UNIVERSIDADE FEDERAL DE SÃO CARLOS (UFSCar)

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA (CCET)

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO (PPG-CC)

GSSA: Uma Arquitetura de Segurança para *Grid Services*

Cláudio Rodolfo Sousa de Oliveira

São Carlos – SP

Maio/2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

O48ga

Oliveira, Cláudio Rodolfo Sousa de.
GSSA : uma arquitetura de segurança para grid services / Cláudio Rodolfo Sousa de Oliveira. -- São Carlos : UFSCar, 2009.
107 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Grade computacional. 2. Redes de computação - segurança. I. Título.

CDD: 004.36 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"GSSA: Uma Arquitetura de Segurança para Grid Services"

CLAUDIO RODOLFO SOUSA DE OLIVEIRA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Hélio Crestana Guardia
(Orientador - DC/UFSCar)

Prof. Dr. Wanderley Lopes de Souza
(Co-orientador - DC/UFSCar)

Prof. Dr. Hermes Senger
(DC/UFSCar)

Prof. Dr. Cláudio Fernando Resin Geyer
(UFRGS)

São Carlos
Maio/2009

Agradecimentos

Agradeço à Deus em primeiro lugar, por tudo de bom que proporcionou e proporciona em minha vida.

Agradeço à minha mãe Zildete S. Sousa, que sempre me apoiou e fez de tudo para que o sonho de cursar um mestrado, fosse realizado. À minha noiva Vanessa M. Dutra, que sempre me deu apoio para continuar e nunca me deixou desanimar.

Agradeço também ao meu orientador de Graduação, Prof. Dr. Ivanor N. de Oliveira, por despertar em mim a busca pelo conhecimento através da pesquisa. Ao meu orientador de Mestrado, Prof. Dr. Hélio C. Guardia, primeiramente por confiar em mim e por todo o auxílio, cooperação, considerações e direcionamentos que culminaram nesse trabalho de mestrado.

Não poderia deixar de esquecer dos agradecimentos aos amigos do GSDR, em especial ao David O. Lorente que sempre me ajudou, principalmente com questões envolvendo o Linux. Ao Msc. Paulo R. M. Cereda e, indiretamente ao Prof. Dr. Sadao Massago do DM/UFSCar, pelo modelo LaTeX utilizado nesta dissertação.

Agradeço aos professores do DC/UFSCar, em especial aos Profs. Drs. Sérgio D. Zorzo, Luiz C. Trevelin, José H. Saito, Antônio F. do Prado e Wanderley L. de Souza, por todos os ensinamentos durante esse período.

Agradeço também à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), por ter me patrocinado financeiramente durante o mestrado.

Por fim, agradeço a todos que, diretamente ou indiretamente, contribuíram para a realização deste trabalho.

"Pensa! O pensamento tem poder.
Mas não adianta só pensar.
Você também tem que dizer!
Diz! Porque as palavras têm poder.
Mas não adianta só dizer.
Você também tem que fazer!
Faz! Porque só saberá se o final vai ser feliz
depois que tudo acontecer. . . "

(Se Liga Aí, GABRIEL O PENSADOR)

Resumo

Mecanismos disponíveis para segurança de *Grid Service (GS)* não permitem associar, transparentemente, diferentes papéis de usuários a diferentes permissões no controle de acesso às operações e recursos. De maneira geral, uma vez autenticado em um sistema, um usuário tem acesso a todos os serviços definidos. Esta dissertação apresenta *GS Security Architecture (GSSA)*, uma arquitetura de segurança baseada na *Open GSs Architecture (OGSA)*, que visa a suprir esta deficiência. GSSA também fornece funcionalidades para autenticação, registro de *log* para não repúdio, auditoria e responsabilização (*accountability*), integridade e confidencialidade das mensagens trocadas entre os GSs e a execução de GSs a partir de *Web Services (WS)*. Esta dissertação descreve também uma implementação de GSSA chamada *GS Security Proxy (GSSP)* e avalia o uso deste *proxy* em uma aplicação da telemedicina.

Palavras-chave: Computação em Grade, *Globus Toolkit*, *Grid Service*, *Web Service*, Segurança.

Abstract

Existing mechanisms for Grid Service (GS) security control do not allow transparently binding different user roles and permissions in the access control for operations and resources. After successfuly authenticated a user may access all registered services. This dissertation presents the GS Security Architecture (GSSA), a security architecture for GSs based on the Open GSs Architecture (OGSA), which aims to fullfil this gap. GSSA also provides funcionalites for authentication, log keeping for non-repudiation, auditing and accountability, secure data exchange between GS, and the execution of GSs from withn Web Services (WS). This dissertation also describes an implementation of the GSSA, named GS Security Proxy (GSSP), and evaluates the use of this proxy in a telemedicine application.

***Keywords**: Grid Computing, Globus Toolkit, Grid Service, Web Service, Security.*

Lista de Figuras

Lista de Abreviaturas e Siglas

ACL *Access Control List*

AOP *Aspect Oriented Programming*

API *Application Programming Interface*

3AGC *Authentication and Authorization Architecture for Grid Computing*

AASDF *Authentication and Authorization Security Descriptor Framework*

CaGSA *Cancer Grid Security Architecture*

CA *Certificate Authority*

CAS *Community Authorization Service*

CPR Credencial *Proxy* Restrita

DS *Delegation Service*

DN *Distinguished Name*

EPR *End Entity Certificate*

EPR *EndPoint Reference*

FTP *File Transfer Protocol*

GT *Globus Toolkit*

GWSC *Globus Web Service Container*

GSI *Grid Security Infrastructure*

GS *Grid Service*

GSSP *Grid Service Security Proxy*

HTTP *HyperText Transfer Protocol*

LDAP *Lightweight Directory Access Protocol*

JVM *Java Virtual Machine*

JWSC *Java Web Service Core*

MAF *Multipolicy Authorization Framework*

MCMS *MyProxy Credential Management System*

OGF *Open Grid Forum*

OGSA *Open Grid Services Architecture*

OASIS *Organization for the Advancement of Structured Information Standards*

PAP *Policy Administration Point*

PDP *Policy Decision Point*

PEP *Policy Enforcement Point*

PIP *Policy Information Point*

PRE Portfolio Reflexivo Eletrônico

PERMIS *PrivilegE and Role Management Infrastructure Standards*

PKI *Public Key Infrastructure*

RFT *Reliable File Transfer*

RH *Resource Home*

RPC *Restricted Proxy Credential*

RBAC *Role-based Access Control*

GSSA *Grid Service Security Architecture*

SAOGS *Security Architecture for Open Grid Service*

GSSP *Grid Service Security Proxy*

SAML *Security Assertion Markup Language*

SOAP *Simple Object Access Protocol*

SSO *Single Sign-On*

SO Sistema Operacional

TLS *Transport Layer Security*

URL *Universal Resource Location*

VO *Virtual Organization*

VOMS *Virtual Organization Membership Service*

WS *Web Service*

WSDD *Web Services Deployment Descriptor*

WSDL *Web Services Description Language*

WSRF *Web Service Resource Framework*

W3C *World Wide Web Consortium*

XACML *eXtensible Access Control Markup Language*

XML *eXtensible Markup Language*

Sumário

CAPÍTULO

1

Introdução

"No princípio criou Deus os céus e a Terra."

BÍBLIA SAGRADA (GÊNESIS 1:1)

A tecnologia de Serviços *Web (Web Service - WS)* (BOSWORTH, 2001), (PAPAZOGLOU, 2003) é baseada na Arquitetura Orientada a Serviços (*Service Oriented Architecture - SOA*) (PAPAZOGLOU, 2003) e padronizada pela *Organization for the Advancement of Structured Information Standards (OASIS)*[1] e *World Wide Web Consortium (W3C)*[2]. Esta tecnologia consolidou-se para a troca de informações entre aplicações distribuídas, sobretudo devido ao fraco acoplamento que emprega nas comunicações entre sistemas clientes e servidores. Os serviços desta arquitetura podem ser implementados e acessados por aplicações criadas em diferentes linguagens de programação e executados em diversas plataformas computacionais.

Esta característica dos WSs torna o seu uso adequado para implementar os serviços

[1] http://www.oasis-open.org.
[2] http://www.w3.org.

de grades computacionais, em que máquinas pertencentes a diferentes domínios administrativos cooperam para a execução distribuída de aplicações. *Grid Services (GSs)* estendem a tecnologia WS e adicionam algumas especificidades, como a ciência do estado de execução dos serviços *(statefulness)* e o uso dos novos padrões de WS, como *WS Resources*, *WS Addressing*, *WS Notification*, entre outros, que permitem a divisão clara das funções da infraestrutura de Grade, facilitando a adição de novas funcionalidades. É importante frisar que, embora qualquer serviço de grade possa ser chamado de *Grid Service*, este termo é utilizado na dissertação apenas para referenciar os serviços de grade baseados em WS.

No ambiente distribuído das grades computacionais, considerações de segurança são fundamentais. Em grades baseadas no *middleware Globus Toolkit*, por exemplo, a Infraestrutura de Segurança para Grades *(Grid Security Infrastructure - GSI)* provê os mecanismos básicos necessários ao uso dos recursos da Grade de forma segura. Nesta infraestrutura, o controle de acesso é baseado principalmente em um arquivo de autorização chamado *grid-mapfile*, o qual permite aos usuários autenticados e registrados neste arquivo, executar qualquer operação dos GSs. Outros mecanismos de segurança e autorização também são utilizados pelo GSI e podem ser configurados de diferentes formas: nos descritores de distribuição e de segurança dos Provedores de GS *(GS Providers)*, nos descritores do *WS Container*, e nos Clientes de GS *(GS Clients)*.

