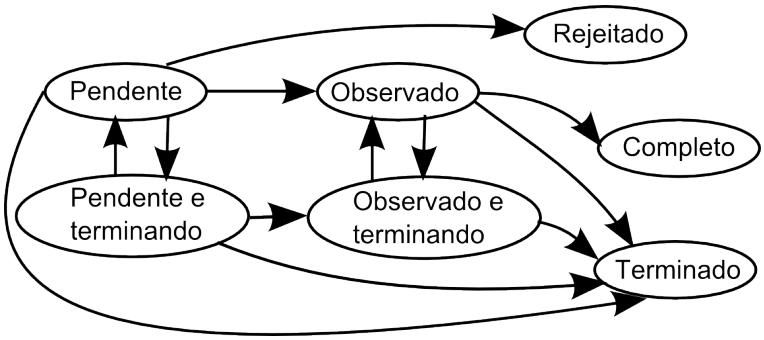


Figura 3.8: Estados do WS-Agreement em tempo de execução. Fonte: (Andrieux et al., 2007)

Acordos e termos possuem estados em tempo de execução que podem ser monitorados. O objetivo do monitoramento desses estados é observar a compatibilidade do nível de qualidade do serviço com o acordo em tempo de execução. A verificação do estado de um acordo requer uma infraestrutura significativa e que depende do ambiente e do domínio da aplicação. A Figura 3.8 apresenta os estados que um WS-Agreement pode assumir em tempo de execução. São eles:

- Pendente: o estado pendente significa que uma proposta de acordo foi feita mas que ainda não foi aceita ou rejeitada.
- Pendente e terminando: o estado pendente e terminando significa que uma proposta de acordo foi feita mas que ainda não foi aceita ou rejeitada e que uma operação de término foi lançada pelo proponente do acordo e está sendo processada.

- Observado: o estado observado significa que uma proposta de acordo foi feita e aceita.
- Observado e terminando: o estado observado e terminando significa que uma proposta de acordo foi feita, aceita e que uma operação de término foi lançada pelo proponente do acordo e está sendo processada.
- Rejeitado: o estado rejeitado significa que uma proposta de acordo foi feita e rejeitada.
- Completo: o estado completo significa que uma proposta de acordo foi recebida e aceita, e que todas as atividades pertencentes ao acordo finalizaram.
- Terminado: o estado terminado significa que uma proposta de acordo foi terminada pelo proponente do acordo e que a obrigação não mais existe.

É importante observar que o acordo entra no estado **completo** apenas quando todos os estados de serviço estão completos. Caso contrário, o estado do acordo é **observado**. Além disso, quando o acordo atinge o estado **completo** ou qualquer outro estado terminante, o contrato entre as partes é finalizado.

### 3.2.3 Considerações sobre acordos

Embora os acordos de serviço tenham se mostrado como uma boa opção de documento para especificar a qualidade de serviço acordada entre consumidor e provedor de recursos de grades computacionais, esses documentos não são suficientes para garantir a qualidade de serviço. É preciso também uma arquitetura de grade que suporte a inclusão de componentes de monitoração dos níveis de serviço, por exemplo. Além disso, outras questões, como escalonamento baseado em QoS e negociação de acordos de serviço, são essenciais para uma solução completa de grades computacionais com suporte a qualidade de serviço. Assim, muitas propostas diferentes de componentes e arquiteturas para grades computacionais têm sido criadas para prover QoS nessas plataformas. O propósito destas é preencher a lacuna que hoje limita o uso comercial desse tipo de plataforma de computação. As seções a seguir apresentam algumas dessas propostas.

## 3.3 Negociação de Acordos de Serviço

O processo de negociação de acordos de serviço em grades tem o propósito de promover um acordo sobre o nível de serviço a ser oferecido pelo provedor ao cliente, através de uma sequência de interações entre as partes (Venugopal et al., 2008). Normalmente, este processo compreende a apresentação de uma proposta de acordo de serviço pelo cliente ao provedor, seguida de uma análise desta no provedor e do envio de uma resposta de aceitação, negação ou uma contra-proposta ao cliente. Pressupõe-se, então, a existência de um padrão de definição de documento de acordo de serviço a ser utilizado por ambas as partes (cliente e provedor). Além disso, o processo de negociação estabelece um protocolo de comunicação entre o provedor e o cliente, a fim de atingir o objetivo, que é o estabelecimento do acordo de serviço aceito por ambos. Do ponto de vista de interação humana, esse processo pode ser completamente automático, humano ou uma composição

das duas abordagens (híbrido). As abordagens mais interessantes são as completamente automáticas e as híbridas, visto que apresentam um ganho de escalabilidade e desempenho em relação às abordagens puramente humanas.

(Venugopal et al., 2008) apresentam um protocolo bilateral para negociação de acordo de serviço baseado no mecanismo de ofertas alternadas de Rubinstein (Rubinstein, 1982). Esse mecanismo possibilita que o provedor ou o cliente crie uma contra-proposta baseada na proposta original, apenas modificando esta em alguns termos. O processo de negociação, neste caso, é completamente automático, ou seja, não há interação humana para a geração de contra-propostas durante a execução do protocolo. Apesar de a proposta ser interessante, o protocolo suporta apenas a negociação de intervalos de tempo e de número de recursos, ou seja, ainda não leva em consideração outros atributos, como recompensa e penalidades em caso de violação. Além disso, o protocolo depende de estimativas fornecidas pelo cliente a respeito do tempo de execução da aplicação, para computar o número de recursos requeridos.

(Vahidov e Neumann, 2008) apresentam uma abordagem baseada em agentes e em sistemas de apoio à decisão. Essa proposta combina efetivamente o julgamento humano com a tomada de decisão realizada por agentes automáticos de gerenciamento durante o processo de negociação de acordos de serviço. Trata-se, portanto, de uma abordagem híbrida com relação à interação humana. O objetivo da proposta é, a partir da intervenção humana, aumentar a previsibilidade dos resultados do negócio do cliente, principalmente em situações em que o comportamento deste depende muito de eventos econômicos e tecnológicos recentes, por exemplo. A proposta foi simulada e apresentou resultados interessantes, como o controle do número de acordos de serviço fechados com clientes em um período de tempo e um aumento considerável de lucro quando comparado a uma abordagem de preço fixo (sem intervenção).

(Czajkowski et al., 2002) propõem o uso do protocolo SNAP (*Service Negotiation and Acquisition Protocol*) como forma de negociação de SLAs e coordenação da gestão de recursos em sistemas distribuídos. Os autores apresentam um modelo para gerenciar o processo de negociação de acesso e o uso de recursos através da definição de um *framework* dentro do qual reserva, aquisição e submissão de tarefas podem ser expressas para qualquer recurso de uma maneira uniforme.

Por fim, durante o processo de negociação, o provedor de serviços precisa identificar se ele pode aceitar um novo SLA baseado nos compromissos acordados para seus recursos e na probabilidade de completar a tarefa no tempo especificado. Assim, a negociação de SLAs tem influência direta sobre o gerenciamento dos recursos da grade. (Menascé e Casalicchio, 2004) discutem e exemplificam a influência da negociação de SLAs na política de escalonamento de recursos em grades computacionais, a fim de atender aos requisitos do SLA e manter as restrições de custo em um nível aceitável para o provedor de serviços.

## 3.4 Escalonamento de Recursos em Grade Baseado em Acordos de Serviços

O processo de escalonamento de recursos em grades consiste no mapeamento de tarefas a recursos de grade, tais como *clusters* computacionais, supercomputadores paralelos, máquinas *desktop*, que pertencem a domínios administrativos diferentes e obedecem a po-

líticas de uso diferentes (Yu et al., 2005). Um mecanismo de escalonamento de recursos em grades baseado em acordos de serviço deve utilizar os requisitos do acordo de serviço (tempo de execução, ambiente de execução, confiabilidade, vazão...) para realizar o mapeamento das tarefas aos recursos, de forma a garantir o atendimento desses requisitos. Além disso, quando se trata de um grade comercial, onde o cliente paga pelo uso dos recursos da grade, é essencial que o mecanismo de escalonamento garanta o atendimento dos requisitos do acordo de serviço com o menor custo possível. Pressupõe-se, então, a definição de um preço para cada recurso disponibilizado no grade.

(Yu et al., 2005) apresentam um mecanismo de escalonamento de recursos em grades baseado em requisitos de QoS para aplicações de *workflow*. Uma aplicação de *workflow* é uma aplicação que requer o processamento de um fluxo de tarefas, as quais são executadas com base em suas dependências temporais de controle e de dados. A proposta de escalonamento de recursos apresentada para este tipo de aplicação define um algoritmo que minimiza o custo de execução sem deixar de atender às restrições de tempo fornecidas pelo cliente no acordo de serviço. Nesse caso, é necessário, portanto, que o cliente defina as restrições de tempo para cada tarefa ou para a aplicação de *workflow* como um todo.

O processo de escalonamento de *workflow* proposto envolve três etapas: planejamento, reserva avançada de recursos e execução com suporte a reescalonamento. Na primeira etapa, cada tarefa é mapeada a um recurso com base nas suas restrições de tempo e na capacidade de QoS do recurso. Na segunda etapa, os recursos mapeados são reservados para garantir o QoS mesmo em um tempo futuro, baseado no *workflow* da aplicação. Por fim, a aplicação começará a executar e, em caso de falha de algum recurso em prover o QoS acordado, o mecanismo de reescalonamento é acionado para redistribuir as tarefas, a fim de não impactar no nível de serviço global da aplicação.

Modelando aplicações de *workflow* como grafos acíclicos dirigidos e utilizando o processo de decisão de Markov (Sutton e Barto, 1998) para escalonar tarefas sequenciais, a proposta obteve resultados muito bons quando comparada com outros três algoritmos de escalonamento. Contudo, num modelo econômico de grades, é muito provável que os preços dos recursos variem com base na lei da oferta e da procura, o que não é abordado nesta proposta.

(Zhu e Agrawal, 2009), por sua vez, propõem uma solução de escalonamento de recursos para aplicações adaptáveis, que são aplicações compostas por serviços que podem ser configurados a cada execução, e que possuem requisitos de prazo para execução. A proposta do artigo pressupõe a existência de uma função de benefício fornecida pelo cliente para a aplicação. O objetivo da estratégia de escalonamento proposta é maximizar essa função de benefício sem comprometer o prazo de execução requerido. Os recursos analisados pelo artigo compreendem CPU, memória e banda de comunicação.

Nessa proposta, o processo de escolha de um nodo para execução de um dado serviço se baseia no conceito de valor de eficiência, definido como um valor que representa quão efetivamente um serviço específico pode ser executado em um nodo particular, levando em conta a adaptação dos parâmetros de serviço e o tempo de execução. O valor de eficiência é inferido durante uma fase de treinamento de regras *fuzzy*. Posteriormente a essa fase, é executado um algoritmo guloso que prioriza os serviços de acordo com o seu impacto na função de benefício e escolhe os recursos de acordo com os valores de eficiência calculados.

Em comparação com o algoritmo ótimo, o benefício médio normalizado da proposta em três cenários (de muito homogêneo a muito heterogêneo) para as aplicações atingiu

87% e a taxa de sucesso foi de 90%. Contudo, a necessidade de se definir, previamente, uma função de benefício para cada aplicação é um limitador muito forte para a adoção dessa proposta em grades comerciais.

(Burchard et al., 2005) apresentam o VRM (*Virtual Resource Manager*), que é uma extensão para adicionar gestão de QoS baseada em SLAs aos sistemas de gerenciamento de recursos (RMS) utilizados nos grades atuais. Tal artigo defende que o SLA deve ser atendido em qualquer momento, e não apenas durante o período em que os recursos estão reservados. Além disso, devem existir mecanismos para garantir o SLA mesmo em caso de falhas em recursos (reescalonamento, por exemplo).

(Ching et al., 2003) afirmam, porém, que um dos maiores problemas na gestão de SLAs em ambientes de grade é que os requisitos definidos em um SLA por um usuário para uma aplicação (tempo de término, aplicação, qualidade de serviço) não têm um mapeamento direto para as características de baixo nível dos recursos requeridos por sistemas de gestão de grades (número de processadores, capacidade de CPU). O mesmo artigo apresenta um processo de gestão de SLA para o projeto SOGRM (*Self Organised Grid Resource Management*) e uma modelagem de recursos que relaciona os requisitos de SLA com as características de baixo nível dos recursos da grade, possibilitando a quantificação do tempo e facilitando o processo de avaliação de novos SLAs.

Mesmo com uma política muito boa de escalonamento e descoberta de recursos no grade, a qualidade de serviço ainda pode sofrer variações durante a execução dos serviços. Essas variações ocorrem por diversos motivos, como falhas dos recursos, sobrecarga de usuários, entre outros. Assim, torna-se essencial a conexão do sistema de gerenciamento e escalonamento de recursos a um componente de monitoramento dos níveis de serviço, de forma que se possa identificar as falhas e prevenir violações de contrato de QoS.

## 3.5 Monitoramento de Acordos de Serviços em Grades

O processo de monitoramento de acordos de serviço consiste em avaliar periodicamente se o nível de serviço oferecido pelo provedor em determinado instante está compatível com o acordo de serviço firmado com o cliente (Raimondi et al., 2008). Em caso de incompatibilidade, constata-se uma violação do acordo e, normalmente, uma penalidade é aplicada ao provedor, em favor do cliente. Pressupõe-se, então, que as penalidades relativas a cada tipo de violação estejam descritas no próprio acordo de serviço firmado ao fim do processo de negociação. Duas questões muito importantes nesse tema são o *overhead* do próprio sistema de monitoramento, que influi no resultado, e a precisão e confiabilidade desse monitoramento, já que, normalmente, não há uma relação de confiança mútua entre provedor e consumidor de serviços em grade. Além disso, o monitoramento pode ser *offline* ou *online* (Raimondi et al., 2008). No monitoramento *offline*, os dados referentes ao nível de serviço são coletados e armazenados para posterior análise de violação. Já no monitoramento *online*, os dados são coletados e analisados durante a execução do serviço, gerando alertas de violação quase que instantaneamente.

(Scorsatto e Melo, 2008) apresentam um serviço de monitoramento de acordos de serviço do tipo WS-Agreement (Andrieux et al., 2007) baseado no *middleware* Globus Toolkit 4. A proposta faz o monitoramento periódico do uso de recursos usando a técnica

de amostragem aleatória estratificada, gerando ações personalizadas em caso de violação do acordo (monitoramento *online*). A definição das ações personalizadas a serem tomadas em caso de violação de um acordo é realizada previamente durante a construção deste. Além disso, uma de suas principais características é o baixo *overhead* da solução (abaixo de 3% do tempo de execução no ambiente de testes utilizado). Contudo, a proposta não trata da questão da confiabilidade da informação, assumindo que há uma relação de confiança mútua entre provedor e cliente.

(Raimondi et al., 2008) apresentam um método de monitoramento *online* de acordos de serviço baseado em autômatos temporais. A idéia básica é transformar requisitos do acordo de serviço, como latência (tempo entre resposta e requisição), confiabilidade (número de erros em um intervalo de tempo) e vazão (número de requisições em um intervalo de tempo), em autômatos temporais cujas linguagens aceitas caracterizam exatamente as violações das especificações. A Figura 3.9 apresenta (a) um autômato temporal para detecção de violações de latência, que chega ao estado final quando a resposta para uma requisição é enviada em um tempo maior do que $t$, (b) um autômato temporal para detecção de violações de confiabilidade, que chega ao estado final quando três falhas ocorrem num intervalo de tempo menor do $t$ e (c) um autômato temporal para detecção de violações de vazão, que chega ao estado final quando o cliente envia três requisições em um intervalo de tempo menor do que $t$. Prova-se que, para esses tipos de requisitos, o monitoramento por autômatos temporais tem complexidade quadrática no número de estados do autômato. Além disso, o método proposto é não-invasivo, ou seja, não necessita de reescrita dos serviços existentes para inserção de instrumentação, e sua implantação não requer conhecimento da aplicação a ser monitorada. Contudo, a proposta só trata de requisitos temporais, mas um acordo de serviço pode também impor requisitos não-temporais.

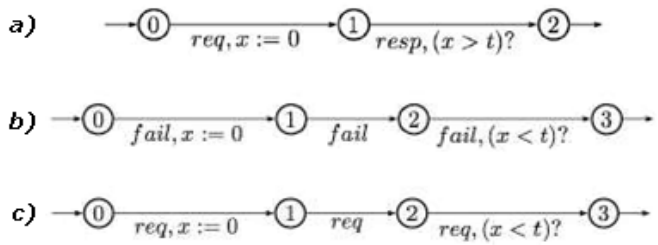


Figura 3.9: a) Autômato temporal para violações de latência. b) Autômato temporal para violações de confiabilidade. c) Autômato temporal para violações de vazão. Fonte: (Raimondi et al., 2008)

O problema dessas abordagens é que nenhuma delas trata de um problema comum em grades computacionais que é a falta de relação de confiança entre provedor e consumidor, já que estes podem estar em domínios administrativos diferentes. Esse problema leva à questão de quem será responsável por auditar a qualidade do serviço fornecido de forma que não existam conflitos de interesse.

O problema da auditoria de qualidade de serviço é o foco deste trabalho. O capítulo 4 apresenta e discute esse problema no contexto do provimento de qualidade de serviço em grades computacionais.

# Capítulo 4

# Auditoria de Qualidade de Serviço em Grades

## 4.1 Visão Geral

O International Telecommunication Union (ITU-T) e o Internet Engineering Task Force (IETF) apresentam definições de auditoria, porém ambas com foco em auditoria na área de segurança (Shirey, 2000) (ITU-T, 2003). Uma definição mais genérica de auditoria é apresentada em (Hasan e Stiller, 2005): "uma auditoria é um exame sistemático e independente de fatos em atividades de sistema para determinar o grau de conformidade com relação a um conjunto de especificações".

Em outras palavras, uma auditoria pode ser definida como uma avaliação sistemática de conformidade realizada por uma entidade independente. A avaliação de conformidade normalmente é realizada no escopo de um processo, de um projeto ou de um produto, e é baseada em alguma especificação de como o objeto avaliado deve funcionar. Assim, uma auditoria sobre o processo de desenvolvimento de um software é uma avaliação, realizada por uma entidade independente, sobre a conformidade do processo executado em relação à especificação original do processo.