Arquiteturas de segurança adicionais para Grades Computacionais estendem as funcionalidades especificadas pela Arquitetura Aberta para GSs *(Open Grid Services Architecture - OGSA)*, que trata da padronização das funcionalidades comuns dos GSs. No que tange o controle de segurança, *Security Architecture for Open Grid Services (SAOGS)* fornece autorização no acesso a GSs, *caGrid Security Architecture (caGSA)* fornece autorização a operações de GSs sendo necessário modificá-los para isso, e *Authentication and Authorization Architecture for Grid Computing (3AGC)* utiliza a Programação Orientada a Aspectos (*Aspect-Oriented Programing - AOP*) no auxílio à autorização de GSs. Também há mecanismos de segurança que estendem as funcio-

nalidades do GSI, como *GridShib*, *Virtual Organization Membership Service (VOMS)*, *PrivilEge and Role Management Infrastructure Standards (PERMIS)*, *Akenti*, *Multipolicy Authorization Framework (MAF)*, oferecendo auxílio aos procedimentos de autorização.

1.1 Motivação

Mesmo com todas as soluções de segurança disponíveis, os GSs, de grades baseadas no *Globus Toolkit*, ainda carecem de um mecanismo de autorização que permita especificar, com base em papéis, quais as operações dos GSs que os usuários têm permissão de executar, de forma transparente aos *GS Providers*.

Também não há mecanismos para o controle de acesso a recursos de GSs, especificando quais usuários podem ler ou alterar os valores destes recursos.

Outra carência de segurança existente é que os sistemas de autorização possuem dificuldade em manter a sincronia das políticas dos usuários, ou seja, garantir que um usuário consiga realizar apenas o que lhe é permitido no exato momento, sem onerar os mecanismos de comunicação. Para tanto, as soluções normalmente utilizadas são consultas periódicas ao sistema de autorização pelo usuário, ou o reinício deste sistema quando as políticas de autorização são atualizadas.

Devido à crescente utilização de WS, um mecanismo para que permita acessar *GS Providers*, de forma segura, a partir de Clientes de WS (*WS Clients*) também é conveniente.

1.2 Objetivos

Visando atender as demandas apresentadas, esta dissertação apresenta uma arquitetura de segurança para GSs, chamada *Grid Service Security Architecture (GSSA)*, que estende a arquitetura OGSA, utilizando seus principais padrões, como *WS Notification*, *WS Addressing*, *WS Resource*, *WS ResourceProperties* e *WS ResourceLifetime*.

A principal funcionalidade especificada por GSSA é um controle de acesso às operações e aos recursos de *GS Providers*, para garantir que apenas os usuários autor-

izados possam executar as operações, ou acessar e modificar os recursos. O controle das operações previsto não necessita alteração dos GSs e exige mínimas modificações nos Clientes de GS *(GS Clients)*. Em relação ao controle de acesso aos recursos de *GS Providers*, GSSA permite especificar quem pode acessá-los ou alterá-los, sem que esta funcionalidade exija a modificação do código fonte de *GS Providers*, bastando que estes sejam recompilados.

GSSA fornece também mecanismos para autenticação, pois autorização demanda identificar univocamente os usuários. Como informações sigilosas podem estar sendo manipuladas pelos GSs, a arquitetura fornece integridade e confidencialidade aos dados trafegados entre eles. Com o controle de quem acessa e o que é acessado, GSSA emprega um registro de *logs* para auxiliar a realização de auditorias, a fim de identificar possíveis falhas de segurança, e *accountability*, para responsabilizar estes usuários por suas ações. GSSA também especifica um mecanismo de tradução para permitir acessar GSs a partir de WSs de forma segura, e garantir o não repúdio, para que usuários não possam negar a autoria das ações realizadas.

A arquitetura proposta procura modularizar as questões de segurança da aplicação cliente, de forma que os desenvolvedores destas aplicações não precisem preocupar-se com essas questões e possam focar-se nas funcionalidades do sistema. Para isso, esta arquitetura exige mínimas alterações na aplicação cliente e nos serviços fornecidos. Além disso, objetiva-se que a arquitetura forneça mecanismos para configuração da aplicação cliente, a fim de poder utilizar diferentes níveis de segurança.

Um *proxy*, chamado *Grid Service Security Proxy (GSSP)*, foi implementado em Java de acordo a arquitetura GSSA proposta, a fim de validá-la. Para tanto, utiliza-se alguns componentes do *Globus Toolkit*, tais como o *Community Authorization Service (CAS)* no auxílio à autorização; o *MyProxy Credential Management System (MCMS)*, no auxílio a autenticação; o *framework Java WS Core (JWSC)*, para implementar os GSs; o *Globus WS Container (GWSC)*, para disponibilizá-los; o *Authentication and Authorization Security Descriptor Framework (AASDF)*, para configuração da transferência segura das mensagens *Simple Object Access Protocol (SOAP)*; e o *Globus Service Build*

Tool (GSBT)[3], para geração das classes *Stub* que tornam transparente para os GSs o envio das mensagens pela rede, além de gerar o serviço no formato GAR *(Grid Archive)* para publicação. Desta forma, a implementação da arquitetura usa as funcionalidades da GSI para prover um controle maior de segurança no acesso às operações dos GSs.

Para a sincronia das políticas do usuário, foi utilizado o padrão *WS Notification*, no qual o sistema de autorização notifica automaticamente os *GS Clients* quando há mudanças nas políticas dos usuários, para que estes possam proceder a busca pelos novos direitos.

Um *framelet* (*framework* de pequeno porte) de autorização foi desenvolvido baseado no paradigma de programação orientado a Aspectos *(Aspect Oriented Programming - AOP)*, para permitir aos desenvolvedores de infraestrutura de autorização indicar quais classes terão os métodos gerenciados. Caso não desejem utilizar o CAS, este *framelet* também permite à aplicação criar seu módulo próprio para mapear as decisões de autorização em recursos de GSs.

1.3 Estrutura do Texto

O restante da dissertação está organizado como subscrito.

O Capítulo 2 discute sobre considerações principais da Computação em Grade, a arquitetura OGSA e o *Web Service Resource Framework (WSRF)* que a implementa.

O Capítulo 3 apresenta o *middleware Globus Toolkit* com os seus principais serviços para controle de segurança.

O Capítulo 4 mostra as principais arquiteturas e mecanismos de segurança para grades computacionais baseadas no *Globus Toolkit*.

O Capítulo 5 apresenta as camadas da GSSA, com as respectivas funcionalidades.

O Capítulo 6 apresenta o GSSP, que é uma implementação da arquitetura GSSA proposta e detalha como foi implementada cada funcionalidade da arquitetura.

O Capítulo 7 aborda os trabalhos relacionados, comparando os mecanismos exis-

[3] `http://gsbt.sourceforge.net`.

tentes para segurança em grades computacionais baseadas no *Globus Toolkit*, com a arquitetura proposta.

O Capítulo 8 relata o estudo de caso utilizando uma aplicação da telemedicina conhecida como Portfólio Reflexivo Eletrônico (PRE), que é acessada por alunos e professores do curso de Medicina da Universidade Federal de São Carlos (UFSCar), via *Web*.

O Capítulo 9 apresenta e analisa os resultados obtidos, comparando a abordagem atual, no qual o *GS Client* acessa diretamente o *GS Provider*, com a que utiliza a implementação da arquitetura. Para isso, analisou-se os tempos de resposta envolvendo diversas requisições, para análise do *overhead* introduzido com o uso da implementação desenvolvida. Além disso, uma análise de segurança também é realizada.

Por fim, o Capítulo 10 apresenta algumas considerações finais e aponta para trabalhos futuros, referentes a controle de acesso a parâmetros de operações e maior granularidade na concessão de permissões.

CAPÍTULO

2

Computação em Grade

"Conectar computadores é um trabalho. Conectar pessoas é uma arte."

ECKART WINTZEN

A computação distribuída é caracterizada pela interligação de diversos computadores através de uma rede, que se comunicam por meio de bibliotecas de passagem de mensagens e por possuir um software (e.g., *middleware*, Sistema Operacional) que gerencia esses computadores, de forma que o usuário tenha a impressão de utilizar um único sistema coerente. (TANEMBAUM, 2003).

Computação em Grade *(Grid Computing - GC)* (FOSTER; KESSELMAN, 1998b), (FOSTER et al., 2001), (FOSTER et al., 2002b) é um ramo da computação distribuída. O termo Computação em Grade surgiu em alusão á rede de energia elétrica *(Electric Power Grid)*, que disponibiliza energia sob demanda para inúmeros usuários, sem que eles precisem se preocupar em como ela chega em suas casas, sendo necessário apenas plugá-la e utilizá-la (CIRNE; NETO, 2005).

De forma similar, a Computação em Grade seria uma rede distribuída, na qual os

indivíduos se conectam para obter serviços computacionais, que agregam recursos computacionais (e.g., ciclos, armazenamento, software, periféricos), sob demanda, de forma similar ao apresentado na Figura 2.1 (CIRNE; NETO, 2005).

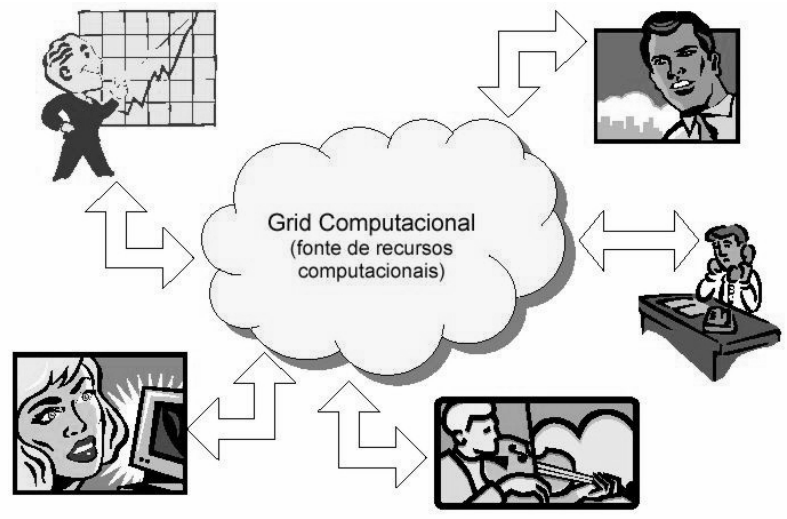

Figura 2.1: Acesso transparente a serviços e recursos computacionais. Fonte: (CIRNE; NETO, 2005).

Os principais objetivos da Computação em Grade são:

- prover serviços sob demanda, permitindo uma maior colaboração entre várias instituições e empresas, por meio do compartilhamento de serviços e recursos computacionais e utilizando mecanismos que facilitem esta interoperabilidade;
- fornecer alto poder computacional, com auxílio de processamento paralelo, distribuído e grande quantidade de recursos computacionais disponíveis (e.g., inúmeros computadores conectados via rede e Internet);
- disponibilizar grande capacidade de armazenamento de dados. As Grades que tem este como objetivo principal são conhecidas como Grades de Dados;
- permitir o acesso a recursos computacionais distribuídos geograficamente de forma transparente, isto é, para o usuário há a impressão de estar acessando um único computador robusto; e

- segurança no acesso a sistemas e dados entre múltiplos domínios, levando em consideração as políticas de segurança de cada domínio.

É necessário que todos estes objetivos sejam alcançados por um custo financeiro inferior às alternativas baseadas em supercomputadores paralelos, apesar desta solução aumentar a complexidade do gerenciamento do sistema.

Em suma, a Computação em Grade é a ciência que estuda meios para tornar viável às tecnologias (e.g., protocolos, ferramentas) necessárias à construção de Grades Computacionais.

2.1 Grades Computacionais

Grades Computacionais (JACOB et al., 2005) são aglomerados de computadores heterogêneos, trabalhando colaborativamente por meio de uma infraestrutura de rede para troca, armazenamento e processamento de informações, no qual envolve diversos domínios administrativos. O progresso crescente dos recursos computacionais (*e.g., hardware*), a popularização da Internet e o baixo custo das redes de computadores de alta velocidade viabilizaram economicamente a criação destas Grades.

As principais características das Grades Computacionais são:

- "Múltiplos Domínios Administrativos", onde cada domínio é responsável pelo gerenciamento de seus recursos computacionais;

- "Heterogeneidade", por serem constituídas de inúmeros tipos de *hardware*, Sistemas Operacionais, *middlewares*, e redes de comunicação, formando um ambiente altamente heterogêneo. Os computadores da Grade são componentes que sozinhos funcionam de forma independente, e estão conectados a ela participando do ambiente de forma colaborativa.

- "Escalabilidade", por permitir o incremento de novos computadores, melhorando o desempenho do ambiente como um todo;

- "Adaptabilidade", de forma que um Gerenciador de Recursos computacionais evita inconsistência ou perda de informações caso algum computador falhe. Este gerenciador monitora de forma dinâmica a inserção ou saída de novos computadores na Grade Computacional;

- "Tolerância a falhas", por fornecer alta disponibilidade dos serviços porque são formadas por inúmeros computadores. Desta forma, para que o sistema falhe por completo é preciso que todos os computadores que o disponibiliza, falhem concomitantemente.

De maneira geral, a implementação das funcionalidades das grades computacionais é implementada na forma de *middlewares*, que estendem os sistemas existentes nos recursos computacionais heterogêneos interligados. Inúmeros *middlewares* foram desenvolvidos para gerenciar os recursos de Grades Computacionais com o objetivo de viabilizar, facilitar e tornar popular a utilização deste ambiente.

2.2 Organização Virtual

Os recursos de Grades Computacionais são compartilhados de forma controlada com definição clara de quem pode acessar recursos compartilhados e sob quais condições este compartilhamento ocorre. Um conjunto de indivíduos e instituições (e.g., empresas, centros de pesquisas) definidas por estas regras de compartilhamento forma uma Organização Virtual (*Virtual Organization* - VO) (FOSTER et al., 2001).

Computadores podem simultaneamente pertencer a inúmeras VOs, de acordo com seus interesses, como pode ser observado na Figura 2.2. Os computadores pertencentes a uma VO que seguem as mesmas políticas de segurança são chamados de Domínios Administrativos. Uma VO pode ter um ou mais domínios administrativos e uma Grade Computacional pode ser formada por uma ou mais VOs autônomas (YU; NG, 2007).

Todos os recursos computacionais de uma VO só podem ser utilizados por usuários da respectiva VO, ou seja, pelos usuários que tiveram os certificados gerados pela

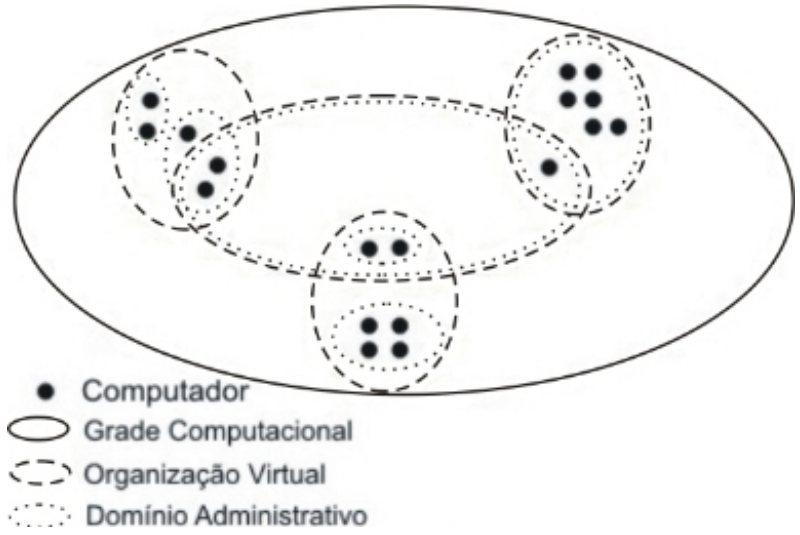

Figura 2.2: Grade Computacional com diversas VOs e Domínios Administrativos.

Autoridade Certificadora da VO e que estão devidamente autenticados e autorizados.

Quando é necessário extrapolar as fronteiras da organização, para utilizar os recursos pertencentes a outra, é necessário que o usuário seja representado por outro usuário da organização que ele precise acessar. Desta forma, é como se o acesso fosse feito por um usuário da VO almejada. Normalmente, estes usuários remotos são mapeados em usuários locais com mais restrições no acesso aos recursos que os usuários próprios da VO.

2.3 *Open Grid Services Architecture (OGSA)*

Open Grid Services Architecture (OGSA) (FOSTER et al., 2002b) foi desenvolvida pela *Open Grid Forum (OGF)*[1] com o objetivo de definir uma arquitetura comum, padrão e aberta para aplicações baseadas em Grade. Esta arquitetura almeja padronizar as funcionalidades comumente encontradas em serviços de Grade (e.g., gerenciamentos de *jobs*, recursos computacionais, segurança, etc.), por meio da especificação de um conjunto de padrões de interfaces para estes serviços (SOTOMAYOR, 2005). Contudo, o conjunto de interfaces encontra-se ainda em definição.

Considerando que, por exemplo, OGSA define que a interface *ResourceDiscovery-*

[1] `http://www.ogf.org`.

Interface tem um método *discoverResource*, deve existir um modo padrão e comum para invocar este método, para que a arquitetura seja adotada como um padrão pela comunidade de Grade. A tecnologia base para a arquitetura poderia, em teoria, ser qualquer tecnologia para comunicação entre aplicações distribuídas (e.g., *Common Object Request Broker Architecture (CORBA)*, *Remote Method Invocation (RMI)*, *Remote Procedure Call (RPC)*, WS), sendo que a tecnologia WS foi considerada a mais adequada.

Embora a arquitetura WS seja a melhor opção, esta não possui um dos mais importantes requisitos do OGSA; a tecnologia de comunicação deve permitir a implementação de serviços que possuam um estado associado (*stateful*). Embora os WS em teoria podem ser tanto *stateless* como *stateful*, eles geralmente são *stateless*, e não existem implementações de WS *stateful* comumente utilizadas.

2.4 *Web Service Resource Framework (WSRF)*

O *Framework* para Recursos de WSs *Web Services Resource Framework (WSRF)* (CZAJKOWSKI et al., 2004) desenvolvido e suportado por OASIS, implementa os requisitos da arquitetura OGSA, isto é, WSRF fornece os serviços *stateful* que a OGSA especifica.