Diferentemente do monitoramento da qualidade de serviço, que normalmente é executada pelo provedor do serviço a fim de evitar uma violação do SLA, uma auditoria de qualidade de serviço avalia a conformidade da qualidade de serviço prestada em relação aos termos do SLA de forma isenta, ou seja, por uma entidade externa independente e de confiança tanto do provedor como do consumidor - o auditor.

À medida em que o provimento de serviços em grades aumenta, mais e mais entidades participantes se relacionam por meio de contratos de prestação de serviço e, consequentemente, de SLAs para esses serviços. Uma vez que muitas dessas entidades participantes da grade não mantêm nenhuma relação de confiança entre si, podem surgir disputas em relação à aferição da qualidade do serviço prestado. Por exemplo, se um consumidor detecta uma violação do SLA durante a prestação de um determinado serviço, o provedor pode contestar a violação apresentando os resultados de sua aferição. Justifica-se, então, a necessidade de se estabelecer uma entidade independente para arbitrar esses conflitos de interesse.

Conforme apresentamos anteriormente, existem diversos trabalhos relacionados com o gerenciamento de SLAs em grades que abordam desde a negociação até o monitoramento

de acordos de nível de serviço. Porém, embora alguns deles mencionem a possibilidade de se estabelecer uma entidade externa para avaliar a qualidade do serviço, poucos trabalhos discutem em detalhes como realizar uma auditoria de qualidade de serviço em ambientes que envolvem domínios administrativos diferentes e plataformas tão heterogêneas como as grades computacionais.

(Barbosa et al., 2006) identificam essa lacuna e apresentam possíveis arquiteturas para introduzir um auditor de qualidade de serviço como uma entidade de confiança na relação entre consumidor e provedor de serviço em grades. A Seção 4.2 apresenta e discute vantagens e desvantagens dessas arquiteturas.

## 4.2 Arquiteturas para Auditoria de Qualidade de Serviço

(Barbosa et al., 2006) analisam 6 arquiteturas para a implementação de auditoria de qualidade de serviço em grades computacionais. A análise qualitativa avalia cada uma das 6 arquiteturas com relação à (i) necessidade de relação de confiança entre consumidor e provedor, (ii) intrusividade da solução, (iii) necessidade de requisições adicionais, (iv) possibilidade de tratamento preferencial para certas requisições, (v) possibilidade de controle da taxa de requisição do consumidor e (vi) se a arquitetura pode ser utilizada quando as mensagens trocadas entre consumidor e provedor são encriptadas. A Tabela 4.1 apresenta o resultado dessa análise entre as arquiteturas propostas.

Tabela 4.1: Análise qualitativa de arquiteturas de auditoria de QoS. Fonte: (Barbosa et al., 2006)

| | Confiança | Intrusividade | Requisição adicional | Tratamento preferencial | Carga consumidor | Mensagem encriptada |
|---|---|---|---|---|---|---|
| 1. Ingênua (*naive*) | Sim | Código | Não | Não | Sim | Sim |
| 2. Farejador de pacotes (*sniffer*) | Não | Hardware | Não | Não | Sim | Não |
| 3. Decorador de *host* | Não | Hardware | Não | Não | Sim | Sim |
| 4. Inspetor independente | Não | Não | Sim | Sim | Uso de tickets | Sim |
| 5. Decorador externo | Não | Não | Não | Não | Sim | Sim |
| 6. Decorador externo com *bypass* | Não | Não | Não | Sim | Uso de tickets | Sim |

No caso da arquitetura 1, ilustrada na Figura 4.1, existem 2 problemas. O primeiro é o da necessidade de relação de confiança entre consumidor e provedor, já que, neste caso, são o consumidor e o provedor que monitoram e fornecem os valores de QoS ao auditor para avaliação de compatibilidade com o acordo. E o segundo problema é o da intrusividade da solução. Uma vez que consumidor e provedor precisam monitorar a qualidade de serviço

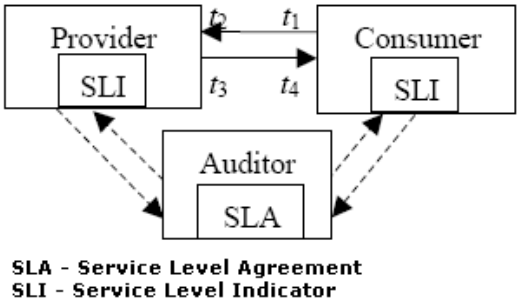


Figura 4.1: Arquitetura 1 (Ingênua). Fonte: (Barbosa et al., 2006)

e fornecer as informações ao auditor, eles precisam incluir código de monitoração em seus serviços originais.

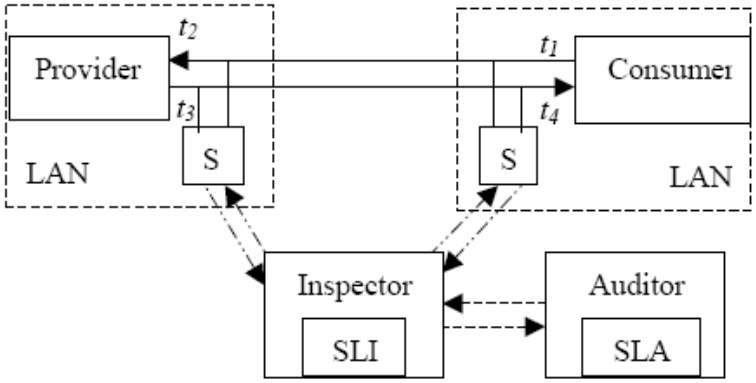


Figura 4.2: Arquitetura 2 (Farejador de Pacotes). Fonte: (Barbosa et al., 2006)

No caso das arquiteturas 2 e 3, Figuras 4.2 e 4.3, também existe o problema de intrusividade, mas desta vez a intrusividade é tanto de código quanto de *hardware*. Como ambas as arquiteturas contam com a instalação de código externo (*sniffer* ou inspetor) tanto no provedor quanto no consumidor para obter os dados de QoS necessários, é preciso uma solução em *hardware*, como os coprocessadores seguros (Smith e Weingart, 1999), para tornar esse código resistente a alterações maliciosas nesses ambientes. Além disso, a arquitetura 2, baseada em *sniffers* de pacotes de rede, não consegue recuperar informações de QoS quando as mensagens trocadas são encriptadas. Essa limitação é amenizada na arquitetura 3, uma vez que os inspetores trabalham em nível de serviço e não de pacotes.

As arquiteturas 4 e 6, Figuras 4.4 e 4.5, também apresentam problemas. Na arquitetura 4, é necessário realizar requisições adicionais ao provedor, a fim de calcular os valores de QoS. Essas requisições adicionais provocam aumento no tráfego de rede e podem sobrecarregar o provedor. Outro problema, encontrado em ambas as arquiteturas 4 e 6, é que o provedor pode ser capaz de identificar as requisições do inspetor e dar-lhes tratamento preferencial, fingindo uma qualidade de serviço superior àquela realmente fornecida ao consumidor nas requisições normais. Nessas arquiteturas, a taxa de requisições do consumidor ao provedor (carga do consumidor) pode ser regulada através do uso de *tickets* digitalmente assinados pelo auditor e entregues ao consumidor sob demanda. O consu-

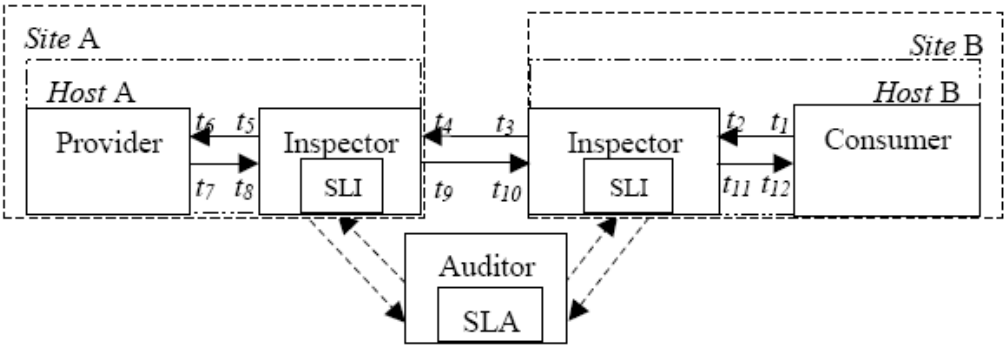


Figura 4.3: Arquitetura 3 (Decorador de *Host*). Fonte: (Barbosa et al., 2006)

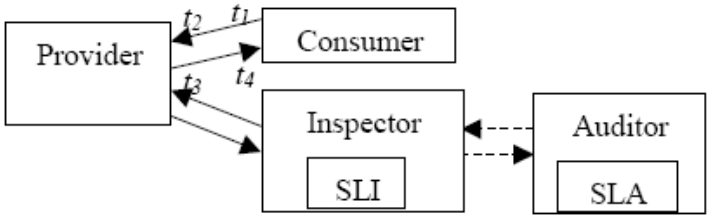


Figura 4.4: Arquitetura 4 (Inspetor Independente). Fonte: (Barbosa et al., 2006)

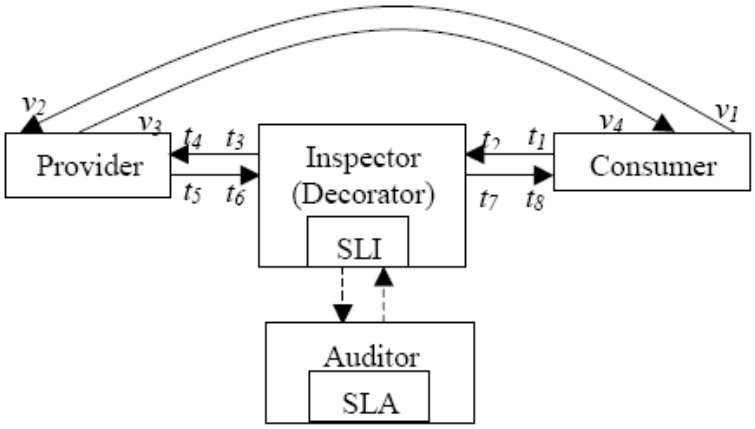


Figura 4.5: Arquitetura 6 (Decorador Externo com *Bypass*). Fonte: (Barbosa et al., 2006)

midor só consegue enviar uma requisição ao provedor com um *ticket* assinado e, assim, garante-se a taxa de requisições acordada no SLA.

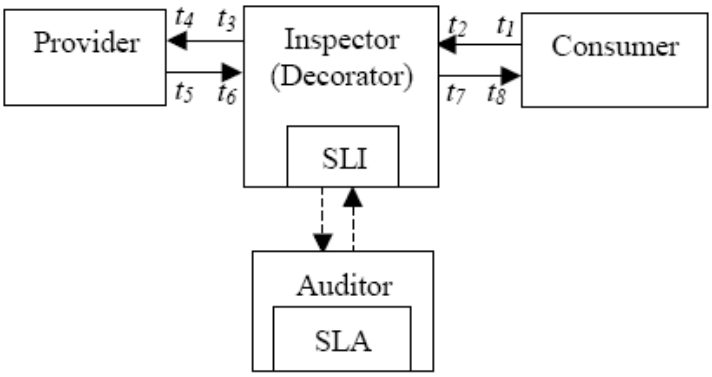


Figura 4.6: Arquitetura 5 (Decorador Externo). Fonte: (Barbosa et al., 2006)

Por fim, de acordo com a análise qualitativa, a arquitetura 5, Figura 4.6, foi a que se mostrou mais adequada ao cenário de utilização previsto. Contudo, a utilização de um auditor externo por onde todas as requisições do consumidor ao provedor devem passar precisa de algumas precauções. Uma delas é a proximidade do auditor em relação ao provedor e ao consumidor, a fim de não gerar um atraso considerável na comunicação e deteriorar o desempenho geral do sistema. Portanto, é preciso uma quantidade de auditores suficiente e uma boa distribuição geográfica destes para que as aplicações não sofram um forte impacto pelo uso de um intermediário na comunicação entre provedor e consumidor.

## 4.3 Trabalhos Relacionados com Auditoria de Qualidade de Serviço

(Shah et al., 2007) afirmam que, com o aumento do número de provedores de serviços online, como os serviços de e-mail, fotos e armazenamento digital, fica cada vez mais evidente a necessidade de se prover garantias de qualidade de serviço aos usuários desses serviços. Em especial, os autores discutem a necessidade dos usuários de serviços de armazenamento digital em obter informações confiáveis sobre os riscos de perda de arquivos nesses ambientes. A disponibilização dessas informações é essencial para o julgamento e a escolha entre várias opções de provedores de armazenamento. O estabelecimento de um auditor externo e de confiança de ambas as partes é proposto, a fim de realizar auditorias nos ambientes dos provedores e informar usuários sobre a capacidade desses provedores em atender a determinados SLAs relacionados com a integridade dos dados. Os provedores seriam responsáveis, através de incentivos ou penalidades, por disponibilizar interfaces específicas para a realização de auditoria interna e externa pelos auditores, de forma automatizada. Os autores apresentam características das auditorias e dessas interfaces específicas, porém não discutem em detalhes como o auditor se insere na relação consumidor-provedor nem apresentam uma implementação baseada em algum tipo de acordo de nível de serviço.

(Skene et al., 2007) argumentam que um SLA de nada serve se as suas restrições não puderem ser monitoradas pelas partes envolvidas. Assim, os autores propõem uma técnica para determinar o grau de monitorabilidade das restrições de um SLA. A monitorabilidade de um SLA é classificada em: não-monitorável, monitorável por uma parte, monitorável por ambas as partes ou arbitrável. A monitorabilidade de um SLA é arbitrável quando uma terceira entidade (um auditor, por exemplo) pode monitorar as suas restrições. Através de uma formalização matemática, a técnica apresentada determina o grau de monitorabilidade de um SLA, a fim de que se possa evitar restrições que não possam ser monitoradas ou que só possam ser monitoradas por uma das partes. Para o caso de SLAs que, no máximo, podem ser monitorados por ambas as partes, os autores também apresentam um teste estatístico de hipótese que pode ser usado para validar as informações fornecidas por uma das partes. Para SLAs arbitráveis, o artigo não discute como o auditor deve ser escolhido pelas partes. A implementação da formalização matemática, porém, se baseia em SLAs especificados com a linguagem SLAng (Lamanna et al., 2003). A linguagem SLAng é concorrente do WS-Agreement e do WSLA no que diz respeito à especificação de acordos de nível de serviço.

(Hasan e Stiller, 2005) apresentam um modelo genérico para implementar um mecanismo de auditoria de SLA. O funcionamento do modelo é demonstrado através de um cenário de exemplo em que um operador de rede oferece serviços de rede com suporte a QoS para seus clientes. Assim como o que ocorre na arquitetura 2 (Farejador de Pacotes) apresentada na Seção 4.2, a monitoração dos níveis de serviço é realizada nos ambientes do provedor e do consumidor no modelo proposto. Os autores reconhecem a possibilidade de alteração maliciosa do componente de medição nos ambientes do provedor e do consumidor e apenas comentam sobre a possibilidade de se utilizar uma entidade externa para a medição e contabilização dos parâmetros acordados no SLA. Também neste artigo, a forma de introdução dessa terceira entidade na relação provedor-consumidor não é discutida, bem como não é apresentada uma implementação que utilize algum tipo de acordo de nível de serviço.

Em (Racz e Stiller, 2009), os autores apresentam outra solução de auditoria para monitorar o cumprimento de SLA acordado entre um provedor de serviço de *streaming* de vídeo (consumidor) e o provedor de rede que hospeda esse serviço (provedor). A solução apresentada é construída sobre o *Auditing Framework for Internet Services* (Hasan e Stiller, 2007) e utiliza a arquitetura 2 para monitorar a largura de banda disponibilizada ao provedor do serviço de *streaming* de vídeo pelo provedor de rede. Assim, a proposta também apresenta os problemas da intrusividade e da necessidade de proteger os componentes de medição contra alterações maliciosas nos ambientes do provedor e do consumidor. Esses problemas e a negociação do auditor externo não são discutidos no artigo.

(Eyermann e Stiller, 2007) discutem a complexidade de auditar a qualidade de serviço na camada de transporte de redes IP quando vários domínios administrativos são envolvidos no provimento de um único serviço. Nesse caso, além de identificar violações dos objetivos do acordo de nível de serviço, é preciso também identificar em que domínio administrativo a violação ocorreu. Segundo os autores, esse requisito aumenta significativamente a complexidade e o overhead da solução de monitoração do nível de serviço. Nesse trabalho é apresentado o protocolo MeSA (*Measured Signaling for Auditing*), que busca minimizar o overhead da monitoração sem deixar de considerar o envolvimento de vários domínios administrativos. Para isso, a solução conta com a presença de roteadores

de borda com suporte ao protocolo MeSA implantados em cada domínio administrativo. Esses roteadores são capazes de tratar os cabeçalhos de pacotes IP extras enviados apenas para mensurar alguns parâmetros de desempenho da camada de transporte, como atraso e jitter, por exemplo. A solução é parecida com a da arquitetura 4 (inspetor independente) apresentada na Seção 4.2, uma vez que esta também envia requisições extras para avaliar a qualidade do serviço prestado. Porém, a expressão auditoria de qualidade de serviço nesse trabalho é usada como sinônimo de monitoração de qualidade de serviço, uma vez que não existe uma entidade independente que valida as informações de qualidade de serviço obtidas.

A Tabela 4.2 apresenta um resumo comparativo entre as características dos conteúdos dos artigos apresentados nas Seções 4.2 e 4.3.

Tabela 4.2: Análise comparativa de sistemas de auditoria de QoS.

| Artigo | Tipo de Acordo | Objeto da Negociação | Auditoria | Negocia Auditor |
|---|---|---|---|---|
| 1. (Barbosa et al., 2006) | N/A | tempo de processamento | Sim | Não |
| 2. (Shah et al., 2007) | N/A | integridade de dados armazenados | Sim | Não |
| 3. (Skene et al., 2007) | SLAng | tempo de processamento | Sim | Não |
| 4. (Hasan e Stiller, 2005) | N/A | vazão | Sim | Não |
| 5. (Racz e Stiller, 2009) | N/A | vazão | Sim | Não |
| 6. (Hasan e Stiller, 2007) | N/A | falhas do sistema, requisições, vazão | Sim | Não |
| 7. (Eyermann e Stiller, 2007) | N/A | atraso, jitter | Não | Não |

Como pode ser observado na Tabela 4.2, apenas o artigo (Skene et al., 2007) apresenta um tipo de acordo específico no desenvolvimento do trabalho - o SLAng. Os demais artigos não citam o tipo de acordo, seja porque são modelos genéricos e, portanto, independentes de linguagens de especificação de acordo, ou porque o trabalho é desenvolvido sobre o resultado de um *parser* previamente aplicado ao acordo, ou seja, o foco do trabalho está em uma etapa posterior à negociação e ao processamento do conteúdo do acordo estabelecido.