Desta forma, enquanto OGSA é a arquitetura, WSRF é a infraestrutura na qual a arquitetura foi construída. Este *framework* provê um modelo de programação visando o desacoplamento entre o estado do recurso e o WS, responsável pela lógica do negócio do serviço (GAWOR, 2004).

O uso do WSRF em Grades Computacionais permite dividir claramente as funções da infraestrutura de Grade, facilitando o suporte incremental e o uso dos diversos padrões de WS.

A especificação do WSRF é formada pelas especificações *WS-Resource*, *WS-ResourceProperties*, *WS-ResourceLifetime*, *WS-ServiceGroup* e *WS-BaseFaults*, da OGSA (CZAJKOWSKI et al., 2004).

- *WS-Resource*: São objetos instâncias de classes ou variáveis que guardam o

estado de execução do WS, representando assim os recursos de GSs.

- *WS-ResourceProperties*: Um recurso é composto por zero ou mais Propriedades de Recurso *Resource Properties*. Estes especificam como as propriedades dos recursos são definidas e acessadas. *Resources Properties* são definidos no arquivo *Web Service Description Language (WSDL)* do GS, que é um documento XML, a fim de informar a *Universal Resource Location (URL)* para acessar o serviço, o nome do serviço, a descrição de cada operação e a forma de realizar requisições às operações.

- *WS-ResourceLifetime*: Recursos não são entidades estáticas que são criadas quando o servidor inicia, ou destruídas, quando o servidor é desligado. Recursos podem ser criados e destruídos a qualquer momento. *WS-ResourceLifetime* fornece mecanismos básicos para gerenciar o ciclo de vida destes recursos.

- *WS-ServiceGroup*: O gerenciamento (e.g., adicionar, remover, encontrar) de grupos de WS ou de Recursos é feito com esta especificação. Esta funcionalidade é a base de muitos sistemas que descobrem recursos como o *IndexService* (GONÇALVES et al., 2006) do *middleware Globus Toolkit 4*.

- *WS-BaseFaults*: Esta especificação descreve um padrão de reportar falhas quando uma exceção é gerada na chamada do WS.

WSRF também implementa as especificações auxiliares *WS Notification* e *WS Addressing* (CZAJKOWSKI et al., 2004).

- *WS Notification*: Esta especificação permite que WSs sejam configurados como Fornecedores de Notificação *(Notification Providers)*, e clientes como Assinantes de Notificação *(Notification Subscribers)*. Quando um evento ocorre no WS (e.g., exceção, requisição ou retorno da execução de um método, modificação de um recurso), uma notificação é enviada aos clientes assinantes do evento em questão.

- *WS Addressing*: Esta especificação fornece mecanismos para endereçar WS, independentemente da camada de transporte. Cada GS é unicamente identificado

através de um objeto *EndPointReference (EPR)* que contém a *Universal Resource Location (URL)* e, opcionalmente, o identificador do recurso.

2.4.1 *Grid Services (Stateful Web Services)*

Web Services são compostos por recursos (atributos e/ou objetos instâncias de classes) e operações (procedimentos ou métodos) que executam processamento sobre estes recursos. Estes serviços são *stateless*, i.e., tanto os recursos quanto as operações desconhecem o estado de execução da aplicação e não armazenam qualquer informação dos recursos entre as requisições das operações. Os recursos dos GSs (conhecidos como *WS Resource* no ambiente de Grade) normalmente precisam ter um estado associado *stateful*, bem como mecanismos para inspeção e manipulação desse estado. Desta forma, o conceito de WS foi estendido para o uso em Grades Computacionais, construindo um novo tipo de *Web Service* que é *stateful* identificado como *Grid Service* (FOSTER et al., 2002a), (SEBU; CIOCÂRLIE, 2006). A Figura 2.3 apresenta a relação entre estes componentes.

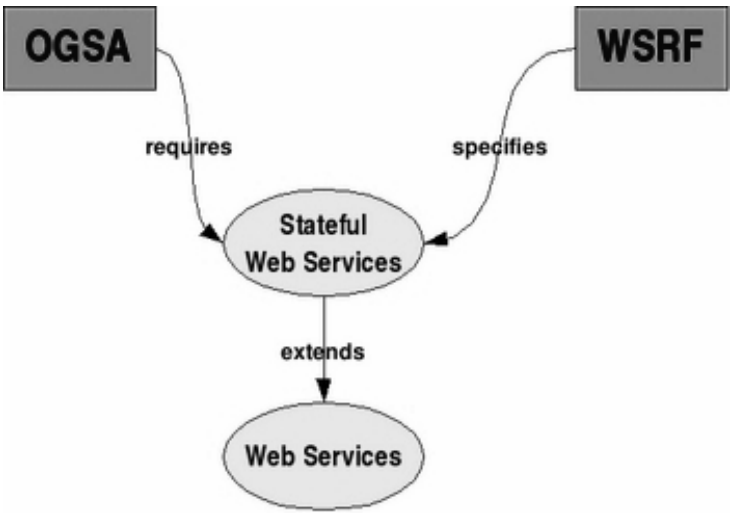

Figura 2.3: Relação entre WS, *Stateful* WS, OGSA e WSRF. Fonte: (SOTOMAYOR, 2005).

GS Providers são formados por recursos[2] (*WS Resource*), por operações, que são as funções que processam informações e manipulam os *WS Resources*, e por um

[2]Recursos de GS são uma abstração em *software* de recursos computacionais.

gerenciador dos recursos conhecido como *Resource Home (RH)*, que é responsável por criar, referenciar e destruir os recursos alocados (FERREIRA et al., 2004), como pode ser observado na Figura 2.4. *WS Resource* são compostos por *ResourceProperties* que são os atributos persistentes do GS. *GS Clients* conectam a *GS Provider* a fim de chamar suas operações e utilizar os recursos disponíveis. Estes recursos são preservados entre as requisições, guardando assim o estado de execução da aplicação (*stateful*).

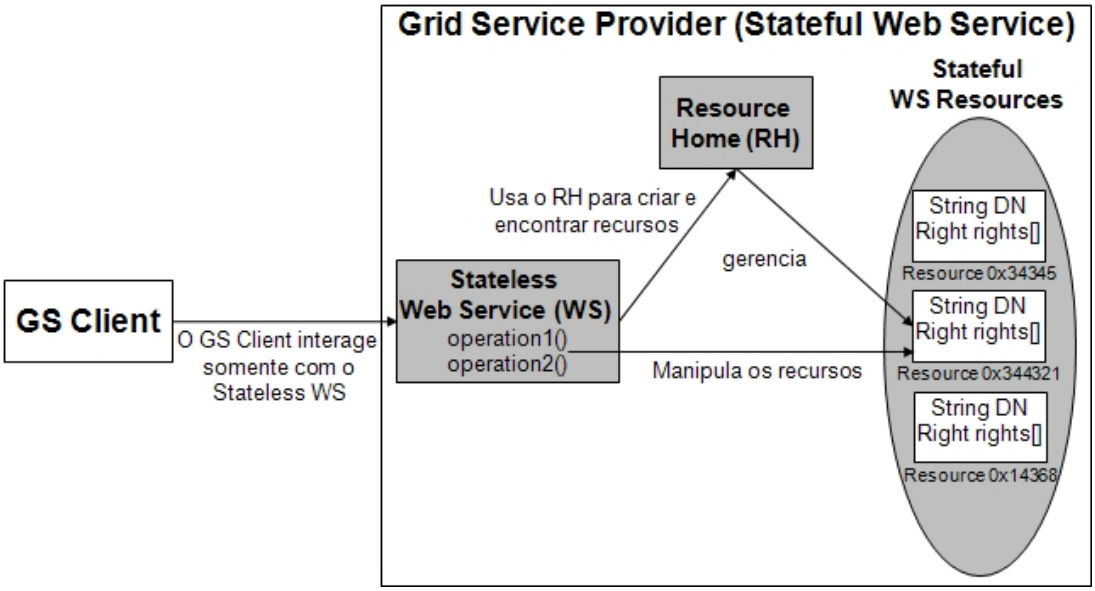

Figura 2.4: Estrutura do GS.

Uma aplicação *GS Client* acessa as operações do *GS Provider* referenciando o descritor WSDL dele e criando um objeto instância que representa o Provider. Depois, um cliente *(GS Client)* chama as operações de forma local, mas elas serão executadas remotamente, sem que a aplicação Client se preocupe com isso.

2.5 Grade *versus Cluster*

De forma similar a Grades Computacionais, *Clusters* Computacionais (Governo Eletrônico, 2006) são formados por grupos de computadores conectados entre si com o objetivo de resolver problemas computacionais robustos.

A maior diferença entre estes agrupamentos de computadores é que Grades são

distribuídas geograficamente, enquanto que *Clusters* estão na maioria das vezes em uma única localização. Outra importante diferença é que *Clusters* são comumente homogêneos, possuindo o mesmo Sistema Operacional e *hardware*, enquanto que Grades são heterogêneas.

Uma Grade pode usar o poder computacional de Computadores Pessoais *(Personal Computers - PCs)* quando ociosos, enquanto *Clusters* são quase sempre dedicados a tarefas pré-estabelecidas, não sendo usados para propósitos pessoais e sim empresariais e de pesquisa.

Outra importante diferença consiste no modo como os recursos são manipulados. No *Cluster*, o sistema inteiro comporta-se como um único sistema onde usuário e recursos são gerenciados por um gerenciador de recursos centralizado. Em relação a Grades Computacionais, todo computador é autônomo, ou seja, tem seu próprio gerenciador de recursos, como uma entidade independente. Grades são dinâmicas e nelas os recursos que integram uma grade podem se conectar ou desconectar a qualquer momento. Já no *cluster* o número de computadores é previamente conhecido, podendo calcular-se o seu poder computacional.