A segunda coluna da Tabela 4.2 apresenta os objetos que fazem parte dos acordos de nível de serviço negociados em cada um dos trabalhos. Esses objetos da negociação variam bastante, desde os mais simples de monitorar, como quantidade de requisições realizadas, até os mais complexos, como a integridade de dados armazenados. Para cada um desses objetos, são definidos objetivos de nível de serviço nos respectivos trabalhos.

Devido à variabilidade de interpretações para o termo auditoria, a quarta coluna da Tabela 4.2 verifica se o artigo trata do assunto auditoria tal qual definido na Seção 4.1. Conforme pode ser observado, apenas o artigo (Eyermann e Stiller, 2007) não usa a definição de auditoria que pressupõe a existência de uma entidade externa independente. Ou seja, apesar de o artigo usar o termo auditoria, como não existe entidade independente no desenvolvimento do trabalho, na realidade trata-se apenas de uma monitoração de acordo de nível de serviço.

A última coluna da Tabela 4.2 identifica os artigos que efetivamente negociam o auditor independente. No caso dos artigos analisados e do melhor conhecimento dos autores, nenhum trabalho está tratando diretamente da questão da negociação do auditor independente. Assim, as seções seguintes deste trabalho discutem e validam uma proposta de mecanismo para negociar auditores externos em ambientes de grades computacionais com suporte a qualidade de serviço.

# Capítulo 5

# Mecanismo de Negociação de Auditor Externo

## 5.1 Considerações Iniciais

Conforme apresentado e discutido na Seção 4.2, existem algumas propostas de arquitetura para tratar da questão de auditoria de qualidade de serviços em grades computacionais. Um estudo que compara essas propostas quantitativa e qualitativamente, apresentado na Seção 4.2, mostra que a Arquitetura 5 em que o auditor se coloca como intermediário obrigatório em cada troca de mensagens entre consumidor e provedor de serviços, apresenta as principais características para viabilizar a auditoria de qualidade de serviço em grades computacionais. A única ressalva apresentada no estudo é que o auditor deve estar conectado ao consumidor e ao provedor de forma a reduzir o atraso de comunicação entre eles, ou seja, a introdução do auditor deve causar o mínimo de interferência na comunicação.

A Figura 5.1 ilustra dois tipos de interação entre consumidor e provedor de serviços. A Figura 5.1.a apresenta uma interação sem a presença do auditor de qualidade de serviços e a Figura 5.1.b apresenta uma interação conforme a Arquitetura 5 (Seção 4.2), em que o auditor de qualidade de serviços se apresenta como intermediário da comunicação entre consumidor e provedor de serviços.

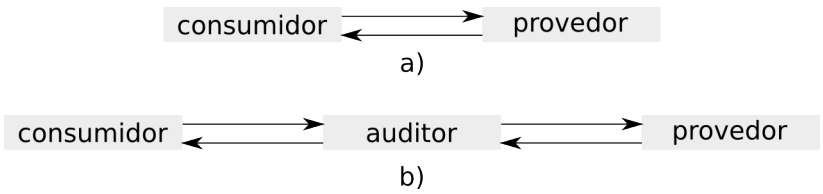


Figura 5.1: a) Interação entre provedor e consumidor sem auditoria. b) Interação entre provedor e consumidor por intermédio de um auditor.

Para introduzir o auditor como intermediário na comunicação entre consumidor e provedor, é interessante que o auditor funcione como um decorador da interface fornecida pelo provedor. De acordo com esse padrão de projeto, discutido em (Gamma et al., 1995), um decorador possui a mesma interface de uma determinada implementação, estendendo ou modificando o seu comportamento. No caso da auditoria de qualidade de serviço, o auditor é o decorador do provedor de serviço e seu comportamento deve ser o de registrar

informações relativas à auditoria do acordo de nível de serviço e repassar toda comunicação do consumidor ao provedor e vice-versa.

Uma interação entre consumidor e provedor de serviços, sem auditoria de qualidade de serviços e seguindo o padrão WS-Agreement (Seção 3.2.2), ocorre conforme a sequência de mensagens apresentada na Figura 5.2. O consumidor inicia a interação solicitando os modelos de acordo de nível de serviço aceitos pelo provedor (1). Então, com base em um dos modelos obtidos, o consumidor cria uma oferta de acordo de nível de serviço para execução de uma tarefa (2). Por fim, o consumidor envia essa oferta ao provedor, solicitando a execução do acordo. O provedor, por sua vez, tem a possibilidade de negar ou aceitar o acordo (3). Caso o acordo seja aceito, o provedor enviará a tarefa para o gerenciador de recursos (4) que, por sua vez, a enviará para execução em um dos recursos da grade computacional.

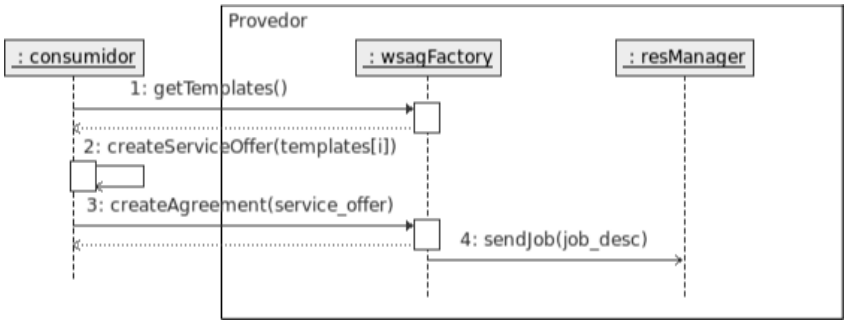


Figura 5.2: Interação entre consumidor e provedor de serviços sem auditoria.

A Seção 5.3 apresenta o mecanismo de negociação de auditoria, que é uma extensão à sequência de mensagens da Figura 5.2 e cujo propósito é negociar e estabelecer um auditor como intermediário entre consumidor e provedor. A Seção 5.2 apresenta premissas e restrições para o projeto do mecanismo de negociação de auditoria proposto.

## 5.2 Premissas e Restrições do Mecanismo

A fim de lidar com auditoria conforme a Arquitetura 5 (Seção 4.2), é preciso levar em consideração que o auditor deve ser uma entidade de confiança tanto do provedor como do consumidor. Assim, assume-se que tanto consumidor como provedor possuem uma lista de auditores confiáveis e que, normalmente, o auditor selecionado no processo de negociação estará presente tanto na lista de auditores confiáveis do provedor como do consumidor. Caso o auditor selecionado seja de confiança apenas do provedor, por exemplo, o consumidor não terá a quem recorrer em caso de discordância em relação à qualidade do serviço prestado. O mesmo ocorre caso o auditor seja de confiança apenas do provedor e, por isso, nesses casos não se verifica qualquer vantagem na contratação de um auditor externo.

Deve-se verificar também que nem todo auditor pode ser capaz de avaliar todos os tipos de serviço fornecidos por um provedor. Por exemplo, os auditores A e B podem ser capazes de avaliar o serviço de execução de tarefas e o auditor C pode ser capaz de avaliar o serviço de armazenamento de dados. Além disso, o auditor A pode ser capaz de avaliar apenas a disponibilidade do serviço, enquanto o auditor B pode ser capaz de avaliar tanto a disponibilidade como o tempo de resposta desse mesmo serviço.

Assim, dependendo dos requisitos de auditoria apresentados pelas partes, apenas uma parte dos auditores confiáveis pode ser capaz de atender às necessidades do provedor e do consumidor. Portanto, assim como o provedor disponibiliza modelos de acordos de serviço descrevendo o que ele é capaz de atender, os auditores também devem disponibilizar seus modelos de acordos de serviço a fim de que seja possível verificar se um determinado auditor pode atender aos requisitos de auditoria de uma interação consumidor-provedor.

É importante também que o mecanismo de negociação de auditoria proposto seja independente e o mais transparente possível tanto para o consumidor como para o provedor de serviços. Na prática, isso implica que deve ser possível inserir ou remover o mecanismo de negociação de auditoria com o mínimo de intrusividade. Preferencialmente, portanto, não deve ser necessário alterar as interfaces de comunicação entre consumidor e provedor ou recompilar serviços que já funcionam sem a negociação de auditoria (Figura 5.2).

## 5.3 Visão Geral do Mecanismo Proposto

Com base nas características apresentadas na Seção 5.2, a Figura 5.3 apresenta o mecanismo proposto de negociação de auditor externo para acordos de nível de serviço em grades computacionais. Na Figura 5.3, o consumidor confia no auditor 1 e o provedor confia nos auditores 1 e 2. A interação se inicia com a solicitação dos modelos de acordo ao provedor de serviços (1). Em seguida, o consumidor cria uma oferta de acordo de serviço baseada em um dos modelos obtidos do provedor (2). Até este momento, tudo ocorre como se não houvesse negociação de auditor externo (Figura 5.2). Em seguida, como o consumidor está interessado em auditoria, este solicita ao provedor os modelos de acordo dos auditores de confiança do provedor (3). O provedor, então, solicita os modelos de acordo aos auditores (4, 5) e os entrega ao consumidor. Em seguida, para cada auditor de confiança do consumidor que está na lista de auditores de confiança do provedor (inferida pelos modelos de acordo obtidos), o consumidor cria uma oferta de acordo de serviço de auditoria (6) e solicita a execução desse acordo por intermédio do provedor de serviços (7). Junto com a oferta de serviço do auditor, o provedor de serviços também recebe do consumidor a oferta do serviço a ser prestado pelo provedor e a indicação do auditor que deve ser contratado. O provedor, então, valida ambas as ofertas (8) e solicita a execução do serviço de auditoria ao auditor indicado pelo consumidor (9). Em seguida, o provedor armazena a oferta de serviço de auditoria (10) e responde ao consumidor. Caso a resposta seja negativa, o consumidor enviará uma nova oferta de serviço de auditoria, repetindo o processo (6, 7, 8, 9, 10), ou abortará a negociação. Caso a resposta seja positiva, o consumidor solicita a execução do acordo de serviço do provedor por intermédio do auditor indicado (11), conforme representado na Arquitetura 5 (Figura 4.6). O auditor solicita ao provedor a execução do acordo de nível de serviço (12) e começa o seu monitoramento (13). O provedor, por sua vez, envia a tarefa ao Gerenciador de Recursos para execução (14).

A região cinza da Figura 5.3 destaca os passos introduzidos pelo mecanismo de negociação de auditoria em relação à Figura 5.2, que apresenta uma interação sem negociação de auditoria.

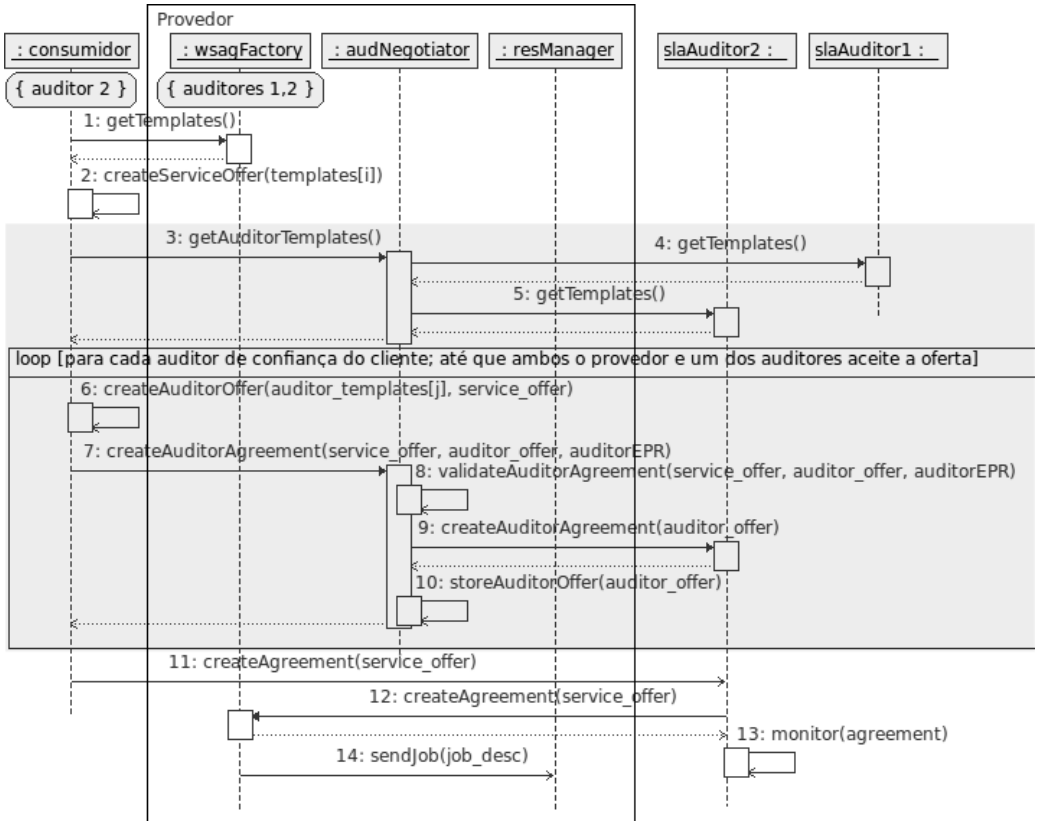


Figura 5.3: Interação entre consumidor e provedor de serviços com auditoria.

## 5.4 Algoritmos do Mecanismo de Negociação

Os algoritmos 1, 2 e 3 detalham o funcionamento do mecanismo de negociação de auditoria proposto na Figura 5.3. Apesar de não apresentarem tratamentos de exceção e validação de entradas e saídas, esses algoritmos fornecem uma visão mais detalhada do mecanismo de negociação de auditoria e dão subsídios para a avaliação teórica apresentada e discutida na Seção 7.1.

O Algoritmo 1 executa o mecanismo de negociação de auditores. Inicialmente, o consumidor solicita os modelos de acordo do provedor (linha 1) e, a partir de um deles, é criada uma oferta de acordo (linha 2). Esses passos são genéricos, ocorrendo também quando não se utiliza o mecanismo de negociação de auditoria. Em seguida, o consumidor solicita ao provedor os modelos de acordo de cada auditor de confiança do provedor (linha 3). O passo da linha 3 é detalhado no Algoritmo 2, que será explicado posteriormente. A linha 4 inicializa uma variável de controle para o laço da linha 5. Na linha 5, é iniciado um laço que itera sobre cada auditor de confiança do provedor. Dentro do laço, o consumidor verifica se o auditor corrente da lista de auditores de confiança do provedor também é de confiança do consumidor (linha 6). Se sim, uma oferta de serviço de auditoria é criada com base no modelo de acordo daquele auditor específico e na oferta de serviço do provedor (linha 7). Na linha 8, a oferta de auditoria é enviada ao provedor para aceitação. O passo da linha 8 é detalhado no Algoritmo 3, que será explicado posteriormente. Na linha 10, a variável de controle do laço é incrementada. A linha 12, fora do laço, verifica se alguma oferta foi aceita pelo provedor e por um dos auditores durante a execução do laço. Se sim,

**Algoritmo 1** Algoritmo do mecanismo de negociação de auditores.

**Entrada:** os auditores de confiança do consumidor ($C$).

```
1: templates ← provedor.getTemplates()
2: serviceOffer ← createServiceOffer(templates)
3: audTemp ← provedor.getAuditorTemplates()
4: i ← 0
5: while i < audTemp.length and not accepted do
6:    if audTemp[i].auditor in C then
7:       auditorOffer ←
         createAuditorOffer(audTemp[i], serviceOffer)
8:       accepted ←
         provedor.createAuditorAgreement(
         serviceOffer, auditorOffer, audTemp[i].auditor)
9:    end if
10:   i++
11: end while
12: if accepted then
13:   audTemp[i − 1].auditor.createAgreement(serviceOffer)
14: end if
```

a oferta de serviço é enviada para o auditor selecionado. O auditor selecionado, então, apenas repassa a solicitação ao provedor e inicia a monitoração do acordo.

O mecanismo utilizado pelo auditor para monitorar o serviço prestado pelo provedor não é objeto desse estudo. O principal objeto desse estudo é o mecanismo para negociação e estabelecimento de um auditor externo que possa servir de árbitro na relação consumidor-provedor no que diz respeito ao nível de serviço prestado. Esse mecanismo é independente da técnica utilizada pelo auditor para avaliar a execução do serviço e as requisições realizadas pelo consumidor.

**Algoritmo 2** Método getAuditorTemplates() do provedor (linha 3 do Algoritmo 1).

**Entrada:** os auditores de confiança do provedor ($P$).

**Saída:** os auditores de confiança do provedor e seus respectivos modelos de acordo ($audTemp$).

```
1: for i = 0 to P.length do
2:    audTemp[i].auditor ← P[i]
3:    audTemp[i].templates ← P[i].getTemplates()
4: end for
5: return audTemp
```

O Algoritmo 2 apresenta a lógica do provedor para solicitar os modelos de acordo dos seus auditores de confiança. Na linha 1, é iniciado um laço que itera sobre cada auditor de confiança do provedor. O provedor armazena o identificador para contato de cada auditor de confiança em uma variável de retorno chamada *audTemp* (linha 2). Na linha 3, o provedor solicita os modelos de acordo disponíveis em cada auditor de confiança e os armazena também na variável *audTemp*. Em seguida, o provedor retorna a variável

*audTemp* na linha 5, com as informações de contato de cada auditor de confiança do provedor e seus respectivos modelos de acordo.

**Algoritmo 3** Método createAuditorAgreement() do provedor (linha 8 do Algoritmo 1).

**Entrada:** uma oferta de serviço ($serviceOffer$), uma oferta de auditoria ($auditorOffer$) e o auditor escolhido ($auditor$).