Apesar das diferenças, cada tecnologia possui o seu campo de aplicações práticas. Um *cluster* pode inclusive ser um valioso integrante em um ambiente de Grade.

Resumo do Capítulo 2

Este Capítulo apresentou a Computação em Grades, as principais características de Grades Computacionais e a arquitetura que possibilita a criação destas grades. Foram expostas as VOs que compõem o ambiente de grade e fez-se um comparativo deste ambiente com o de *clusters*. Apresentou-se também a arquitetura OGSA e o *framework* WSRF.

A Computação em Grade é um paradigma da computação distribuída, que estuda meios para tornar viável a criação de Grades Computacionais, que são aglomerados de dispositivos computacionais heterogêneos, trabalhando colaborativamente por meio de uma infraestrutura de rede para troca e armazenamento de informações. As

organizações virtuais são formadas por um conjunto de indivíduos e instituições que definem regras de como os recursos devem ser compartilhados. A arquitetura OGSA padroniza os GSs, e o WSRF que auxilia a implementação destes serviços. As principais diferenças entre a computação em grade e a baseada em *clusters* são a grande heterogeneidade, a abrangência e a dispersão geográfica das Grades.

No próximo Capítulo será apresentado o *middleware Globus Toolkit*, utilizado na construção de grades computacionais e os seus principais serviços de segurança.

CAPÍTULO

3

Segurança no *Globus Toolkit*

"O superior não pode atuar sobre o inferior sem um elo intermediário que os una."

FRANZ HARTMAN

O *middleware Globus Toolkit* (FOSTER; KESSELMAN, 1998a), (FOSTER, 2005b), (FOSTER, 2005a), (GLOBUS TEAM, 2009c) fornece a implementação de serviços, protocolos, bibliotecas, ferramentas, etc., necessários à construção de um ambiente de Grade Computacional. Este *middleware open source* é desenvolvido pela *Globus Alliance*[1].

Globus Toolkit consolidou-se como um padrão *de facto* para a construção de Grades Computacionais e é apoiado e financiado por grandes empresas como, *Hewlett-Packard (HP)*, *Sun Microsystems*, *International Business Machines (IBM)*, *Intel Corporation*, entre outras. *Globus Toolkit* também recebe o apoio de importantes centros de pesquisas

[1] `http://www.globus.com`.

como as Universidades de Chicago[2], Edimburgo[3], entre outras.

O *Globus Toolkit* 4 está em conformidade com a maioria dos requisitos da *Open Grid Service Architecture (OGSA)* incluindo funcionalidades de monitoramento e descoberta de serviços pertencentes à camada *Information Services*, uma infraestrutura de submissão de tarefas, formada pela camada *Execution Management*, uma infraestrutura de segurança, que corresponde à camada *Security*, e funcionalidades de gerenciamento de dados, que correspondem à camada *Data Management*, como pode ser observado na Figura 3.1.

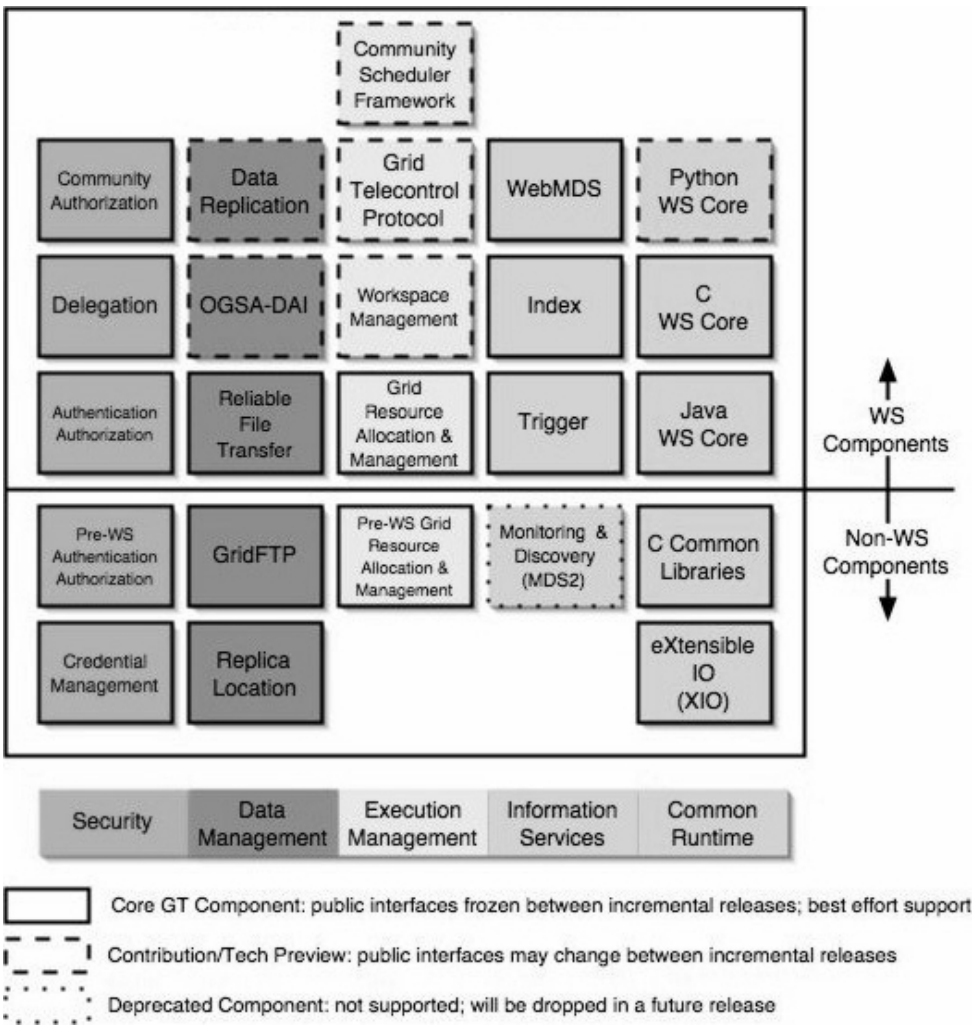

Figura 3.1: Infraestrutura de serviços do *Globus Toolkit* 4. Fonte: (FOSTER, 2005a).

[2] `http://www.uchicago.edu`.
[3] `http://www.ed.ac.uk`.

Essa figura mostra os principais serviços ou componentes de cada camada. Os componentes WS (*WS Components*) do *Globus Toolkit* são implementados utilizando o *Web Service Resource Framework (WSRF)*. Já os serviços do *Globus Toolkit* que não são implementados utilizando este *framework* são chamados de componentes não WS (*Non-WS Components*). O *Globus Toolkit* 4 inclui uma implementação parcial do WSRF/OGSA, faltando, por exemplo, suporte aos Distribuidores de Notificações. A relação entre a OGSA, o WSRF e o *Globus Toolkit* 4 pode ser visto na Figura 3.2.

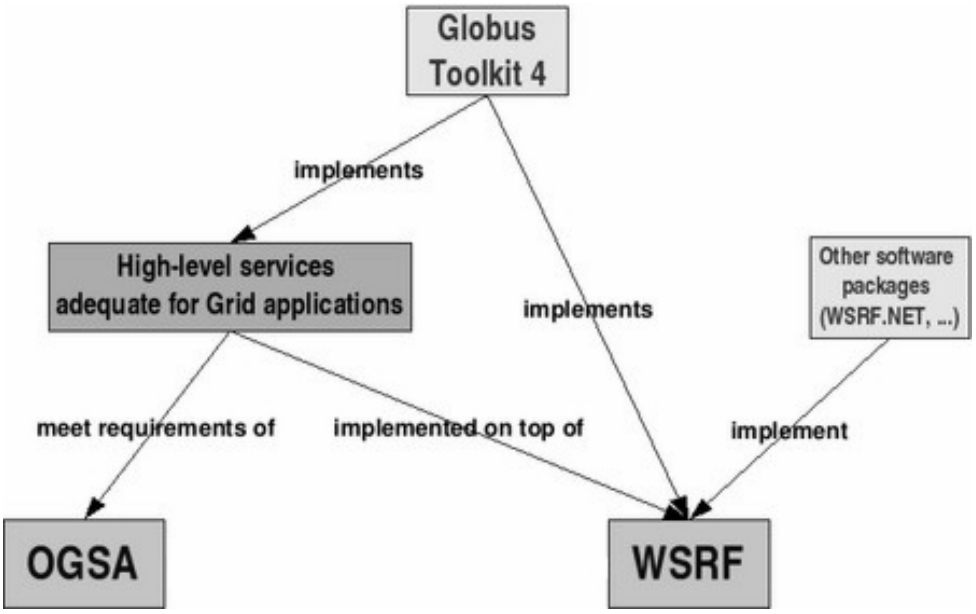

Figura 3.2: Relação entre OGSA, WSRF e *Globus Toolkit* 4. Fonte: (SOTOMAYOR, 2005).