**Saída:** $true$ se a oferta é válida e o auditor a aceitou. Caso contrário, $false$.

```
1: accepted ← validateAuditorAgreement(
   serviceOffer, auditorOffer, auditor)
2: if accepted then
3:     accepted ←
       auditor.createAuditorAgreement(auditorOffer)
4:     storeAuditorOffer(auditorOffer)
5: end if
6: return accepted
```

O Algoritmo 3 detalha a lógica de criação do acordo de auditoria. Na linha 1, o provedor valida as ofertas de serviço e de auditoria. Nesse momento, o provedor pode negar a oferta de acordo de serviço devido a alguma inconsistência na oferta em relação aos modelos fornecidos ou apenas por não poder cumprir o acordo por algum motivo. O provedor também pode negar a oferta de serviço de auditoria por não concordar com os termos da auditoria. Em ambos os casos, o consumidor receberá uma resposta negativa e, então, poderá lançar uma nova oferta ou apenas abortar a negociação. A lógica de validação das ofertas varia de acordo com a política de execução de serviços do provedor e com a capacidade momentânea dos seus recursos. No mínimo, porém, é interessante que o provedor verifique se a oferta de serviço está de acordo com um dos modelos fornecidos ao consumidor e se a oferta de auditoria está coerente com a oferta de serviço. Caso o provedor valide as ofertas com sucesso, ele solicita a execução do serviço de auditoria ao auditor (linhas 2 e 3). O auditor, por sua vez, também pode negar a execução do serviço por algum motivo. Porém, quando o auditor aceita um acordo de auditoria, ele se compromete com a execução da auditoria sobre o serviço a ser fornecido pelo provedor, conforme especificado no acordo. Na linha 4, o provedor armazena a oferta de auditoria enviada ao auditor, de forma a poder referenciá-la em caso de necessidade. Essa necessidade pode ocorrer, por exemplo, em caso de falha temporária do serviço de auditoria. Nesse caso, pode ser necessário enviar novamente a proposta de auditoria ao auditor para que o serviço de auditoria seja restabelecido. Na linha 6, o provedor retorna *true* ou *false* de acordo com a aceitação das ofertas pelo provedor e pelo auditor.

Note que, depois da negociação do auditor, toda interação que seria realizada diretamente entre consumidor e provedor passa a ser intermediada pelo auditor (linha 13 do Algoritmo 1). Dessa forma, o auditor pode validar a execução do serviço tanto do ponto de vista do desempenho do provedor como do ponto de vista das requisições realizadas pelo consumidor. Contudo, para que o auditor possa solicitar serviços ao provedor, é necessário que este realize as solicitações em favor do consumidor, ou seja, para o provedor, as solicitações vindas do auditor devem ser previamente autorizadas pelo consumidor. Isto acontece porque o acordo de nível de serviço é estabelecido entre duas partes: o provedor e o consumidor. Ou seja, o provedor tem direitos e deveres contratuais apenas com o con-

sumidor. Além disso, segundo a Arquitetura 5 (Figura 4.6), o auditor não deve realizar requisições adicionais ao provedor. Assim, toda requisição ao provedor vem diretamente do consumidor (caso não tenha sido estabelecido um auditor) ou de um auditor em favor de um consumidor (caso o auditor já tenha sido estabelecido). Na prática, esse requisito pode ser implementado através de delegação de credenciais do consumidor para o auditor no momento da solicitação de execução do acordo de serviço. Na Seção 6.3, a delegação de credenciais é apresentada em detalhes no contexto do protótipo do mecanismo de negociação implementado.

Note também que, de acordo com os algoritmos 1, 2 e 3, a negociação do auditor sempre termina, seja pela falta de auditores confiáveis a negociar ou pela escolha de um dos auditores confiáveis. Ou seja, o mecanismo de negociação não entrará em *loop* infinito, uma vez que a lista de auditores confiáveis é finita, tanto no provedor como no consumidor.

## 5.5 Serviços do Mecanismo de Negociação

Conforme pode ser verificado, o mecanismo proposto na Figura 5.3 acrescentou dois tipos de serviço à interação apresentada na Figura 5.2. Esses novos serviços são o *audNegotiator* e o *slaAuditor*. Note que os demais serviços (*wsagFactory* e *resManager*) são independentes do mecanismo de negociação, visto que suas operações são as mesmas de quando não há negociação de auditoria (Figura 5.2). Os serviços *audNegotiator* e *slaAuditor*, porém, são específicos do mecanismo de negociação de auditoria e são detalhados a seguir.

Na interação apresentada na Figura 5.3, pode-se observar que a negociação do auditor é realizada pelo novo serviço *audNegotiator*. Esse serviço é responsável por dar suporte a todas as novas mensagens que são trocadas com consumidor e auditores durante o processo de negociação da auditoria. Assim, tanto a interface como a implementação originais do provedor não são modificadas. Ao invés disso, apenas o novo serviço *audNegotiator* é adicionado ao provedor e sua interface é publicada para que o consumidor tenha acesso. Dessa forma, a introdução do serviço de negociação de auditores é transparente para o consumidor e para os serviços já existentes no provedor, o que facilita a sua implantação.

O serviço *slaAuditor*, por sua vez, representa o auditor cujo estabelecimento na interação consumidor-provedor é negociado. O serviço *slaAuditor* é responsável por informar os modelos de acordo disponíveis ao provedor, bem como validar e executar um acordo de auditoria, conforme especificado pelo consumidor e validado pelo provedor.

## 5.6 Interfaces dos Serviços do Mecanismo de Negociação

A Figura 5.4 mostra um extrato da interface pública do serviço *audNegotiator* especificada em Web Services Description Language (Seção 2.4.2).

Conforme a Figura 5.4, o serviço *audNegotiator* disponibiliza duas operações: *getAuditorTemplates* e *createAuditorAgreement* (linhas 34 e 38). Cada uma dessas operações possui uma mensagem de entrada e uma mensagem de saída. No caso da operação *createAuditorAgreement*, a mensagem de entrada é *CreateAuditorAgreementRequestMessage* e a de saída é *CreateAuditorAgreementResponseMessage* (linhas 39 e 40). Na seção Men-

```
1     </xsd:complexType>
2 <!-- ... ============= T Y P E S -->
3     <xsd:element name="CreateAuditorAgreementRequest">
4         <xsd:complexType>
5             <xsd:sequence>
6 <xsd:element name="ServiceOffer" type="wsag:AgreementType" minOccurs="0" />
7 <xsd:element name="AuditorOffer" type="wsag:AgreementType" minOccurs="0" />
8 <xsd:element name="AuditorEPR" type="wsa:EndpointReferenceType" minOccurs="0" />
9             </xsd:sequence>
10        </xsd:complexType>
11    </xsd:element>
12    <xsd:element name="CreateAuditorAgreementResponse">
13        <xsd:complexType/>
14    </xsd:element>
15 </xsd:schema>
16 </types>
17 <!--================== M E S S A G E S -->
18 <message name="GetAuditorTemplatesRequestMessage">
19     <part name="parameters" element="tns:GetAuditorTemplatesRequest"/>
20 </message>
21 <message name="GetAuditorTemplatesResponseMessage">
22     <part name="parameters" element="tns:GetAuditorTemplatesResponse"/>
23 </message>
24 <message name="CreateAuditorAgreementRequestMessage">
25     <part name="parameters"
26         element="tns:CreateAuditorAgreementRequest" />
27 </message>
28 <message name="CreateAuditorAgreementResponseMessage">
29     <part name="parameters"
30         element="tns:CreateAuditorAgreementResponse" />
31 </message>
32 <!--================== P O R T T Y P E -->
33 <portType name="SlaAuditorNegotiationPortType">
34     <operation name="getAuditorTemplates">
35         <input message="tns:GetAuditorTemplatesRequestMessage"/>
36         <output message="tns:GetAuditorTemplatesResponseMessage"/>
37     </operation>
38     <operation name="createAuditorAgreement">
39         <input message="tns:CreateAuditorAgreementRequestMessage"/>
40         <output message="tns:CreateAuditorAgreementResponseMessage"/>
41     </operation>
42 </portType>
```

Figura 5.4: Especificação da interface pública do serviço *audNegotiator*.

sagens (linhas 18 a 31), podem ser encontrados os parâmetros de cada uma dessas mensagens. A mensagem *CreateAuditorAgreementRequestMessage* (linha 24), por exemplo, possui um parâmetro do tipo *CreateAuditorAgreementRequest* e a mensagem *CreateAuditorAgreementResponseMessage* (linha 28) possui um parâmetro do tipo *CreateAuditorAgreementResponse*. Esses tipos estão definidos na seção Tipos (linhas 3 a 16), conforme apresentado na Figura 5.4. O tipo *CreateAuditorAgreementRequest* (linhas 3 a 11), por exemplo, é um tipo composto por três elementos: *ServiceOffer*, *AuditorOffer* e *AuditorEPR*. O tipo *CreateAuditorAgreementResponse* (linhas 12 a 14), por sua vez, não possui elementos, ou seja, não carrega consigo qualquer informação.

As operações definidas em WSDL e seus respectivos tipos devem ter uma correspondência na implementação, ou seja, as operações devem ter métodos correspondentes no código Java, assim como os tipos também devem ter classes correspondentes na implementação. A Figura 5.5 apresenta a implementação da operação *getAuditorTemplates* na linguagem Java. Note que a declaração do método *getAuditorTemplates* apresenta um tipo de retorno chamado *GetAuditorTemplatesResponse* (linha 1) e um parâmetro de entrada chamado *GetAuditorTemplatesRequest* (linha 2). Essas classes têm correspondência direta com o tipo do parâmetro especificado em WSDL para a mensagem da operação *getAuditorTemplates* (linhas 19 e 22 da Figura 5.4).

```
public GetAuditorTemplatesResponse getAuditorTemplates(
        GetAuditorTemplatesRequest request) throws RemoteException {
    logger.debug("SlaAuditorNegotiationService.getAuditorTemplates(GetAuditorTemplatesRequest)");
    EndpointReferenceType[] auditorEPRs = getResource().getAuditorEPRs();
    AuditorTemplateDuo[] auditorTemplateDuos = new AuditorTemplateDuo[auditorEPRs.length];
    for (int i = 0; i < auditorEPRs.length; i++) {
        AuditorTemplateDuo auditorTemplateDuo = new AuditorTemplateDuo();
        EndpointReferenceType auditorEPR = auditorEPRs[i];
        AgreementTemplateType[] templates = getSlaAuditorTemplates(auditorEPR);
        auditorTemplateDuo.setEPR(auditorEPR);
        auditorTemplateDuo.setTemplates(templates);
        auditorTemplateDuos[i] = auditorTemplateDuo;
    }
    GetAuditorTemplatesResponse response = new GetAuditorTemplatesResponse();
    response.setAuditorTemplateDuos(auditorTemplateDuos);
    return response;
}
```

Figura 5.5: Implementação da operação *getAuditorTemplates* da interface do serviço *audNegotiator*.

Da mesma forma que o serviço *audNegotiator*, o serviço *slaAuditor* também possui uma interface especificada em WSDL e uma implementação correspondente em código Java. O serviço *slaAuditor*, porém, possui 3 operações: *createAgreement*, *createAuditorAgreement* e *getTemplates*, conforme apresentado na Figura 5.6.

```
<!--============================================================
                        P O R T T Y P E
  ============================================================-->
<portType name="SlaAuditorPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrpw:GetMultipleResourceProperties
                    wsrpw:SetResourceProperties
                    wsrpw:QueryResourceProperties">

    <operation name="createAgreement">
        <input message="tns:CreateAgreementRequestMessage"/>
        <output message="tns:CreateAgreementResponseMessage"/>
    </operation>

    <operation name="createAuditorAgreement">
        <input message="tns:CreateAuditorAgreementRequestMessage"/>
        <output message="tns:CreateAuditorAgreementResponseMessage"/>
    </operation>

    <operation name="getTemplates">
        <input message="tns:GetTemplatesRequestMessage"/>
        <output message="tns:GetTemplatesResponseMessage"/>
    </operation>
</portType>
```

Figura 5.6: Especificação da interface pública do serviço *slaAuditor*.

## 5.7 Modelos e Ofertas do Auditor

### 5.7.1 Modelo de Acordo do Serviço de Auditoria

O objetivo final do consumidor é estabelecer um acordo para a execução de um serviço no provedor. O mecanismo de negociação aqui proposto busca garantir que as partes não serão lesadas durante a execução desse acordo. Para isso, o mecanismo estabelece um árbitro para a relação, chamado auditor.

A introdução do auditor na relação se caracteriza como o estabelecimento de mais um acordo de serviço: o serviço de auditoria prestado pelo auditor. Assim, da mesma

forma em que um acordo é estabelecido para a execução do serviço no provedor, outro acordo é estabelecido com o auditor para o serviço de auditoria. Esses acordos, conforme apresentado na Seção 3.2.2, são derivados de modelos de acordo. A Figura 5.7 apresenta as duas principais seções de um modelo de acordo de auditoria: *Terms* (linhas 1 a 10) e *CreationConstraints* (linhas 12 a 38).

O único termo desse modelo de acordo de auditoria está descrito nas linhas 4 a 7 da Figura 5.7. O nome do termo é *RESPONSE_TIME* e o nome do serviço correspondente é *RESPONSE_TIME_SERVICE*. Esse é o serviço do auditor que monitora o tempo de resposta do provedor durante a execução de um acordo. Na linha 6, é definido um valor padrão em segundos para o tempo de resposta esperado para o provedor.

A seção *CreationConstraints* define o conteúdo esperado do termo *RESPONSE_TIME* em uma oferta baseada nesse modelo. As linhas 13 a 25 definem que o conteúdo deve ser um parâmetro do tipo inteiro chamado *response_time_value*. As linhas 27 a 37, por sua vez, definem a faixa de valores inteiros aceitos pelo serviço de auditoria de tempo de resposta. No caso do exemplo, essa faixa é entre 6 e 60 segundos (linhas 34 e 35). Essa definição implica, portanto, que o auditor é capaz de avaliar o tempo de resposta do provedor em qualquer valor inteiro entre 6 e 60 segundos.

```
1     <ns1:Terms xsi:type="ns1:TermTreeType">
2         <ns1:All xsi:type="ns1:TermCompositorType">
3             <ns1:OneOrMore xsi:type="ns1:TermCompositorType">
4                 <ns1:ServiceDescriptionTerm ns1:Name="RESPONSE_TIME"
5 ns1:ServiceName="RESPONSE_TIME_SERVICE" xsi:type="ns1:ServiceDescriptionTermType">
6                     <xs:integer name="response_time_value">8</xs:integer>
7                 </ns1:ServiceDescriptionTerm>
8             </ns1:OneOrMore>
9         </ns1:All>
10    </ns1:Terms>
11
12    <ns1:CreationConstraints xsi:type="ns1:ConstraintSectionType">
13        <ns1:Item ns1:Name="ResponseTimeServiceDescriptionConstraint"
14 xsi:type="ns1:OfferItemType">
15            <ns1:Location xsi:type="xs:string"
16 xmlns:xsd="http://www.w3.org/2001/XMLSchema">declare namespace ns1='http://
17 schemas.ggf.org/graap/2007/03/ws-agreement';$this/ns1:AgreementOffer/ns1:Terms/
18 ns1:All/ns1:ServiceDescriptionTerm[@ns1:Name='RESPONSE_TIME']</ns1:Location>
19            <ns1:ItemConstraint>
20                <xs:sequence xmlns:xs="http://www.w3.org/2001/XMLSchema">
21                    <xs:element maxOccurs="1" minOccurs="1"
22 name="response_time_value" type="xs:integer" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
23                </xs:sequence>
24            </ns1:ItemConstraint>
25        </ns1:Item>
26
27        <ns1:Item ns1:Name="ResponseTimeValue" xsi:type="ns1:OfferItemType">
28            <ns1:Location xsi:type="xsd:string"
29 xmlns:xsd="http://www.w3.org/2001/XMLSchema">declare namespace xs='http://
30 www.w3.org/2001/XMLSchema';declare namespace ns1='http://schemas.ggf.org/graap/2007/03/
31 ws-agreement';$this/ns1:AgreementOffer/ns1:Terms/ns1:All/
32 ns1:ServiceDescriptionTerm[@ns1:Name='RESPONSE_TIME']/xs:integer</ns1:Location>
33            <ns1:ItemConstraint>
34                <xs:minInclusive value="6" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
35                <xs:maxInclusive value="60" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
36            </ns1:ItemConstraint>
37        </ns1:Item>
38    </ns1:CreationConstraints>
```

Figura 5.7: Modelo de acordo do serviço de auditoria.

### 5.7.2 Oferta de Acordo do Serviço de Auditoria

Após receber o modelo de acordo da Figura 5.7, o consumidor cria uma oferta de acordo de auditoria baseada nesse modelo. A oferta deve fazer referência ao modelo no qual

se baseia e também deve respeitar as restrições impostas pelo modelo. Vale salientar, porém, que mesmo seguindo essas orientações, nada garante que o auditor irá aceitar a oferta. Isto pode ocorrer porque o modelo de acordo não é uma garantia, mas apenas uma indicação de que termos de acordo e quais valores para esses termos o auditor tem mais probabilidade de aceitar. A Figura 5.8 apresenta uma possível oferta de acordo baseada no modelo da Figura 5.7.

```
 1 <ns1:AgreementOffer
 2     xmlns:ns1="http://schemas.ggf.org/graap/2007/03/ws-agreement"
 3     xmlns:xs="http://www.w3.org/2001/XMLSchema"
 4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 5     xsi:type="ns1:AgreementTemplateType"
 6     ns1:AgreementId="1">
 7
 8  <ns1:Name xsi:type="xsd:string"
 9 xmlns:xsd="http://www.w3.org/2001/XMLSchema">AUDITOR_OFFER_1</ns1:Name>
10
11     <ns1:Context xsi:type="ns1:AgreementContextType">
12         <ns1:ServiceProvider xsi:type="ns1:AgreementRoleType">
13 AgreementResponder</ns1:ServiceProvider>
14         <ns1:TemplateId xsi:type="xsd:string"
15 xmlns:xsd="http://www.w3.org/2001/XMLSchema">1</ns1:TemplateId>
16         <ns1:TemplateName xsi:type="xsd:string"
17 xmlns:xsd="http://www.w3.org/2001/XMLSchema">AUDITOR_TEMPLATE_1</ns1:TemplateName>
18     </ns1:Context>
19
20     <ns1:Terms xsi:type="ns1:TermTreeType">
21         <ns1:All xsi:type="ns1:TermCompositorType">
22             <ns1:OneOrMore xsi:type="ns1:TermCompositorType">
23
24                 <ns1:ServiceDescriptionTerm ns1:Name="RESPONSE_TIME"
25 ns1:ServiceName="RESPONSE_TIME_SERVICE" xsi:type="ns1:ServiceDescriptionTermType">
26                     <xs:integer name="response_time_value">8</xs:integer>
27                 </ns1:ServiceDescriptionTerm>
28
29             </ns1:OneOrMore>
30         </ns1:All>
31     </ns1:Terms>
32 </ns1:AgreementOffer>
```

Figura 5.8: Oferta de acordo do serviço de auditoria.

As linhas 8 e 9 definem o nome da oferta de acordo e as linhas 14 a 17 referenciam o modelo de acordo no qual a oferta se baseia. As linhas 20 a 31 definem os termos da oferta de acordo. No caso do exemplo, existe um único termo na oferta que é exatamente o termo *RESPONSE_TIME* (linhas 24 a 27). O parâmetro passado para o serviço *RESPONSE_TIME_SERVICE* chama-se *response_time_value* e possui o valor inteiro 8. Essa definição está de acordo com as restrições impostas no modelo de acordo apresentado na Figura 5.7 e, por isso, esta é uma oferta válida. Note que o *response_time_value* especificado na oferta de acordo do auditor diz respeito ao tempo de resposta a ser monitorado pelo auditor e, por isso, normalmente será igual ao tempo de resposta especificado na oferta de acordo do provedor para o serviço solicitado.

## 5.8 Modelos e Ofertas do Provedor

### 5.8.1 Modelo de Acordo do Serviço do Provedor

Assim como o auditor, o provedor também possui modelos de acordo para os seus serviços. Esses modelos de acordo são utilizados pelo consumidor para definir uma oferta de serviço que será enviada para o provedor. A Figura 5.9 apresenta um extrato de exemplo de modelo de acordo apresentado para o consumidor. Esse extrato destaca a seção

*CreationConstraints* do modelo de acordo. Conforme mencionado anteriormente, essa é a seção do modelo que define as restrições para os valores dos termos do serviço. No caso do exemplo, essa seção possui duas restrições, sendo a primeira definida nas linhas 2 a 12 e a segunda nas linhas 14 a 29.

```
1 <ns1:CreationConstraints xsi:type="ns1:ConstraintSectionType">
2 <ns1:Item ns1:Name="ResponseTimeValue" xsi:type="ns1:OfferItemType">
3     <ns1:Location xsi:type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4         declare namespace xs='http://www.w3.org/2001/XMLSchema';declare namespace
5         ns1='http://schemas.ggf.org/graap/2007/03/ws-agreement';$this/ns1:AgreementOffer/
6         ns1:Terms/ns1:All/ns1:GuaranteeTerm[@ns1:Name='RESPONSE_TIME_GUARANTEE']/
7         ns1:ServiceLevelObjective/ns1:KPITarget/ns1:CustomServiceLevel</ns1:Location>
8     <ns1:ItemConstraint>
9         <xs:minInclusive value="4" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
10        <xs:maxInclusive value="18" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
11    </ns1:ItemConstraint>
12 </ns1:Item>
13
14 <ns1:Item ns1:Name="OfferStructureConstraint" xsi:type="ns1:OfferItemType">
15    <ns1:Location xsi:type="xs:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
16        declare namespace ns1='http://schemas.ggf.org/graap/2007/03/ws-agreement';
17        $this/ns1:AgreementOffer/ns1:Terms/ns1:All</ns1:Location>
18    <ns1:ItemConstraint>
19        <xs:sequence xmlns:xs="http://www.w3.org/2001/XMLSchema"
20            xmlns:ns1="http://schemas.ggf.org/graap/2007/03/ws-agreement">
21            <xs:element name="ExactlyOne" minOccurs="1" maxOccurs="1"
22                type="ns1:TermCompositorType"/>
23            <xs:element name="ServiceDescriptionTerm" type="ns1:ServiceDescriptionTermType"/>
24            <xs:element name="ServiceDescriptionTerm" type="ns1:ServiceDescriptionTermType"/>
25            <xs:element maxOccurs="1" minOccurs="0" name="RESPONSE_TIME_GUARANTEE"
26                type="ns1:GuaranteeTerm"/>
27        </xs:sequence>
28    </ns1:ItemConstraint>
29 </ns1:Item>
30 </ns1:CreationConstraints>
```

Figura 5.9: Modelo de acordo do serviço do provedor.

A primeira restrição, chamada *ResponseTimeValue*, define as restrições para os valores possíveis de tempo de resposta esperado do provedor. Assim, as linhas 9 e 10 informam ao consumidor que o provedor é capaz de oferecer um serviço em que o tempo de resposta máximo pode variar entre 4 e 18. Note que essas restrições se referem à capacidade do provedor em fornecer um serviço com um determinado tempo de resposta. Isso é diferente da restrição de tempo de resposta apresentada pelo auditor no seu modelo de acordo (Figura 5.7), visto que a restrição apresentada pelo auditor diz respeito à capacidade do auditor em avaliar o tempo de resposta do provedor. Por exemplo, um provedor pode ser capaz de realizar determinado serviço com um tempo de resposta entre 4 e 10 segundos, enquanto um auditor pode ser capaz de monitorar aquele serviço do provedor apenas a partir de 7 segundos. Nesse caso, se um consumidor solicitar o serviço do provedor com um tempo de resposta de 4 segundos, o auditor em questão não será capaz de avaliar adequadamente as violações de acordo, visto que ele não conseguirá identificar tempos de resposta abaixo de 7 segundos.

A segunda restrição, chamada *OfferStructureConstraint*, define restrições para a estrutura dos termos de uma oferta baseada nesse modelo. Assim, as linhas 19 a 27 informam ao consumidor que os termos da oferta devem consistir de uma sequência de elementos do tipo *ServiceDescriptionTerm* seguidas de um elemento opcional do tipo *GuaranteeTerm* chamado *RESPONSE_TIME_GUARANTEE*, que representa a especificação de tempo de resposta máximo esperado pelo consumidor do serviço. Note que o tempo de resposta no modelo do auditor (Figura 5.7) é especificado como um parâmetro do serviço de auditoria de tempo de resposta e que, no modelo do provedor, o tempo de resposta é especificado como um termo de garantia (Figura 5.9) do serviço provido. Isto acontece porque o tempo

de resposta no auditor, conforme mencionado anteriormente, não é uma garantia do serviço de auditoria de tempo de resposta, mas sim uma configuração deste para avaliar o provedor. No caso do provedor, por outro lado, o tempo de resposta é de fato uma garantia sobre a execução do serviço provido.

### 5.8.2 Oferta de Acordo do Serviço do Provedor

O modelo apresentado na Figura 5.9 é utilizado pelo consumidor para compor uma oferta a ser enviada ao provedor. A Figura 5.10 apresenta um extrato de um exemplo de oferta baseada no modelo apresentado na Figura 5.9. Esse extrato apresenta um termo de serviço (linhas 1 a 13) e um termo de garantia (linhas 15 a 25).

```
 1 <ns1:ExactlyOne xsi:type="ns1:TermCompositorType">
 2     <ns1:ServiceDescriptionTerm ns1:Name="APPLICATION_STD"
 3         ns1:ServiceName="UNICORE6" xsi:type="ns1:ServiceDescriptionTermType">
 4         <jsdl:JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl">
 5             <jsdl:JobDescription>
 6                 <jsdl:Application>
 7                     <jsdl:ApplicationName>testtouch</jsdl:ApplicationName>
 8                     <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
 9                 </jsdl:Application>
10             </jsdl:JobDescription>
11         </jsdl:JobDefinition>
12     </ns1:ServiceDescriptionTerm>
13 </ns1:ExactlyOne>
14
15 <ns1:GuaranteeTerm Name="RESPONSE_TIME_GUARANTEE" Obligated="ServiceProvider">
16     <ns1:ServiceScope ServiceName="UNICORE6"></ns1:ServiceScope>
17     <ns1:ServiceLevelObjective>
18         <ns1:KPITarget>
19             <ns1:KPIName>RESPONSE_TIME</ns1:KPIName>
20             <ns1:CustomServiceLevel xsi:type="xsd:integer"
21                 xmlns:xsd="http://www.w3.org/2001/XMLSchema">8</ns1:CustomServiceLevel>
22         </ns1:KPITarget>
23     </ns1:ServiceLevelObjective>
24     <ns1:BusinessValueList></ns1:BusinessValueList>
25 </ns1:GuaranteeTerm>
```

Figura 5.10: Oferta de acordo do serviço do provedor.

O termo de serviço da Figura 5.10 faz referência ao serviço *UNICORE6* (linha 3) e especifica uma tarefa ser executada remotamente. A tarefa a ser executada é uma aplicação chamada *testtouch* (linha 7) e cuja versão é 1.0 (linha 8).

O termo de garantia chama-se *RESPONSE_TIME_GUARANTEE* na Figura 5.10 e a parte obrigada a cumpri-lo é o provedor (linha 15). O termo de garantia se aplica ao serviço *UNICORE6*, conforme especificado na linha 16. O objetivo de nível de serviço (linhas 17 a 23) define qual é o indicador associado ao termo de garantia e qual é o valor esperado para esse indicador. O indicador desse termo de garantia é o tempo de resposta (linha 19) e o valor máximo esperado é de 8 segundos (linhas 20 e 21). Ou seja, o consumidor espera que o serviço *UNICORE6* (que é a execução de uma tarefa remotamente) seja executado em, no máximo, 8 segundos. O elemento *BusinessValueList* que está vazio no exemplo poderia ser usado para especificar uma multa a ser aplicada ao provedor em caso de violação desse termo de garantia.

# Capítulo 6

# Protótipo do Mecanismo de Negociação

Um protótipo do mecanismo de negociação de auditor foi implementado em Java e o Globus Toolkit 4.0.8 (Seção 2.3) foi utilizado como *middleware* de grade computacional. O Globus Toolkit foi escolhido por ser um dos mais importantes *middlewares* de grade computacional atuais, contando com uma grande comunidade de usuários, e também por sua estreita relação com o órgão que criou e mantém as principais especificações de serviços e arquiteturas para grades computacionais, que é o *Open Grid Forum* (OGF). A maturidade do Globus Toolkit no mercado se reflete em sua grande comunidade de suporte e facilita a implementação de soluções que buscam aderência aos padrões definidos pelo OGF.

Para obter suporte ao padrão WS-Agreement 1.0 (Seção 3.2.2) no Globus Toolkit 4.0.8, foi utilizado o pacote de serviços SLA4D-Grid Negotiation Manager 1.2. Conforme apresentado na Seção 2.3, o SLA4D-Grid Negotiation Manager 1.2 é um pacote de serviços Web que implementa a especificação do WS-Agreement 1.0 e que pode ser facilmente integrado ao Globus Toolkit 4.0.8. Com esse conjunto de serviços, é possível estabelecer acordos de nível de serviço seguindo o padrão WS-Agreement 1.0.

A Figura 6.1 ilustra as camadas e módulos de sistemas utilizados pelo mecanismo de negociação de auditoria. Na camada superior estão os serviços de negociação e de auditoria de qualidade de serviços, que são representados pelos serviços *audNegotiator* e *slaAuditor* na Figura 5.3. Na camada intermediária, está o pacote de serviços SLA4D-Grid Negotiation Manager 1.2, do qual dependem os serviços da camada superior. Nesse pacote está a implementação do padrão WS-Agreement, que é representada pelo serviço *wsagFactory* na Figura 5.3. Na camada inferior, estão o Globus Toolkit 4.0.8 e o Torque/PBS 2.5.2.

Na prática, os serviços das camadas superior e intermediária são implantados como serviços Web no Globus Toolkit 4.0.8, conforme o diagrama de implantação apresentado na Figura 6.2. O Torque/PBS 2.5.2 (Seção 2.3), por sua vez, é o gerenciador de recursos utilizado em conjunto com o Globus Toolkit. O gerenciador de recursos está representado na Figura 5.3 sob o nome *resManager*.

Como mencionado na Seção 5.3, a forma como as responsabilidades foram separadas entre *wsagFactory* e *audNegotiator* garante a independência entre esses serviços. Dessa forma, o mecanismo de negociação de auditoria aqui proposto é completamente inde-


Figura 6.1: Módulos e camadas do mecanismo de negociação de auditoria.

pendente da implementação do WS-Agreement 1.0 utilizada. Ou seja, esse mecanismo depende apenas da especificação do WS-Agreement, e não da implementação escolhida.

Assim como os demais serviços do mecanismo, a implementação do auditor também foi realizada na linguagem Java e implantada como um serviço Web na plataforma Globus Toolkit 4.0.8. Isso garante a padronização da comunicação entre os serviços do provedor e destes com os serviços do auditor. Porém, a implantação do auditor como um serviço na plataforma de grade do Globus Toolkit não é obrigatória, visto que o auditor não possui requisitos de grade computacional. O auditor apenas precisa estar acessível às outras partes e ter acesso ao provedor, além de saber interpretar e tratar de acordos de nível de serviço no padrão WS-Agreement.

O consumidor foi implementado em Java e em Shell Script como uma aplicação *standalone*. A criação das ofertas de acordo de serviço no consumidor, tanto para o provedor como para o auditor, são atualmente manuais. Ou seja, não foi implementado qualquer algoritmo para gerar ofertas de acordo de serviço a partir dos modelos de acordo obtidos do provedor e dos auditores. A criação de ofertas de acordo de nível de serviço é uma questão bastante específica para cada interesse e domínio de aplicação do consumidor. Muitas variáveis estão envolvidas nesse processo, tanto no âmbito financeiro como no âmbito técnico do serviço requerido. Além disso, a criação das ofertas de acordo de serviço não é o foco desse trabalho, visto que a forma como são geradas não tem impacto sobre o mecanismo de negociação de auditores.

A Figura 6.2 apresenta um diagrama de implantação que ilustra como os serviços mencionados foram implantados e como se comunicam dentro da plataforma de grade computacional.

## 6.1 Módulo Client

O cliente é uma aplicação *standalone* implementada em Shell Script e Java. Em linhas gerais, o cliente é executado através de um script chamado *wsag-negotiate-auditor*. Para recuperar os modelos de acordo dos auditores, por exemplo, a linha de comando ilustrada na Figura 6.3 deve ser executada.

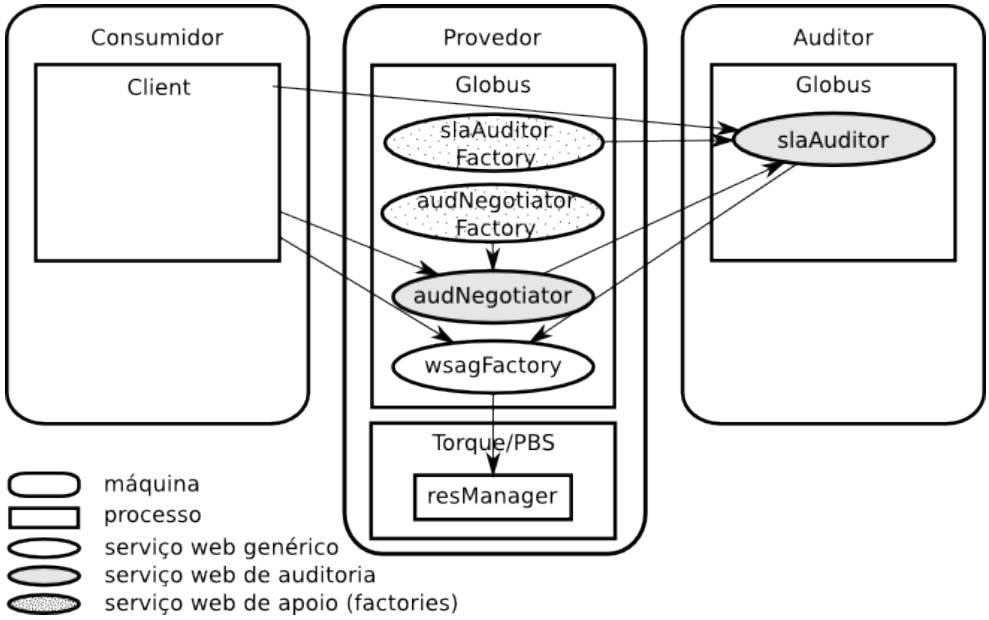


Figura 6.2: Diagrama de implantação do mecanismo de negociação de auditoria.

```
wsag-negotiate-auditor -a getAuditorTemplates
-e "/tmp/eprAuditorNegotiation.xml"
-o "/tmp/auditorEPR.xml;/tmp/auditorTemplate.xml"
```

Figura 6.3: Linha de comando que recupera modelos de acordo dos auditores.

O parâmetro *getAuditorTemplates*, identificado por *-a* na Figura 6.3, é a ação que deve ser executada. O parâmetro identificado por *-e* é o endereço do serviço *audNegotiator* armazenado em formato XML. O parâmetro identificado por *-o* especifica os nomes dos arquivos de saída da execução. Nesse caso, o primeiro nome de arquivo (/tmp/auditorEPR.xml) é onde será armazenada a referência ao auditor que enviou os modelos e o segundo nome de arquivo (/tmp/auditorTemplate.xml) é onde será armazenado o modelo de acordo retornado pelo auditor.

O script *wsag-negotiate-auditor*, então, chama a implementação Java para executar a tarefa. O código Java verifica qual é a ação a ser executada através da leitura do parâmetro *-a* e chama o método correspondente. A Figura 6.4 apresenta o código Java correspondente ao método que solicita ao provedor os modelos de acordo dos auditores.

As linhas 4 a 7 (Figura 6.4) recuperam um objeto que faz referência ao serviço *audNegotiator* a partir do arquivo XML passado no parâmetro *-e*. As linhas 9 e 10 chamam a operação *getAuditorTemplates* no serviço *audNegotiator*. As linhas 12 a 22 recuperam o conteúdo da resposta obtida do serviço *audNegotiator* e o armazena nos arquivos de saída, conforme solicitado pelo cliente através do parâmetro *-o*.