No *Globus Toolkit*, a Infraestrutura de Segurança para Grade *(Grid Security Infrastructure - GSI)* engloba toda a camada *Security* da Figura 3.1. GSI é responsável por fornecer os mecanismos de segurança para o controle de recursos de Grades Computacionais gerenciados pelo *middleware Globus Toolkit*. Para tanto, esta infraestrutura é formada por diversos mecanismos com fins específicos, como o *MyProxy Credencial Management System (MCMS)*, usado no gerenciamento de credenciais, *Community Authorization Service (CAS)*, que permite o controle no acesso aos recursos da Grade Compucional, *Delegation Service (DS)*, utilizado para delegar credenciais a outros serviços hospedados no mesmo *container* deste serviço, e *Authentication and*

Authorization Security Descriptor Framework (AASDF), para o controle de autenticação e autorização, seja para componentes WS ou Non-WS.

3.1 *Grid Security Infrastructure (GSI)*

Grid Security Infrastructure (GSI) (FOSTER et al., 1998), (BUTLER et al., 2000), (GLOBUS SECURITY TEAM, 2009d) é baseada no uso de uma Infraestrutura de Chaves Públicas *(Public Key Infrastructure - PKI)* (TUECKE et al., 2004), na qual são utilizadas credenciais digitais para fins de autenticação e criptografia. Uma entidade confiável, conhecida como Autoridade Certificadora *(Certificate Authority - CA)* (GLOBUS SECURITY TEAM, 2009h) armazena e assina os certificados digitais utilizados na Grade. GSI também fornece outros mecanismos que estendem a PKI, como Credenciais *Proxy* e Delegação de Credenciais.

Como pode ser observado na Figura 3.3, GSI possui camadas bem definidas para prover as funcionalidades de autenticação, autorização, proteção e formatação das mensagens SOAP. Para cada camada há uma série de padrões e mecanismo que são utilizados a fim de atender a estas funcionalidades.

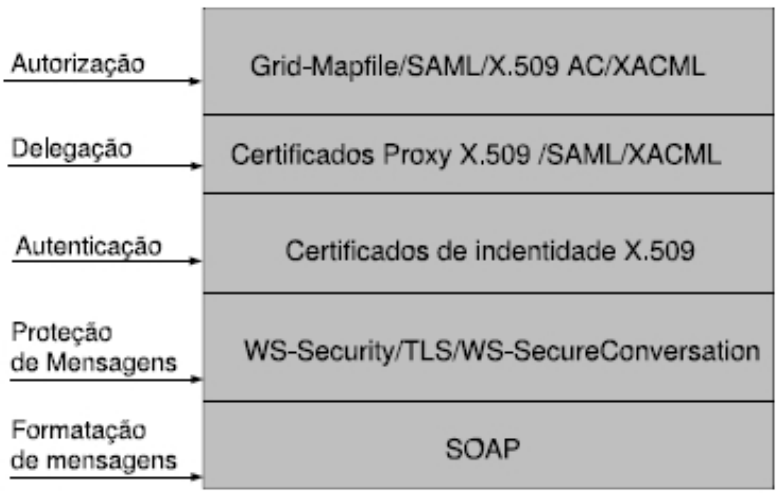

Figura 3.3: Visão Geral do GSI. Fonte: (JUNIOR; KON, 2007).

Há vários tipos de credenciais digitais (incluindo de usuário, de computador e de serviço) que seguem o padrão X.509. Estas credenciais são chamadas de *End Entity*

Credentials (EEC). Uma credencial digital é formada por um certificado digital e sua respectiva chave privada. O certificado digital contém um *Distinguished Name (DN)* do usuário (formado por organização, unidade organizacional, nome comum e país), a chave pública, a data de criação, a data de expiração, o DN da CA, e a assinatura digital da CA. A chave privada é guardada de forma cifrada no sistema e para acessá-la é necessário informar uma senha cadastrada.

Credenciais *Proxy* (TUECKE et al., 2004) são criadas a partir de uma EEC, ou de outra credencial *proxy*, e assinadas pela credencial criadora, a fim de serem utilizadas em seu lugar em processos de autenticação. Estes certificados também possuem chaves pública e privada próprias.

A Credencial *Proxy* possui um tempo de vida limitado, por isso, sua chave privada não é protegida criptograficamente com uso de senha. A chave privada associada a este tipo de certificado é protegida apenas pelas restrições no sistema de arquivos do sistema. Isso evita que a chave privada da credencial do usuário seja decifrada constantemente, diminuindo sua exposição (NOVOTNY et al., 2001). Credenciais *Proxy* viabilizam também o mecanismo de *Single Sign On (SSO)*, que é a aparência de uma autenticação única do usuário no sistema, evitando a inconveniência de sua re-autenticação em um curto período de tempo para acesso coordenado a múltiplos recursos da grade.

Há também uma classe de Credenciais *Proxy*, chamadas de Restritas, que contém além das informações comuns, direitos de acesso sobre recursos.

Para que um computador faça parte efetiva da Grade Computacional, isto é, para que seus recursos possam ser utilizados em benefício das aplicações da Grade, este deve possuir uma Credencial de Computador (FERREIRA et al., 2003).

Os passos para obtenção da Credencial de Computador estão na Figura 3.4. São eles: 1) criação de um certificado e uma chave privada referente a ele; 2) envio deste certificado para a CA (e.g., via transferência de arquivo ou e-mail); 3) assinatura do certificado de computador pela CA; 4) envio do certificado assinado de volta para o computador requisitante para que ele possa apresentá-lo quando lhe for requisitado

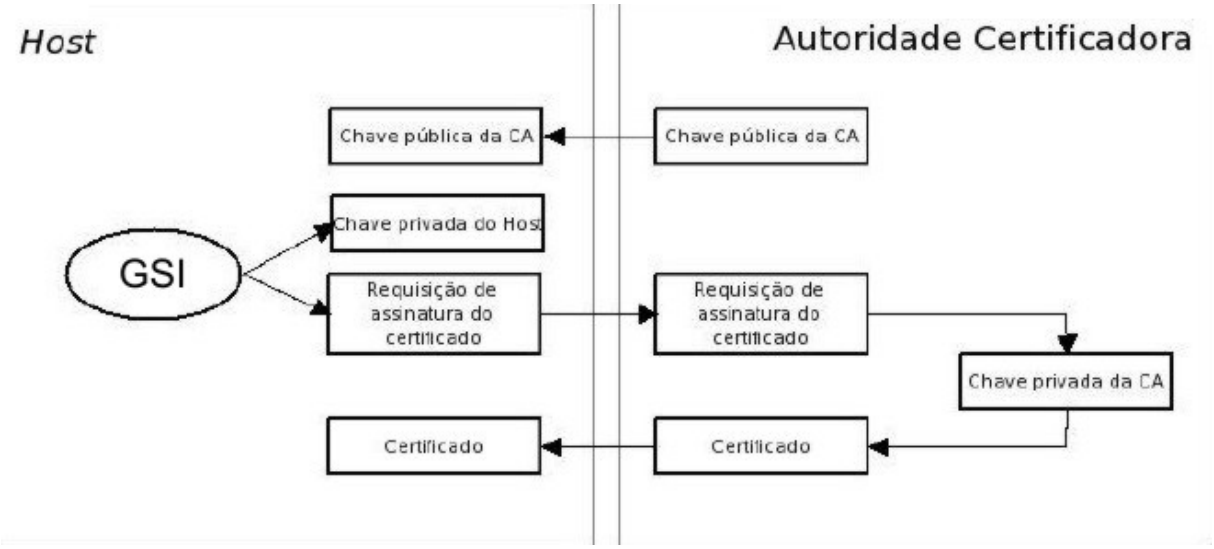

Figura 3.4: Processo de Criação de Certificados Digitais com o GSI. Fonte: (FERREIRA et al., 2003).

(FERREIRA et al., 2003).

Para que um usuário possa autenticar-se e utilizar os recursos da grade, ele deve possuir uma Credencial de Usuário. Os passos para obtenção da Credencial de Usuário são os mesmos da obtenção da Credencial de Computador, mudando apenas os caminhos da localização do certificado e da chave privada (FERREIRA et al., 2003).

3.1.1 *MyProxy Credential Management Service (MCMS)*

O Serviço de Gerenciamento de Credenciais *(MyProxy Credential Management Service - MCMS)* (NOVOTNY et al., 2001), (GLOBUS SECURITY TEAM, 2009e) permite acesso remoto e delegação das credenciais do usuário de diversas Organizações Virtuais (*Virtual Organizations - VO*), de modo seguro, desde que estas estejam autorizadas no arquivo *myproxy-server.config*. Isso elimina a necessidade de copiar manualmente as credencias entre computadores quando um usuário quiser autenticar-se em outros computadores que não têm suas credenciais, prevenindo problemas de segurança que esta cópia pode trazer.

MCMS permite: 1) que qualquer aplicação autônoma acesse a Grade Computa-

cional e realize tarefas em favor do usuário; 2) ao usuário ou aplicação, autenticar-se na Grade mesmo que não tenha credenciais e 3) ao usuário ou aplicação desenvolver qualquer operação que a credencial conceda.

MCMS utiliza o serviço de delegação para delegar uma credencial *proxy* ao usuário solicitante; posteriormente, MCMS retorna a credencial ao usuário por um canal seguro de comunicação. *MyProxy* pode realizar esta delegação porque tem uma credencial de cada usuário delegada ao repositório *MyProxy*.

Quando os usuários da Grade desejam armazenar suas credenciais no repositório do MCMS, os seguintes passos devem ser realizados: 1) o usuário requisita ao serviço MCMS a inserção de uma nova credencial, utilizando a interface *myproxy-init*; 2) MCMS requisita uma senha ao usuário (esta senha será usada para codificar a chave privada da credencial que o repositório armazenará), cria uma credencial e a envia ao usuário; 2) o usuário assina esta credencial utilizando sua credencial *proxy* gerada na autenticação na Grade, e a reenvia ao MCMS, e 3) MCMS armazena esta credencial no repositório.