O objeto recuperado na linha 6 é do tipo *SlaAuditorNegotiationPortType*. Esse tipo, bem como todos os tipos em Java correspondentes aos definidos na descrição WSDL das interfaces dos serviços (Figura 5.4), é gerado de forma automática. Assim, antes da compilação do serviço, um script de apoio da plataforma Globus Toolkit 4.0.8 chamado *globus-build-service.sh* é executado, passando como parâmetro o endereço do arquivo que

```
1 private static void getAuditorTemplates(String eprFilesString, String outputFilesString)
2         throws FileNotFoundException, DeserializationException,
3         IOException, ServiceException, SerializationException {
4     EndpointReferenceType endpointReference = readEPRsFromFiles(eprFilesString)[0];
5
6     SlaAuditorNegotiationPortType slaAuditorNegotiationPortType =
7         createSlaAuditorNegotiationPortTypeReference(endpointReference);
8
9     GetAuditorTemplatesResponse response = slaAuditorNegotiationPortType
10            .getAuditorTemplates(new GetAuditorTemplatesRequest());
11
12    AuditorTemplateDuo[] auditorTemplateDuos = response.getAuditorTemplateDuos();
13
14    logger.debug("Auditor templates:" + auditorTemplateDuos);
15    if (auditorTemplateDuos != null && auditorTemplateDuos.length > 0) {
16        logger.debug("auditorTemplate[0]: " + auditorTemplateDuos[0]);
17    }
18
19    String[] outputFiles = outputFilesString.split(";");
20    writeSlaAuditorEPRToFile(SlaAuditorQNames.RESOURCE_REFERENCE, outputFiles[0],
21            auditorTemplateDuos);
22    writeTemplatesToFile(outputFiles[1], auditorTemplateDuos);
23 }
```

Figura 6.4: Implementação do método *getAuditorTemplates* no cliente.

contém a descrição da interface em WSDL (Figura 5.4). A partir das informações contidas nesse arquivo, o script gera classes Java correspondentes aos tipos definidos. Essas classes, então, são utilizadas para referenciar os serviços e para passar parâmetros para as operações desses serviços, como o parâmetro *GetAuditorTemplatesRequest* do método *getAuditorTemplates* (Figura 5.5).

Para implantar os serviços Web no Globus Toolkit 4.0.8, é preciso informar ao Globus quais são os serviços a serem implantados. Isso é feito através de um arquivo de configuração chamado *deploy-server.wsdd*. Esse arquivo de configuração explicita, entre outros parâmetros, qual é e onde está localizado o arquivo que define a interface de cada serviço a ser implantado (WSDL), e qual é a classe Java que implementa o serviço. Essas informações estão representadas nas linhas 4 a 6 (serviço *audNegotiator*) e 17 e 18 (serviço *slaAuditor*) do extrato do arquivo de configuração presente na Figura 6.5.

```
1 <!-- Sla Auditor Negotiation Service -->
2 <service name="br/unb/slaauditor/server/impl/SlaAuditorNegotiationService"
3         provider="Handler" use="literal" style="document">
4     <parameter name="className" value="br.unb.slaauditor.server.impl.SlaAuditorNegotiationService"/>
5     <wsdlFile>share/schema/slaauditor/
6         SlaAuditorNegotiationService_instance/SlaAuditorNegotiation_service.wsdl</wsdlFile>
7     <parameter name="allowedMethods" value="*"/>
8     <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
9     <parameter name="scope" value="Application"/>
10    <parameter name="providers" value="GetRPProvider"/>
11    <parameter name="loadOnStartup" value="true"/>
12 </service>
13
14 <!-- Sla Auditor Service -->
15 <service name="br/unb/slaauditor/server/impl/SlaAuditorService"
16         provider="Handler" use="literal" style="document">
17     <parameter name="className" value="br.unb.slaauditor.server.impl.SlaAuditorService"/>
18     <wsdlFile>share/schema/slaauditor/SlaAuditorService_instance/SlaAuditor_service.wsdl</wsdlFile>
19     <parameter name="allowedMethods" value="*"/>
20     <parameter name="securityDescriptor" value="etc/br_unb_slaauditor_server/security-config.xml"/>
21     <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
22     <parameter name="scope" value="Application"/>
23     <parameter name="providers" value="GetRPProvider GetMRPProvider SetRPProvider QueryRPProvider"/>
24     <parameter name="loadOnStartup" value="true"/>
25 </service>
```

Figura 6.5: Configuração da implantação dos serviços no Globus Toolkit 4.0.8.

## 6.2 Serviços de Apoio: Factories

Para o mecanismo funcionar adequadamente, é necessário instanciar os serviços *audNegotiator* e *slaAuditor* com os parâmetros corretos. Por exemplo, o serviço *audNegotiator* precisa saber quais são os auditores nos quais ele pode confiar e obter referências para eles. O serviço *slaAuditor*, por sua vez, precisa saber quais são os modelos de acordo de auditoria que ele pode aceitar.

Para instanciar os serviços do mecanismo de auditoria com os parâmetros certos, foram criados dois serviços de apoio adicionais. Esses serviços são o *slaAuditorFactory* e o *audNegotiatorFactory*, que servem apenas como *factories* (Gamma et al., 1995) para os auditores e para o serviço de negociação de auditores do provedor, respectivamente. O padrão de projeto *Factory* especifica uma forma de instanciar objetos de forma a diminuir o acoplamento entre o cliente (instanciador dos objetos) e os objetos instanciados.

Os serviços do tipo *factory*, portanto, são uma das formas possíveis para instanciar os serviços *slaAuditor* e *audNegotiator*. Outras formas poderiam ter sido escolhidas para instanciar esses serviços com os parâmetros necessários, porém a escolha pela criação de serviços *factory* se justifica pela diminuição do acoplamento entre instanciadores e instanciados, conforme mencionado anteriormente.

Esses serviços são apenas de apoio à execução do protótipo implementado e não possuem qualquer influência sobre o mecanismo de negociação de auditores proposto neste trabalho. Os serviços *slaAuditorFactory* e *audNegotiatorFactory* foram implantados como serviços Web na plataforma Globus Toolkit 4.0.8, assim como os demais serviços do mecanismo de negociação de auditoria, conforme apresentado na Figura 6.2.

## 6.3 Autenticação e Autorização de Consumidores

Para que o mecanismo de negociação de auditores possa ser utilizado em ambientes de grade computacional reais, é preciso garantir que os usuários sejam autenticados e autorizados a invocar as operações dos serviços Web disponíveis. Assim, nenhum usuário não identificado ou não autorizado deve poder solicitar a execução de um acordo de serviço no provedor ou no auditor.

O Globus Toolkit 4.0.8 permite trabalhar com autenticação e autorização de usuários via certificados digitais. Assim, o consumidor (Figura 5.3) precisa possuir um certificado digital válido e autorizações tanto no provedor como no auditor para invocar suas operações.

A segurança via certificado digital necessita de uma autoridade certificadora para emitir os certificados. O Globus Toolkit 4.0.8 disponibiliza uma autoridade certificadora minimalista chamada *SimpleCA*, que pode ser usada para gerar certificados digitais válidos. Através dessa autoridade certificadora, foram gerados certificados digitais válidos para consumidor, provedor e auditor.

Além disso, o provedor e o auditor foram configurados para autorizar a invocação de suas operações. A autorização de invocação de operações nos serviços Web é realizada por meio de um arquivo de configuração chamado *grid-mapfile*. Esse arquivo possui a identificação única (*Subject Name*) de cada certificado digital autorizado a invocar as operações dos serviços Web.

Uma autorização mais refinada para cada operação do serviço Web pode ser especificada através de um arquivo de configuração de segurança chamado *security-config.xml*. Através desse arquivo, foi possível configurar a operação *createAgreement* do serviço *slaAuditor* para ser executada com as credenciais do consumidor, conforme Figura 6.6. Essa configuração de segurança na operação *createAgreement* (linha 6) do serviço *slaAuditor* (linha 2) é importante para que o provedor aceite a invocação do auditor para executar o acordo (Figura 5.3) como se fosse uma invocação do próprio consumidor, já que o acordo é estabelecido entre consumidor e provedor, conforme explicado na Seção 5.3.

```
1 <securityConfig xmlns="http://www.globus.org"
2     xmlns:tns="http://slaauditor.unb.br/server/impl/SlaAuditorService_instance">
3
4 <authz value="none"/>
5
6 <method name="tns:createAgreement">
7   <auth-method>
8     <GSISecureConversation/>
9   </auth-method>
10  <run-as>
11    <caller-identity/>
12  </run-as>
13 </method>
14
15 <!-- Default for other methods -->
16 <auth-method>
17     <GSISecureConversation/>
18 </auth-method>
19
20 </securityConfig>
```

Figura 6.6: Configuração refinada de segurança do serviço *slaAuditor*.

# Capítulo 7

# Avaliação e Resultados

O mecanismo de negociação de auditores proposto no Capítulo 5 foi avaliado de forma teórica e experimental. A avaliação teórica é apresentada na Seção 7.1, onde é feita a análise da complexidade assintótica de tempo e espaço do mecanismo de negociação. A avaliação experimental do mecanismo foi realizada com o protótipo descrito no Capítulo 6 e os resultados obtidos são discutidos na Seção 7.2.

## 7.1 Avaliação Teórica

A Figura 5.3 ilustra de forma simplificada os passos do mecanismo de negociação de auditores. Para realizar a análise de complexidade, porém, foram utilizados os algoritmos 1, 2 e 3 apresentados na Seção 5.3. Esses algoritmos apresentam detalhes importantes para a realização da análise de complexidade.

O objetivo da análise de complexidade do mecanismo de negociação de auditores é avaliar a complexidade da solução em relação a uma execução de serviço sem negociação de auditores (Figura 5.2). Assim, foram analisadas as complexidades das linhas do Algoritmo 1 correspondentes ao mecanismo de negociação de auditores (região cinza da Figura 5.3). É preciso ressaltar também que a análise de complexidade não tem o objetivo de avaliar a monitoração do acordo realizada pelo auditor através do método $monitor(agreement)$ (passo 13 da Figura 5.3). Embora o protótipo utilizado para a avaliação experimental realize uma monitoração simples do acordo, a forma como o auditor realiza a monitoração é independente do mecanismo de negociação e varia de auditor para auditor, não fazendo parte, portanto, dessa avaliação de complexidade.

A análise de complexidade também viabiliza a verificação da escalabilidade do mecanismo de negociação. Para isso, a análise de complexidade foi realizada de forma a identificar como variam os requisitos de tempo e de espaço do mecanismo em relação ao número de auditores de confiança do consumidor e do provedor. A pergunta que se deseja responder é quanto o aumento ou a diminuição do número de auditores de confiança do consumidor e do provedor influencia o tempo e o espaço requeridos pelo mecanismo.

As seções 7.1.1 e 7.1.2 apresentam as análises de complexidade de tempo e de espaço do mecanismo, respectivamente, em relação ao número de auditores de confiança do consumidor e do provedor.

### 7.1.1 Complexidade de Tempo

Conforme mencionado anteriormente, a análise assintótica de complexidade de tempo foi realizada com base nos algoritmos 1, 2 e 3 apresentados na Seção 5.3. A análise realizada é uma análise de pior caso em relação aos conjuntos de auditores de confiança do consumidor e do provedor.

O Algoritmo 1 foi analisado da linha 3 à linha 11, uma vez que as demais linhas são executadas mesmo quando não se realiza a negociação do auditor (Figura 5.2). Para essa análise, são considerados:

- o conjunto $P$ de auditores confiáveis do provedor;
- o conjunto $C$ de auditores confiáveis do consumidor; e
- o conjunto $T$ de modelos de acordo disponibilizados por todos os auditores.

A linha 3 do Algoritmo 1 chama o método *getAuditorTemplates* do provedor. A descrição desse método é apresentada no Algoritmo 2. De acordo com o Algoritmo 2, o método *getAuditorTemplates* executa um laço que passa por cada auditor de confiança do provedor (conjunto $P$), e solicita seus respectivos modelos de acordo de auditoria disponíveis. O método não recebe qualquer parâmetro do consumidor e retorna um registro que contém cada auditor de confiança do provedor juntamente com seus respectivos modelos de acordo de auditoria. A análise assintótica de complexidade de tempo desse método, portanto, é $O(|P|)$, onde $|P|$ é a cardinalidade do conjunto $P$ de auditores confiáveis do provedor.

A linha 4 do Algoritmo 1 é apenas uma atribuição de valor à variável $i$ e não depende dos conjuntos de auditores de confiança do provedor ou do consumidor.

A linha 5, por sua vez, apresenta um laço que itera pelos elementos do conjunto de registros de auditores e modelos recebido do provedor na linha 3. Esse laço também tem uma condição de saída que depende do valor da variável *accepted*. Essa variável booleana muda de estado na linha 8 quando uma oferta de auditoria é validada pelo provedor e aceita por algum dos auditores de confiança tanto do provedor como do consumidor. No pior caso, porém, esse laço também é $O(|P|)$, visto que o conjunto *audTemp* possui tamanho máximo igual ao conjunto de auditores de confiança do provedor ($P$).

A linha 6 do Algoritmo 1 possui uma condição para a execução das linhas 7 e 8. A condição depende de uma avaliação de pertinência sobre o conjunto $C$ de auditores de confiança do consumidor. A avaliação verifica se o auditor corrente da iteração sobre os auditores de confiança do provedor também pertence ao conjunto de auditores de confiança do consumidor. A análise de complexidade de tempo dessa avaliação depende do tipo de estrutura de dados que foi utilizado para implementar o conjunto $C$ e do tipo de busca que será feita nessa estrutura. No pior caso, o conjunto $C$ é implementado através de uma lista encadeada simples e desordenada e a busca será sequencial. Isso implica que será necessário passar por cada auditor de confiança do auditor para verificar a pertinência do auditor corrente a esse conjunto. Nesse caso, a complexidade assintótica de tempo da busca é $O(|C|)$.

A execução das linhas 7 e 8 do Algoritmo 1 depende da cardinalidade do conjunto interseção entre $C$ e $P$, visto que só serão executadas caso existam auditores de confiança tanto do consumidor como do provedor. Na linha 7, o consumidor cria uma oferta de

auditoria com base no auditor corrente e na oferta do serviço a ser executado. Conforme mencionado no Capítulo 5, a criação da oferta de auditoria não é automática, visto que depende da necessidade e da estratégia do consumidor para obter o serviço desejado. Além disso, essa etapa também é muito dependente do domínio de aplicação em questão e da forma como foram construídos os modelos de acordo de auditoria disponibilizados por cada auditor. De qualquer forma, a criação da oferta de auditoria não depende do número de auditores de confiança do provedor ou do consumidor e, por isso, a complexidade de tempo dessa linha é $O(1)$.

Na linha 8, o consumidor solicita ao provedor que seja criado um acordo de auditoria com o auditor corrente, baseado na oferta de auditoria criada na linha 7. Para isso, é chamado o método *createAuditorAgreement* do provedor. Esse método é apresentado no Algoritmo 3. O método *createAuditorAgreement* recebe como parâmetros a oferta do serviço a ser executado, a oferta de auditoria criada na linha 7 e o auditor corrente escolhido para realizar a auditoria. Na linha 1 do Algoritmo 3, o provedor valida a oferta de auditoria com base na oferta do serviço a ser realizado. Essa validação depende fortemente dos serviços e restrições apresentados pelo provedor e pelo auditor nos respectivos modelos de acordo. Porém, essa validação independe dos conjuntos de auditores de confiança do provedor e do consumidor ($P$ e $C$, respectivamente). A linha 2 possui uma condição para a execução das linhas 3 e 4 que é baseada no valor da variável *accepted*, que foi atribuído na linha 1 de acordo com a validade da oferta de auditoria. Caso a oferta de auditoria seja válida, a execução da linha 3 irá solicitar ao auditor selecionado que seja criado um acordo de auditoria com base na oferta recebida. Nesse caso, a linha 4 também será executada, o que resultará no armazenamento da oferta de auditoria pelo provedor. Assim como a linha 1, as linhas de 2 a 5 do Algoritmo 3 não dependem do número de auditores de confiança do provedor ou do consumidor. A linha 6 apenas retorna o valor da variável *accepted*, que indica se a oferta foi validada pelo provedor e se o acordo de auditoria foi criado no auditor. Portanto, a complexidade assintótica de tempo do Algoritmo 3 em relação a $P$ e a $C$ é $O(1)$.

Por fim, a linha 10 do Algoritmo 1 apenas incrementa o valor da variável $i$ e, portanto, tem complexidade $O(1)$.

Com base na análise de complexidade de tempo realizada em cada linha dos algoritmos 1, 2 e 3, chega-se à conclusão de que a complexidade assintótica de tempo do mecanismo de negociação de auditoria é $O(|P| * |C|)$.

### 7.1.2 Complexidade de Espaço

A análise de complexidade de espaço do mecanismo também foi uma análise de pior caso baseada nos algoritmos 1, 2 e 3 apresentados na Seção 5.3. Nesse caso, porém, além da influência dos conjuntos $P$ e $C$ de auditores confiáveis, também foi considerada a influência do número de modelos de acordo presentes no sistema. Os modelos de acordo de auditoria precisam ser armazenados e transferidos entre auditor, provedor e consumidor e, por isso, não podem ser desprezados na análise de complexidade de espaço. Os modelos de acordo do provedor, porém, podem ser desprezados, visto que eles não fazem parte do mecanismo de negociação de auditoria, estando presentes também quando não se realiza esse tipo de negociação (Figura 5.2). O conjunto de todos os modelos de acordo disponibilizados pelos auditores é o conjunto $T$ na análise de complexidade de espaço apresentada a seguir. Pelo

mesmo motivo apresentado na Seção 7.1.1, as linhas 1, 2, 12 e 13 do Algoritmo 1 também não foram consideradas nessa análise.

A linha 3 do Algoritmo 1 executa o método *getAuditorTemplates* do provedor, detalhado no Algoritmo 2. Esse método depende do conjunto de auditores de confiança do provedor ($P$) e também armazena todos os modelos de acordo recebidos de cada um desses auditores. Logo, a complexidade assintótica de espaço desse método é $O(|P|+|T|)$. Note que esse método é executado no provedor de serviços. Veremos a seguir que essa é a parcela mais significativa da complexidade de espaço do provedor de serviços. A outra parcela significativa da complexidade de espaço do mecanismo de negociação está presente no consumidor de serviços.

A linha 3 do Algoritmo 1 também armazena o retorno do método *getAuditorTemplates* na variável *audTemp* do consumidor. Esse armazenamento requer um espaço em memória também da ordem de $O(|P| + |T|)$ no consumidor.

As linhas 4 e 5 do Algoritmo 1 não influenciam a análise assintótica de complexidade de espaço do mecanismo, visto que os únicos armazenamentos extras requeridos são os valores das variáveis $i$ e *accepted*. Esses armazenamentos, porém, são indepedentes dos conjuntos $P$, $C$ e $T$ e, portanto, o espaço necessário para executar essas linhas é da ordem de $O(1)$.

A linha 6, porém, executa uma pesquisa sobre os auditores de confiança do consumidor ($C$), verificando se o auditor corrente está presente nesse conjunto. Essa linha, portanto, evidencia a dependência do consumidor em relação ao conjunto $C$. A pesquisa no conjunto $C$, porém, pode ser realizada sem a necessidade de qualquer outro armazenamento de informações. Logo, a linha 6 possui complexidade assintótica de espaço da ordem de $O(|C|)$.

A linha 7 do Algoritmo 1 representa a criação manual de uma oferta de auditoria. Por ser manual, a criação da oferta não será contabilizada na análise de espaço. Embora seja necessário o armazenamento da oferta criada, esse armazenamento tem influência apenas da ordem de $O(1)$ sobre a análise de complexidade de espaço do mecanismo.

A linha 8 executa o método *createAuditorAgreement* do provedor, que é detalhado no Algoritmo 3. A linha 1 do Algoritmo 3 executa a validação da oferta de auditoria com base em três informações: a oferta de serviços, a oferta de auditoria e o auditor selecionado. Além disso, o resultado da validação é armazenado em uma variável booleana chamada *accepted*. O espaço requerido para armazenar essas quatro informações independe das cardinalidades dos conjuntos $P$, $C$ e $T$ e, portanto, a linha 1 do Algoritmo 3 contribui apenas com uma constante para a análise de complexidade de espaço do mecanismo. A linha 2 do Algoritmo 3 apenas verifica o valor da variável *accepted* e, por isso, não influencia a análise de espaço. A linha 3 apenas muda o valor da variável *accepted* de acordo com a aceitação da oferta de auditoria pelo auditor; e a linha 4 apenas armazena a oferta de auditoria no provedor de serviços, o que contribui com uma constante para a análise de complexidade de espaço. No total, portanto, o método *createAuditorAgreement* do provedor contribui apenas com constantes para a análise de espaço do mecanismo de negociação.

A linha 8 do Algoritmo 1 também armazena o resultado da execução do método *createAuditorAgreement* na variável *accepted*. Esse armazenamento também requer espaço da ordem de $O(1)$, visto que não depende das cardinalidades dos conjuntos $P$, $C$ e $T$.

Por fim, a linha 10 do Algoritmo 1 apenas incrementa o valor da variável $i$ e, portanto, não influencia a complexidade de espaço do mecanismo.

Com base na análise apresentada para os algoritmos 1, 2 e 3, o espaço requerido pelo mecanismo, portanto, é da ordem de $O(|P| + |T|)$ no provedor e de $O(|P| + |C| + |T|)$ no consumidor.

## 7.2 Avaliação Experimental

Para testar o mecanismo de negociação de auditoria proposto, o protótipo implementado foi experimentado em uma grade computacional com quatro desktops do Laboratório de Pós-Graduação do Departamento de Ciência da Computação da Universidade de Brasília e um netbook. Os quatro desktops usados nos experimentos possuem todos a mesma configuração de hardware, que corresponde a um processador Intel Core 2 Duo E7200 de 2.53GHz, um pente de memória RAM de 2GB, placa-mãe Digitron IPM31 versão 1.01G, placa de rede Realtek RTL8111/8168B conectada a 100MB/s e disco rígido Samsung HD161HJ com sistema de arquivos ext4. A configuração de software também é idêntica em todos os desktops, correspondendo ao sistema operacional Ubuntu Desktop 10.10 e o conjunto de sistemas descritos no Capítulo 6. O netbook utilizado nos experimentos possui um processador Intel ULV Pentium SU4100 de 1.30GHz, um pente de memória RAM de 2GB, placa-mãe Dell 0MGWHR versão A04, placa de rede Atheros AR8132 Gigabit Ethernet Adapter conectada a 100MB/s e disco rígido WDC WD3200BEVT-7 com sistema de arquivos ext4. A configuração de software do netbook corresponde ao sistema operacional Ubuntu Desktop 10.04 e o conjunto de sistemas descritos no Capítulo 6.

Os experimentos foram executados em cinco cenários distintos sendo que, em cada um deles, o netbook foi utilizado como consumidor, um dos desktops foi usado como provedor de serviços e cada um dos outros três desktops foi utilizado para hospedar um auditor distinto. Para cada cenário distinto, foi mensurado o tempo do início ao fim da execução e foi verificada a corretude do resultado obtido. Como o espaço ocupado pelos modelos de acordo é muito pequeno, da ordem de menos de 5KB por modelo, não foram realizados experimentos relacionados à avaliação do espaço requerido pelo mecanismo de negociação. A Tabela 7.1 apresenta cada um dos cenários de avaliação experimental executados.

Com exceção do cenário 2, em que não há execução de serviço, todos os demais cenários utilizaram as mesmas ofertas de serviço e de auditoria nos casos de sucesso (oferta padrão). Ou seja, as ofertas que foram aceitas tanto pelo provedor como pelo auditor e que, portanto, resultaram na execução do serviço, são sempre idênticas em todos os cenários. Essa restrição é importante para que os tempos de execução dos cenários possam ser comparados entre si. Se os serviços executados fossem diferentes entre os cenários, não seria possível comparar os seus tempos de execução.

O serviço especificado na oferta padrão de serviço é a execução de uma aplicação chamada *testtouch*, cujo resultado é apenas a criação de um arquivo vazio. O serviço de auditoria é o monitoramento do tempo de resposta requerido para a execução desse serviço, que é especificado em 8 segundos tanto na oferta de serviço como na oferta de auditoria padrão. Nos casos em que não há sucesso na aceitação da oferta, seja pelo provedor ou pelo auditor, é utilizada uma oferta diferente da padrão.

Tabela 7.1: Cenários de avaliação experimental do protótipo do mecanismo de negociação de auditoria.

| Cenário | Descrição |
|---|---|
| Cenário 1 | O primeiro cenário experimentado diz respeito a uma execução de serviço sem a utilização do mecanismo de negociação de auditor proposto. |
| Cenário 2 | O segundo cenário representa um caso em que a lista de auditores de confiança do provedor não possui interseção com a lista de auditores de confiança do consumidor. Nesse cenário, portanto, não há ofertas para estabelecimento de auditor externo e não há execução de serviço. |
| Cenário 3 | O terceiro cenário é um caso em que o único auditor de confiança de ambas as partes é negociado e estabelecido com sucesso. |
| Cenário 4 | No quarto cenário, o primeiro auditor nega a oferta de auditoria e o segundo auditor é negociado e estabelecido com sucesso. |
| Cenário 5 | No quinto e último cenário, a primeira oferta de auditoria é negada pelo provedor, a segunda oferta é negada pelo auditor, e a terceira oferta é aceita pelo provedor e pelo auditor. |

As seções a seguir detalham a configuração e a execução de cada um dos cenários experimentados com o protótipo do mecanismo de negociação de auditoria.

### 7.2.1 Cenário 1 - Sem Mecanismo de Negociação de Auditor

O cenário de avaliação 1 representa uma execução de serviço sem a utilização do mecanismo de negociação de auditoria proposto neste trabalho. Ou seja, o cenário 1 representa exatamente a interação apresentada na Figura 5.2, em que o serviço é executado mas não é realizada a negociação de um auditor externo para avaliar a qualidade do serviço prestado. Nesse cenário, portanto, nem o consumidor nem o provedor possuem uma lista de auditores de confiança.

Assim como nos demais cenários de avaliação apresentados a seguir, nesse cenário o consumidor foi executado no netbook e o provedor de serviços foi executado em um dos desktops. O consumidor solicita os modelos de acordo ao provedor, cria uma oferta padrão baseada em um dos modelos recebidos e a envia para execução no provedor de serviços. Assim, o serviço é executado pelo provedor sem que seja realizada a auditoria externa de qualidade. Como não há auditores nesse cenário, as demais máquinas não são utilizadas. O serviço executado é a aplicação *testtouch*, conforme definido na oferta padrão.

### 7.2.2 Cenário 2 - Sem Sucesso na Negociação

O cenário de avaliação 2 representa o caso em que não é possível estabelecer um auditor externo por não existir interseção entre a lista de auditores de confiança do provedor e a lista de auditores de confiança do consumidor. Como é premissa do mecanismo de negociação de auditoria que o auditor escolhido seja de confiança de ambas as partes (provedor e consumidor), não é possível o estabelecimento do auditor nesse cenário.

Sejam $P$ a lista de auditores de confiança do provedor e $C$ a lista de auditores de confiança do consumidor e sejam $a1$, $a2$ e $a3$ auditores distintos, nesse cenário $P = \{a1, a2\}$ e $C = \{a3\}$. Ou seja, o provedor confia nos auditores $a1$ e $a2$, enquanto o consumidor confia apenas no auditor $a3$. Note que $P \cap C = \emptyset$.

Nesse cenário, o consumidor executa as linhas de 1 a 6 do Algoritmo 1. Porém, como não há interseção entre $P$ e $C$, o consumidor nunca chega a criar e enviar uma oferta de auditoria baseada em um dos modelos recebidos dos auditores $a1$ e $a2$ (linhas 7 e 8 do Algoritmo 1), que são de confiança apenas do provedor. Como não há envio e aceitação de oferta de auditoria, o consumidor também não executa a linha 13 do Algoritmo 1. Ou seja, nesse cenário não há execução de serviço pelo provedor, devido ao não estabelecimento do auditor de confiança entre as partes.

Assim como em todos os cenários de avaliação, o consumidor foi executado no netbook e o provedor foi executado em um dos desktops da grade, o qual disponibilizou os serviços *wsagFactory* e *audNegotiator*. Cada auditor ($a1$, $a2$ e $a3$), por sua vez, foi hospedado em um dos outros três desktops, os quais disponibilizaram o serviço *slaAuditor*.

### 7.2.3 Cenário 3 - Sucesso com 1 Auditor

O cenário de avaliação 3 representa um caso simples de sucesso na negociação e estabelecimento do auditor externo. Nesse cenário, o provedor confia em três auditores ($a1$, $a2$ e $a3$), enquanto o consumidor confia apenas no auditor $a3$. Portanto, o auditor $a3$ é o único auditor de confiança tanto do provedor como do consumidor.

Formalmente, sejam $P$ a lista de auditores de confiança do provedor e $C$ a lista de auditores de confiança do consumidor e sejam $a1$, $a2$ e $a3$ auditores distintos, nesse cenário $P = \{a1, a2, a3\}$ e $C = \{a3\}$. Logo, $P \cap C = \{a3\}$.

O consumidor executa todas as linhas do Algoritmo 1, recuperando os modelos de acordo dos auditores $a1$, $a2$ e $a3$ na linha 3 e enviando uma oferta de auditoria padrão para o auditor $a3$ através do provedor na linha 8. Essa oferta de auditoria padrão é validada e aceita tanto pelo provedor como pelo auditor $a3$. A aceitação da oferta padrão resulta na execução do serviço pelo provedor, conforme especificado na linha 13 do Algoritmo 1.

### 7.2.4 Cenário 4 - Sucesso com 1 Falha em Auditor

O cenário de avaliação 4 representa uma situação em que existem dois auditores de confiança do provedor e do consumidor, porém a oferta enviada ao primeiro auditor de confiança não é aceita pelo auditor. Em seguida, é enviada uma oferta padrão ao segundo auditor, a qual é aceita e o serviço é executado pelo provedor. Esse cenário, portanto, é um caso em que ocorre uma negação de oferta por um dos auditores durante a negociação do auditor externo. Diferentemente do cenário de avaliação 2, este é um cenário de sucesso, uma vez que o auditor externo é estabelecido entre as partes e o serviço é executado pelo provedor.

Formalmente, sejam $P$ a lista de auditores de confiança do provedor e $C$ a lista de auditores de confiança do consumidor e sejam $a1$, $a2$ e $a3$ auditores distintos, nesse cenário $P = \{a1, a2, a3\}$ e $C = \{a2, a3\}$. Logo, $P \cap C = \{a2, a3\}$.

O consumidor recupera os modelos de acordo dos auditores $a2$ e $a3$ na linha 3 do Algoritmo 1. Na primeira iteração do laço, é criada uma oferta não-padrão baseada em um modelo de acordo do auditor $a2$ na linha 7. Essa oferta não-padrão possui propositadamente um prazo de validade expirado, ou seja, possui data de validade anterior à data do experimento. Essa oferta é enviada ao auditor $a2$ através do provedor na linha 8. Ao receber a oferta, o auditor executa a validação e verifica que a oferta está expirada. Nesse caso, então, a oferta é negada pelo auditor e o consumidor é obrigado a tentar uma

nova oferta com o próximo auditor de confiança de ambas as partes, que é o auditor $a3$. O consumidor, então, avança no laço da linha 5 do Algoritmo 1, cria uma oferta padrão na linha 7 e, na linha 8, a envia ao auditor $a3$ através do provedor. Ao receber a oferta padrão de auditoria, o auditor $a3$ a aceita e o consumidor, então, solicita a execução do serviço pelo provedor na linha 13 do Algoritmo 1. O serviço é executado com sucesso pelo provedor e, assim, a execução do experimento é finalizada.

### 7.2.5 Cenário 5 - Sucesso com 2 Falhas (Provedor e Auditor)

O cenário de avaliação 5 representa um caso em que existem três auditores de confiança tanto do provedor como do consumidor, porém a primeira oferta de auditoria é negada pelo provedor, a segunda oferta é negada pelo auditor e a terceira oferta é uma oferta padrão aceita para execução pelo terceiro e último auditor. Esse cenário, portanto, é um cenário de sucesso similar ao cenário de avaliação 4, com a diferença de que há um auditor a mais de confiança de ambas as partes e que este novo auditor não chega a receber sua oferta de auditoria porque esta é negada anteriormente pelo próprio provedor.

Formalmente, sejam $P$ a lista de auditores de confiança do provedor e $C$ a lista de auditores de confiança do consumidor e sejam $a1$, $a2$ e $a3$ auditores distintos, nesse cenário $P = \{a1, a2, a3\}$ e $C = \{a1, a2, a3\}$. Logo, $P = C$.

O consumidor executa as quatro primeiras linhas do Algoritmo 1, recuperando os modelos de acordo dos auditores $a1$, $a2$ e $a3$ na linha 3. O laço da linha 5 é executado, iterando sobre todos os três auditores de confiança do provedor, que também são de confiança do consumidor. Na primeira iteração do laço, o consumidor cria uma oferta não-padrão de auditoria para o auditor $a1$ na linha 7 e a envia através do provedor na linha 8. Essa oferta possui propositadamente um tempo de resposta igual 6 segundos, diferentemente do tempo de resposta especificado na oferta de serviço, que é igual a 8 segundos. O provedor, ao receber essa oferta, executa sua validação e verifica a discrepância entre os tempos de resposta do serviço a ser executado pelo provedor e da auditoria a ser realizada pelo auditor. A oferta é negada pelo provedor e o consumidor, então, avança para o próximo auditor de confiança, que é o auditor $a2$. Na segunda iteração da linha 7, o consumidor cria uma oferta idêntica àquela enviada ao auditor $a2$ no cenário de avaliação 4. Ou seja, ela possui um tempo de validade expirado. Essa oferta é recebida pelo auditor $a2$, através do provedor, e é negada. O consumidor, então, avança para o próximo auditor de confiança no laço da linha 5 do Algoritmo 1. Na terceira iteração da linha 7, o consumidor cria uma oferta padrão a ser enviada para o auditor $a3$ na linha 8. O auditor recebe e aceita essa oferta, resultando na execução do serviço pelo provedor, conforme especificado na linha 13.

### 7.2.6 Comparação de Resultados

Quanto à execução do algoritmo, todos os cenários de avaliação experimental do protótipo do mecanismo de negociação de auditoria funcionaram de acordo com o esperado. A Tabela 7.2 apresenta um resumo dos resultados obtidos em cada um dos cenários experimentais avaliados, contendo o número de execuções de cada cenário e o tempo médio de execução.

Tabela 7.2: Resumo de resultados dos cenários de avaliação experimental.

| Cenário | Execuções | Tempo Médio |
|---|---|---|
| Cenário 1 | 20 vezes | 23,10s |
| Cenário 2 | 20 vezes | 13,55s |
| Cenário 3 | 20 vezes | 59,25s |
| Cenário 4 | 20 vezes | 62,95s |
| Cenário 5 | 20 vezes | 69,65s |

No cenário 1, não há utilização do mecanismo de negociação de auditoria. O tempo médio de execução desse cenário foi de 23,10 segundos, o que corresponde a todos os passos apresentados na Figura 5.2, incluindo recuperar os modelos de acordo do provedor, criar a oferta, enviá-la ao provedor e finalmente executar o serviço. O serviço executado corresponde à aplicação *testtouch*, que apenas cria um arquivo, conforme descrito na Seção 7.2.1. Esse tempo apenas serve como referência para comparar com os demais cenários, nos quais é utilizado o mecanismo de negociação de auditoria.

Conforme descrito na Seção 7.2.2, não há execução do serviço no cenário 2 devido à inexistência de um auditor de confiança tanto do provedor como do consumidor. Contudo, os modelos de acordo ainda são recuperados do provedor e uma oferta de acordo também é criada. Além disso, os modelos de acordo dos dois auditores de confiança do provedor são recuperados. Apesar dos passos adicionais de recuperação de modelos e do tempo requerido para verificar que não existe auditor de confiança de ambas as partes, o tempo médio de execução do cenário 2 foi de apenas 13,55 segundos, ou seja, 9,55 segundos menor do que o tempo médio de execução do cenário de avaliação 1. Isto acontece porque não há execução do serviço pelo provedor no cenário 2.