Isto criará uma corrente de confiança entre a credencial do usuário, assinada pela CA, e a credencial *proxy* emitida pelo MCMS, como mostrado na Figura 3.5.

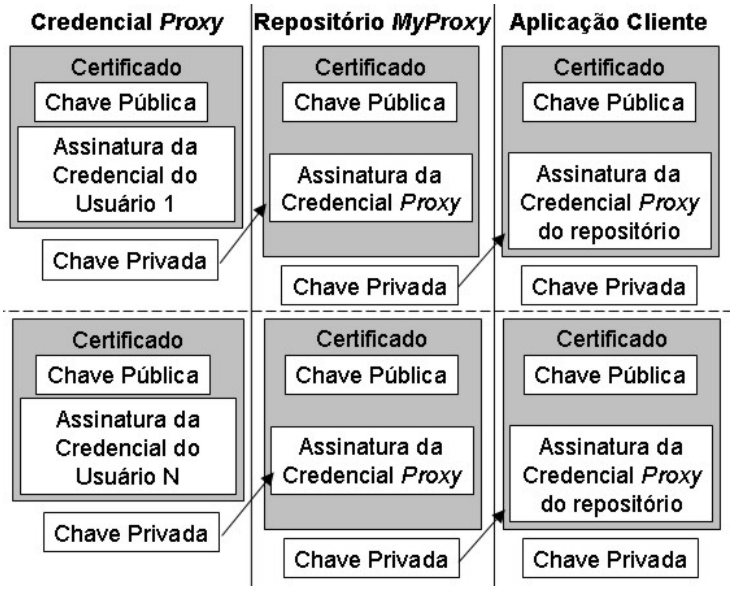

Figura 3.5: Corrente de confiança entre a credencial do usuário e a credencial obtida do repositório do MCMS.

A sequência de passos para obter a credencial armazenada pelo serviço MCMS pode ser vista na Figura 3.6.

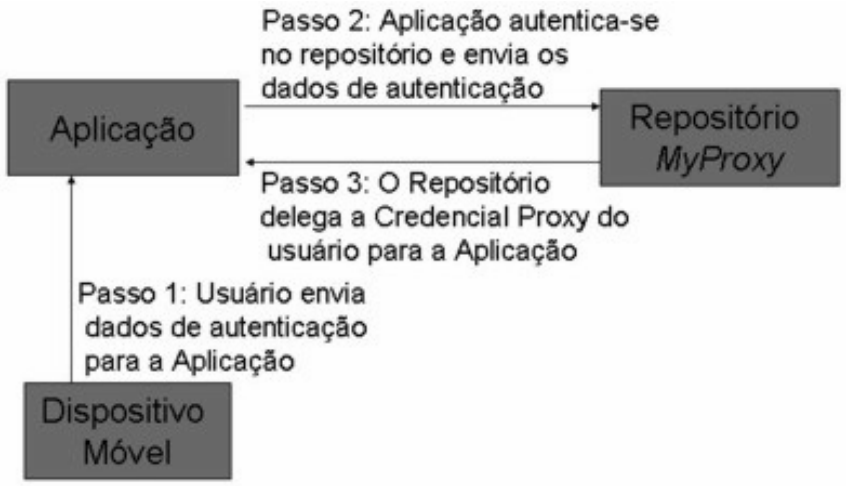

Figura 3.6: Interação entre uma aplicação cliente e o serviço MCMS.

No passo 1 da Figura 3.6, a aplicação cliente (ou usuário) relata para a aplicação na Grade Computacional sua intenção de obter uma credencial *proxy* para autenticar-se. Então, a aplicação cliente envia a senha utilizada para armazenar a credencial no repositório *MyProxy*. No passo 2, a aplicação da grade cria uma credencial *proxy* utilizando a interface *myproxy-logon* e a envia ao serviço *MyProxy* com a senha. No passo 3, o serviço MCMS verifica o controle de acesso ao repositório, consultando o arquivo "myproxy-server.config", e depois assina a credencial e a envia de volta para a aplicação da Grade. Então, esta aplicação autentica-se no computador onde está localizada, configurando a variável de ambiente *X509_USER_PROXY*. A partir daí, a aplicação da Grade Computacional pode atuar em benefício do usuário.

3.1.2 *Community Authorization Service (CAS)*

O Serviço para Autorização de Comunidade *(Community Authorization Service - CAS)* (PEARLMAN; FOSTER, 2002), (PEARLMAN et al., 2003), (GLOBUS SECURITY TEAM, 2009b) foi desenvolvido baseado no framework WSRF. Este serviço é usado para fornecer um controle de acesso aos arquivos e diretórios (recursos) de Grades Computacionais. CAS especifica políticas que informam quais ações um grupo de usuários

pode executar em um recurso ou grupo de recursos.

CAS resolve problemas críticos de autorização em VOs como: escalabilidade, porque o sistema de autorização mantém os registros de usuários e recursos, ao invés de cada usuário ter os registros de todos os demais usuários e recursos, e flexibilidade e expressividade, uma vez que permite acordos das políticas dos usuários e das comunidades serem expressos diretamente neste serviço.

Para o controle de autorização na Web são usados alguns padrões auxiliares, como *Security Assertion Markup Language (SAML)* (MALER et al., 2003). SAML é baseado no padrão XML e é utilizado para troca de asserções de autorização entre sistemas. Uma asserção de autorização é uma informação que determina uma decisão de acesso de um usuário a um recurso (ROSSET et al., 2004).

A implementação de CAS demanda a extensão dos mecanismos do GSI como: 1) criação de Credenciais *Proxy* Restritas (CPR), para permitir granularidade fina na concessão de direitos; 2) uma linguagem de políticas, para especificar os direitos do usuário embutidos na CPR; 3) bibliotecas e interface para programação de aplicações (*Application Programming Interface - API*), para delegar CPR e aplicar as políticas por Provedores de Recurso.

Antes de uma aplicação ou do usuário obter CPR do CAS é necessário cadastrar as políticas de autorização no banco de dados deste serviço (*casDatabase*). Para isso, inicialmente o administrador do CAS cadastra as CAs confiáveis, cadastra os usuários da Grade, informando qual CA assinou os certificados correspondentes e os adiciona em grupos de usuários. Depois disto, o administrador cadastra os *namespaces*, cadastra os objetos (i.e., URL de arquivos e diretórios), informando a qual *namespace* o objeto pertence, e os adiciona em grupos de objetos. Posteriormente, cadastra os tipo de serviços (i.e. tipos de objetos) e as ações que podem ser realizadas nestes tipos de serviço (e.g., ler, escrever, executar). Finalmente, o administrador cadastra as políticas especificando quais ações os grupos de usuário podem executar nos objetos ou grupos de objetos.

A corrente de confiança entre a credencial do usuário e a CPR emitida pelo CAS

pode ser observada na Figura 3.7.

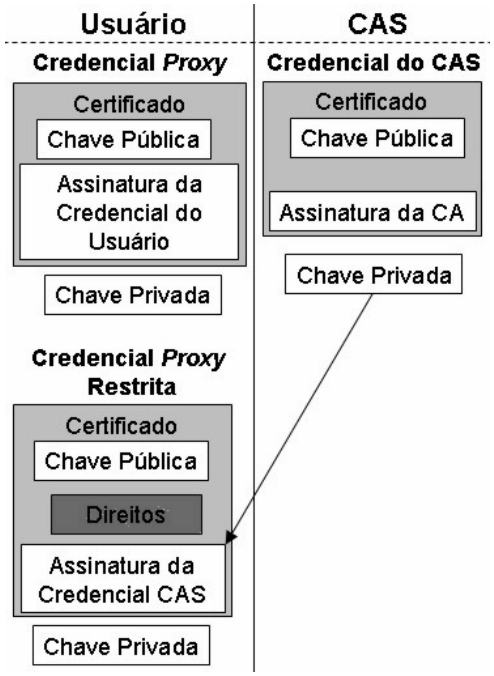

Figura 3.7: Corrente de confiança entre a credencial do usuário e a CPR.

A Figura 3.8 contém os passos para uso do CAS, após o administrador popular o banco de dados. Como se observa, CAS funciona no modo de autorização *Push*, no qual o usuário contacta um sistema de autorização, obtém seus direitos e os envia ao gerenciador de recursos, a fim de utilizar os recursos. Este serviço realiza Controle de Acesso Baseado em Papéis *(Role-based Access Control - RBAC)* (SANDHUF et al., 1994), no qual as políticas são especificadas para grupos de usuários baseada nos papéis estes desempenham no sistema.

No passo 1 da Figura 3.8, a aplicação (usuário) autentica no CAS e requisita seus direitos utilizando a interface *cas-proxy-init*. No passo 2, CAS verifica as políticas e retorna uma CPR delegada à aplicação e que contém asserções SAML com os direitos do usuário, embutidas nela (WELCH et al., 2003b). No Passo 3, a aplicação usa CPR para autenticar-se no Provedor de Recursos usando a interface *cas-wrap*. No passo 4, o Provedor de Recursos libera ou não o acesso ao recurso de acordo com os direitos das assertivas.

No Apêndice A há um exemplo de assertivas SAML contidas em uma CPR.