No cenário 3, o único auditor de confiança de ambas as partes é negociado e estabelecido entre o consumidor e o provedor, e o provedor executa o serviço com sucesso. O tempo médio de execução desse cenário foi de 59,25 segundos, o que representa aproximadamente três vezes o tempo de execução do cenário 1. Embora possa parecer muito tempo para uma negociação envolvendo três auditores, o serviço *testtouch* executado nesses cenários é muito mais simples do que os casos de uso reais de grades computacionais. Normalmente, as aplicações de grades computacionais levam várias horas, até mesmo dias, para terminar. Isto significa que uma negociação de auditor que leva menos de 1 minuto para finalizar e ocorre apenas uma vez em cada interação entre provedor e consumidor é praticamente insignificante.

Os cenários 4 e 5 também terminam em uma negociação de auditor bem sucedida e com subsequente execução do serviço *testtouch*. Contudo, durante o processo de negociação desses cenários, alguns acordos de nível de serviço são rejeitados. No cenário 4, um acordo de nível de serviço é rejeitado pelo auditor $a2$ e, no cenário 5, um acordo é rejeitado pelo provedor de serviços e outro acordo é rejeitado pelo auditor $a2$. Apesar dessas falhas durante a negociação do auditor, o tempo médio de execução desses cenários aumentou pouco quando comparado com o aumento de 36,15 segundos observado do cenário 1 ao cenário 3. Assim, embora o processo de negociação seja mais complexo nos cenários 4 e 5, o tempo de execução foi apenas 6% a 16% maior do que o tempo de execução de um cenário de negociação simples (cenário 3).

Pode-se concluir que a maior parte do aumento de tempo do cenário 1 ao cenário 3

está relacionada com a recuperação dos modelos de acordo dos auditores. Isto significa que um aumento no número de auditores na negociação pode causar mais impacto do que um aumento no número de rejeições durante o processo de negociação. Contudo, a análise assintótica do algoritmo do mecanismo de negociação, apresentada na Seção 7.1, mostra que o tempo de execução varia linearmente em relação ao número de auditores na negociação. Além disso, acredita-se que, em casos de uso reais de grades computacionais, o número de auditores de confiança de ambas as partes deve ser muito pequeno, assim como nos cenários experimentais avaliados.

# Capítulo 8

# Conclusões e Trabalhos Futuros

Este trabalho apresentou e avaliou um mecanismo para negociação e estabelecimento de um auditor de qualidade de serviço entre o provedor e o consumidor de serviços em plataformas de grade computacional. O auditor negociado pelo mecanismo é uma entidade externa de confiança tanto do provedor como do consumidor de serviços que, ao ser estabelecido, participa de todas as requisições de serviço entre as partes. O auditor externo, portanto, funciona como um mediador que verifica a conformidade das requisições do consumidor e do serviço fornecido pelo provedor em relação a um acordo de nível de serviço previamente estabelecido.

O mecanismo proposto possui baixa intrusividade nos ambientes do provedor e do consumidor de serviços, podendo ser ativado e desativado a qualquer instante, sem a necessidade de modificação dos serviços já existentes e disponíveis. O mecanismo apenas adiciona novas funcionalidades e as disponibiliza, de forma transparente, através de serviços Web no ambiente do provedor. Além disso, o mecanismo se baseia no WS-Agreement (Andrieux et al., 2007), que é um padrão de mercado aberto para especificação e estabelecimento de acordos de nível de serviço. A utilização do padrão WS-Agreement - amplamente adotado para dar suporte a qualidade de serviço em ambientes de grade computacional - facilita a adoção e a evolução do mecanismo de negociação proposto.

O mecanismo de negociação de auditoria proposto foi avaliado de forma teórica através de análise assintótica das complexidades de tempo e de espaço dos seus algoritmos. Essas análises mostraram variações lineares de tempo e de espaço em relação à quantidade de auditores confiáveis do provedor e do consumidor e em relação ao número de modelos de acordo de serviço disponibilizados pelos auditores envolvidos na negociação. Além disso, acredita-se que, em um cenário real, a quantidade de auditores de confiança do provedor e do consumidor será muito pequena (em torno de 3 auditores, por exemplo) e que, portanto, a taxa de variação linear pode ser considerada satisfatória para a maioria dos casos.

Além disso, o mecanismo foi avaliado de forma experimental através da implementação e execução de um protótipo. O protótipo foi desenvolvido em Java, Web Services Description Language e Shell Script. A plataforma de grade computacional utilizada no protótipo foi o Globus Toolkit (Foster, 2005). O Globus Toolkit foi complementado com um conjunto de serviços Web que fornecem suporte ao padrão WS-Agreement - o SLA4D-Grid Negotiation Manager (sla, 2011). O protótipo do mecanismo foi implantado e testado em uma grade computacional com quatro desktops do Laboratório de Pós-Graduação do

Departamento de Ciência da Computação da Universidade de Brasília e um netbook. Em cada um dos cinco cenários experimentados, o protótipo foi avaliado quanto ao resultado e ao tempo de execução. Os cenários estudados contemplaram tanto casos de falha total da negociação, como falhas parciais e casos de sucesso, sendo que o mecanismo de negociação funcionou conforme esperado em cada um deles.

No cenário de avaliação 5 (Seção 7.2.5), por exemplo, foi avaliada uma situação em que três auditores participam da negociação para estabelecimento do auditor externo. Porém, as ofertas realizadas para os dois primeiros auditores são negadas pelo provedor e pelo auditor, respectivamente. Assim, apenas a terceira oferta, realizada para o último auditor, é aceita e executada com sucesso. Esse cenário foi executado em média em 69,65 segundos, o que representa aproximadamente três vezes o tempo de execução do mesmo serviço sem a utilização do mecanismo de auditoria (apresentado no cenário 1, Seção 7.2.1). Apesar disso, o tempo é considerado satisfatório, visto que o serviço executado em 23,10 segundos no cenário 1 é um serviço muito simples de criação de um arquivo vazio. Em uma situação real, porém, o serviço a ser executado pode demorar várias horas ou mesmo dias e, assim, o tempo da negociação se torna irrelevante em comparação com o tempo de execução do serviço.

O código do protótipo e sua documentação estão disponíveis publicamente no endereço http://code.google.com/p/sla-auditor-negotiator/downloads/list.

Para trabalhos futuros, sugere-se a inclusão de um algoritmo heurístico para a criação das ofertas de serviço. Atualmente no mecanismo proposto, a criação das ofertas de serviço é manual, requerendo a intervenção humana no processo de negociação. A inclusão de um algoritmo heurístico viabiliza a automação de toda a negociação do auditor, inclusive possibilitando uma negociação *multi-round* automática, em que o algoritmo poderia fazer ajustes e submeter novas ofertas de serviço ao mesmo auditor em caso de rejeição.

Sugere-se também como trabalho futuro, a extensão do mecanismo proposto para lidar com casos em que o serviço requerido pelo consumidor deve ser fornecido por vários provedores em cooperação. Nesses casos, poderia ser necessário selecionar mais de um auditor, e cada auditor escolhido poderia avaliar a qualidade de serviço de mais de um provedor simultaneamente. Poderia ser necessário também haver troca de informações entre auditores, a fim de avaliar uma qualidade de serviço fim-a-fim para o consumidor, ignorando violações intermediárias do acordo que sejam irrelevantes.

Outro trabalho futuro sugerido é a introdução de um catálogo de auditores a partir do qual consumidores e provedores possam conhecer os diferentes tipos de auditores disponíveis e, com isso, selecionar os auditores que irão compor as listas de auditores de confiança de cada parte. O catálogo poderia disponibilizar informações de um modelo de reputação para auditores, a fim de que consumidores e provedores pudessem selecionar os auditores com melhor reputação.

Além disso, sugere-se a extensão do mecanismo proposto para tratar possíveis casos de renegociação de auditoria. Ou seja, caso um auditor não seja mais capaz de prover o serviço de auditoria acordado, pode ser necessário interromper o serviço fornecido pelo provedor até que se negocie e se estabeleça um novo auditor. Esse comportamento poderia ser configurado através de critérios em uma política de auditoria previamente estabelecida em acordo de nível de serviço.

Por fim, outra sugestão de trabalho futuro é a adaptação desse mecanismo para sua

utilização em ambientes de computação em nuvem, nos quais devem ser considerados aspectos como a elasticidade do conjunto de recursos que fornecem o serviço a ser auditado.

# Referências


Internet engineering task force (ietf). http://www.ietf.org/, April 2010.

International telecommunication union (itu). http://www.itu.int, April 2010.

Open grid forum. http://www.ogf.org, April 2010.

Sla4d-grid. http://www.sla4d-grid.de/, March 2011.

Torque/pbs resource manager. http://www.clusterresources.com/products/torque-resource-manager.php, March 2011.

Kaizar Amin, Gregor von Laszewski, Mihael Hategan, Rashid Al-Ali, Omer Rana, e David Walker. An abstraction model for a grid execution framework. *J. Syst. Archit.*, 52(2): 73–87, 2006. ISSN 1383-7621. doi: http://dx.doi.org/10.1016/j.sysarc.2004.10.007.

Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, e Ming Xu. Web services agreement specification. http://www.ogf.org/documents/GFD.107.pdf, March 2007.

Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, e Andreas Savva. Job submission description language (jsdl) specification, version 1.0. http://www.gridforum.org/documents/GFD.56.pdf, November 2005.

Ana Carolina Barbosa, Jacques Sauvé, Walfredo Cirne, e Mirna Carelli. Evaluating architectures for independently auditing service level agreements. *Future Gener. Comput. Syst.*, 22(7):721–731, 2006. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2006.01.001.

P. Bianco, G. Lewis, e P. Merson. Service level agreements in service-oriented architecture environments. Technical report, Software Engineering Institute, 2008.

S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, e W. Weiss. An architecture for differentiated services (diffserv). http://tools.ietf.org/html/rfc2475, December 1998.

Ed. R. Braden, L. Zhang, S. Berson, S. Herzog, e S. Jamin. Resource reservation protocol (rsvp). http://tools.ietf.org/html/rfc2205, September 1997.

Lars-Olof Burchard, Barry Linnert, Felix Heine, Matthias Hovestadt, Odej Kao, e Axel Keller. A quality-of-service architecture for future grid computing applications. In

*IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2*, page 132.1, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2312-9. doi: http://dx.doi.org/10.1109/IPDPS.2005.62.

K. Mani Chandy e Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/214451.214456.

Adrian Ching, Lionel Sacks, e Paul McKee. Sla management and resource modelling for grid computing. In *Proceedings of London Communications Symposium 2003*, 2003.

Erik Christensen, Francisco Curbera, Greg Meredith, e Sanjiva Weerawarana. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl, March 2001.

Walfredo Cirne, Daniel Paranhos, Lauro Costa, Elizeu Santos-Neto, Francisco Brasileiro, Jacques Sauvé, Fabrício A. B. Silva, Carla O. Barros, e Cirano Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. *Parallel Processing, International Conference on*, 0:407, 2003. ISSN 0190-3918. doi: http://doi.ieeecomputersociety.org/10.1109/ICPP.2003.1240605.

National Research Council. *Realizing the Information Future: The Internet and Beyond*. National Academies Press, 1994. ISBN 0309050448.

Karl Czajkowski, Ian Foster, Carl Kesselman, Volker Sander, Volker S, e Steven Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *In 8th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 153–183, 2002.

Karl Czajkowski, Don Ferguson, Ian Foster, Jeff Frey, Steve Graham, Tom Maguire, David Snelling, e Steve Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring and evolution, version 1.1. http://www.globus.org/wsrf/specs/ogsi-to-wsrf-1.0.pdf, May 2004a.

Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, e William Vambenepe. The ws-resource framework, version 1.0. http://www.globus.org/wsrf/specs/ws-wsrf.pdf, May 2004b.

Frank Eyermann e Burkhard Stiller. A protocol to support multi-domain auditing of internet-based transport services. In *ICIMP '07: Proceedings of the Second International Conference on Internet Monitoring and Protection*, page 19, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2911-9. doi: http://dx.doi.org/10.1109/ICIMP.2007.5.

I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, e J. Von Reich. The open grid services architecture, version 1.5. http://www.ogf.org/documents/GFD.80.pdf, July 2006.

Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In Hai Jin, Daniel Reed, e Wenbin Jiang, editors, *Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Berlin / Heidelberg, 2005.

Ian Foster e Carl Kesselman. The grid: blueprint for a new computing infrastructure. pages 259–278, 1998.

Ian Foster, Carl Kesselman, e Steven Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001.

Ian Foster, Carl Kesselman, Jeffrey M. Nick, e Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Globus Alliance, http://www.globus.org/alliance/publications/papers/ogsa.pdf, 2002.

Erich Gamma, Richard Helm, Ralph Johnson, e John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN 0201633612.

S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, e R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Pearson Education, 2001.

Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, e Yves Lafon. Simple object access protocol (soap) 1.2. http://www.w3.org/TR/soap/, April 2007.

Hasan e Burkhard Stiller. A generic model and architecture for automated auditing. In *DSOM*, pages 121–132, 2005.

Hasan e Burkhard Stiller. Auric: A scalable and highly reusable sla compliance auditing framework. In *DSOM*, pages 203–215, 2007.

T. Maguire I. Foster e D. Snelling. Ogsa wsrf basic profile 1.0. http://www.ogf.org/documents/GFD.72.pdf, May 2006.

Itu. Terms and definitions related to quality of service and network performance including dependability. http://www.itu.int/rec/T-REC-E.800/en, August 1994.

ITU. Information technology - open distributed processing - reference model: Foundations. http://www.itu.int/itu-t/recommendations/rec.aspx?rec=X.902, September 1995.

ITU-T. Compendium of approved itu-t security definitions. Technical report, Study Group on Communication System Security, 2003.

Li-jie Jin, Vijay Machiraju, e Akhil Sahai. Analysis on service level agreement of web services. Technical report, HP Laboratories, 2002.

Alexander Keller e Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11 (1):57–81, 2003. ISSN 1064-7570. doi: http://dx.doi.org/10.1023/A:1022445108617.

Attila Kertesz, Gabor Kecskemeti, e Ivona Brandic. An sla-based resource virtualization approach for on-demand service provision. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 27–34, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-580-2. doi: http://doi.acm.org/10.1145/1555336.1555341.

Derrick Kondo, Andrew A. Chien, e Henri Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 17, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2153-3. doi: http://dx.doi.org/10.1109/SC.2004.50.

H. Kreger. Web services conceptual architecture. Technical report, IBM Software Group, 2001.

D. Davide Lamanna, James Skene, e Wolfgang Emmerich. Slang: A language for defining service level agreements. In *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, page 100, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1910-5.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359545.359563.

Mario Marchese. *QoS Over Heterogeneous Networks*. Wiley, 2007. ISBN 978-0-470-01752-4.

Piergiulio Maryni e Franco Davoli. Load estimation and control in best-effort network domains. *J. Netw. Syst. Manage.*, 8(4):527–541, 2000. ISSN 1064-7570. doi: http://dx.doi.org/10.1023/A:1026434517164.

Daniel A. Menascé e Emiliano Casalicchio. Qos in grid computing. *IEEE Internet Computing*, 8(4):85–87, 2004. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/MIC.2004.24.

Elliott I. Organick. *The multics system: an examination of its structure*. MIT Press, Cambridge, MA, USA, 1972. ISBN 0-262-15012-3.

D. Clark R. Braden e S. Shenker. Integrated services in the internet architecture: an overview (intserv). http://tools.ietf.org/html/rfc1633, June 1994.

Hasan Peter Racz e Burkhard Stiller. Monitoring of sla compliances for hosted streaming services. In *IM'09: Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, pages 251–258, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-3486-2.

Franco Raimondi, James Skene, e Wolfgang Emmerich. Efficient online monitoring of web-service slas. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 170–180, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-995-1. doi: http://doi.acm.org/10.1145/1453101.1453125.

Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–110, 1982. doi: 10.2307/1912531.

Akhil Sahai, Sven Graupner, Vijay Machiraju, e Aad van Moorsel. Specifying and monitoring guarantees in commercial grids through sla. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 292, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1919-9.

R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *P2P '01: Proceedings of the First International Conference on Peer-to-Peer Computing*, page 101, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1503-7.

Glauber Scorsatto e Alba Cristina Melo. Gramos: A flexible service for ws-agreement monitoring in grid environments. In *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 534–543, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85450-0. doi: http://dx.doi.org/10.1007/978-3-540-85451-7_57.

Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, e Ram Swaminathan. Auditing to keep online storage services honest. In *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 1–6, Berkeley, CA, USA, 2007. USENIX Association.

R. Shirey. Internet security glossary. Technical report, Internet Engineering Task Force, 2000.

James Skene, Allan Skene, Jason Crampton, e Wolfgang Emmerich. The monitorability of service-level agreements for application-service provision. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 3–14, New York, NY, USA, 2007. ACM. ISBN 1-59593-297-6. doi: http://doi.acm.org/10.1145/1216993.1216997.

Sean W. Smith e Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31(9):831–860, 1999. ISSN 1389-1286.

Richard S. Sutton e Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998. ISBN 0262193981.

Konstantinos Tserpes, Dimosthenis Kyriazis, Andreas Menychtas, e Theodora A. Varvarigou. A novel mechanism for provisioning of high-level quality of service information in grid environments. *European Journal of Operational Research*, 191(3):1113–1131, 2008.

S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, e P. Vanderbilt. Open grid services infrastructure (ogsi), version 1.0. http://www.ogf.org/documents/GFD.15.pdf, June 2003.

Rustam Vahidov e Dirk Neumann. Situated decision support for managing service level agreement negotiations. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 52, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 0-7695-3075-8. doi: http://dx.doi.org/10.1109/HICSS.2008.395.

Srikumar Venugopal, Xingchen Chu, e Rajkumar Buyya. A negotiation mechanism for advance resource reservation using the alternate offers protocol. In *Proceedings of 16th International Workshop on Quality of Service*, pages 40–49, 2008.

Jia Yu, Rajkumar Buyya, e Chen Khong Tham. Qos-based scheduling of workflow applications on service grids. In *Proceedings of 1st IEEE International Conference on e-Science and Grid Computing*. IEEE CS Press, 2005.

Qian Zhu e Gagan Agrawal. A resource allocation approach for supporting time-critical applications in grid environments. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–12, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3751-1. doi: http://dx.doi.org/10.1109/IPDPS.2009.5161023.