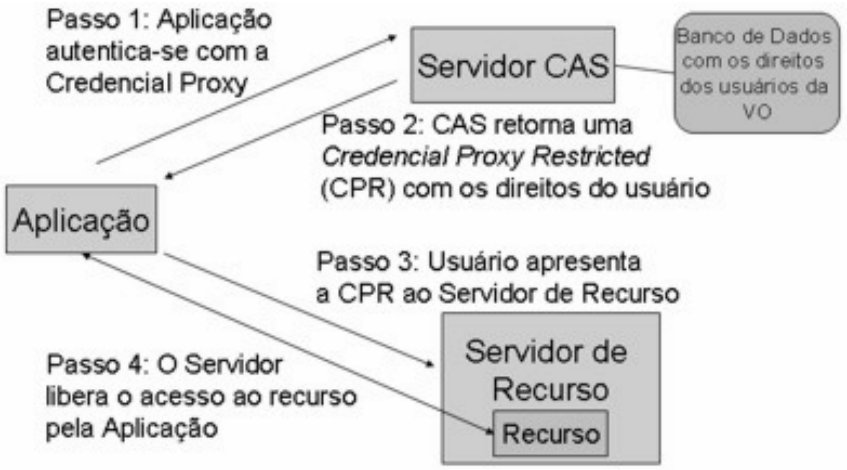

Figura 3.8: Usuário obtendo a CPR e a utilizando em um Provedor de Recursos para acessar o recurso.

O protocolo *Grid File Transfer Protocol (GridFTP)* (RADIC et al., 2007), (GLOBUS TEAM, 2009a) é comumente utilizado para transferir arquivos em Grades Computacionais, mesmo quando o RFT é empregado, funcionando como Provedor de Recursos para estes arquivos. Este serviço foi modificado para entender, analisar e aplicar os direitos contidos na CPR. Assim como o CAS, a maioria dos sistemas de autorização é baseada na especificação *eXtensible Access Control Markup Language (XACML)*.

3.1.3 *Delegation Service (DS)*

O Serviço de Delegação de Credenciais *(Delegation Service - DS)* (GLOBUS SECURITY TEAM, 2009c) é desenvolvido baseado no WSRF. Este serviço facilita a delegação das credenciais dos usuários para outros serviços. Os serviços que queiram utilizar estas credenciais devem estar hospedados no mesmo *container* e, por consequência, no mesmo computador onde o DS está hospedado.

O Serviço para Fabricação de Delegação (*Delegation Factory Service - DFS)* permite armazenar credenciais delegadas na forma de *WS Resource*, para que o DS as delegue a serviços que podem utilizá-las.

No processo de delegação mostrado na Figura 3.9, o DFS também publica seu certificado *proxy* como um WS Resource. O cliente obtém o certificado do DFS, o valida e o autoriza, para posteriormente extrair a chave deste certificado.

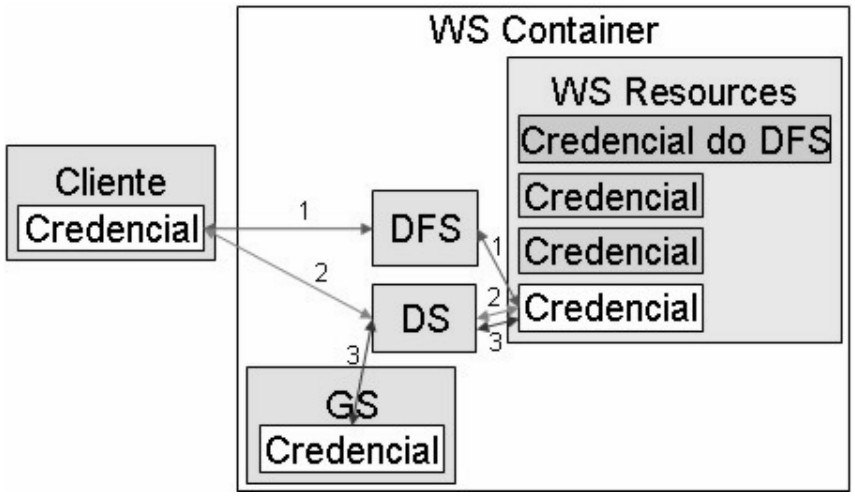

Figura 3.9: Serviço de Delegação de Credenciais.

O cliente então cria um certificado *proxy* usando essa chave pública para então invocar a interface remota no DS para delegar a credencial. Por fim, o cliente passa seu certificado para o DFS, que retorna um objeto *EndPointReference (EPR)* que serve para identificar a credencial delegada. O EPR pode ser distribuído aos serviços que o usuário gostaria de delegar os seus direitos.

O usuário pode precisar atualizar a credencial delegada, e isto deve ser feito antes que esta expire. Se o usuário não atualizar a credencial e esta expirar, ela será apagada e o EPR não poderá ser reutilizado. DS permite aos clientes atualizarem remotamente às suas credenciais delegadas. A renovação de Credencial segue os mesmos passos de delegação acima, exceto a etapa final, em que o cliente entra em contato com seu WS-Resource invés do DFS.

Os serviços que estão aptos a utilizar as credenciais do DS podem registrar-se no DS, a fim de receber notificações quando as credenciais são modificadas. Para isso este serviço precisa criar um objeto *listener* que implementa a interface `org.globus.delegation.DelegationRefreshListener` e registrar o *listener* no DS. Após o registro, DS verifica se a identidade do delegador equivale à identidade passada e uma vez que o *listener* tenha sido autorizado, a credencial delegada é disponibilizada para o serviço.

A interface *globus-credential-delegate* pode ser utilizada para delegar uma creden-

cial, *globus-credential-refresh* para atualizar uma credencial delegada e *wsrf-destroy* para remover uma credencial.

3.1.4 *Authentication and Authorization Security Descriptor Framework (AASDF)*

O *Framework* Descritor de Segurança para Autenticação e Autorização *(Authentication and Authorization Security Descriptor Framework - AASDF)* ,(GLOBUS SECURITY TEAM, 2009f), (GLOBUS SECURITY TEAM, 2009g), (GLOBUS SECURITY TEAM, 2009a) aplica as políticas de autorização configuradas no sistema servidor *(GS Provider)* e no sistema cliente *(GS Client)*. AASDF permite aos desenvolvedores configurar uma cadeia de mecanismos de autorização de forma programática ou declarativa, utilizando arquivos para configuração de segurança. Este *framework* também permite a ligação de novos esquemas de autorização (além dos fornecidos com o *framework*). Além disso, AASDF permite que esta configuração seja feita no cliente, no serviço ou no *container*.

Os métodos de autenticação digital, cujo objetivo consiste em assegurar que o usuário é quem diz ser, utilizado na transmissão segura das mensagens realizadas pelo AASDF são:

- *None*, indicando que a autenticação não é requerida;
- *GSISecureMessage*, que indica que será utilizada transferência segura às mensagens individuais na camada de aplicação, utilizando-se para isso o padrão *WS-Security*. Este método possui outras duas propriedades conhecidas como *integrity*, que garante a integridade das mensagens assinando-as, e *privacy*, que garante a privacidade das mensagens cifrando-as e assinando-as.
- *GSISecureConversation*, que indica que será utilizada uma sessão de comunicação segura, na camada de aplicação, utilizando-se para isso o padrão *WS-SecureConversation*. Este método também suporta as propriedades *integrity* e *privacy*;
- *GSITransport*, indicando que será utilizado o transporte seguro das mensagens,

utilizando para isso o protocolo *HyperTranfer Transport (HTTP)* sobre Transport Layer Security (TLS). Este método também suporta *integrity* e *privacy*.

Nos GSs Providers, as configurações das autorização requeridas, nos arquivos de segurança podem ser do tipo:

- *None*, indicando que nenhuma autorização será realizada;
- *Self*, em que o *GS Client* será autorizado a usar o serviço se possuir a identidade (DN) igual a do serviço;
- *GridMap*, indicando que o *GS Client* será autorizado se sua identidade é listada no arquivo de autorização "grid-mapfile";
- *Identity*, no qual o *GS Client* será autorizado se sua credencial for igual a uma identidade especificada; e
- *Host*, indicando que o *GS Client* será autorizado se este apresentar uma "credencial de computador", cujo o mesmo é identificado pelo *hostname*, que seja equivalente a um *hostname* esperado, ou seja, garante-se que a requisição venha de um computador particular.

No *GS Client*, a segurança é feita de forma programática, através da definição de propriedades de segurança exigida nas Classes *Stubs*, utilizadas para realizar a invocação da operação de GS, ou por configuração do descritor de disponibilização de WSs *(Web Services Deployment Descriptor - WSDD) client-config.wsdd*, utilizado pela aplicação cliente.

Há também uma configuração de autorização no *GS Client*, indicando as exigências que este impõe, para contactar um *GS Provider*, são elas:

- *None*, indicando que nenhuma autorização será realizada;
- *Self*, no qual o *GS Client* poderá chamar o serviço se a identidade do serviço for igual a identidade do *GS Client*;

- *Identity*, em que o *GS Client* somente permite enviar requisições ao *GS Provider*, se este possuir uma identidade igual a especificada pelo Client; e

- *Host*, indicando que o *GS Client* estará apto a acessar o *GS Provider* se este possuir uma credencial de computador. Para isso, o *GS Client* verifica se o *hostname* da credencial é igual ao *hostname* esperado.

As autorizações de *GS Clients* são independentes e separadas da autorização dos *GS Providers*.

A Figura 3.10 mostra os elementos utilizados pelo AASDF para auxílio a autorização e autenticação na Grade.

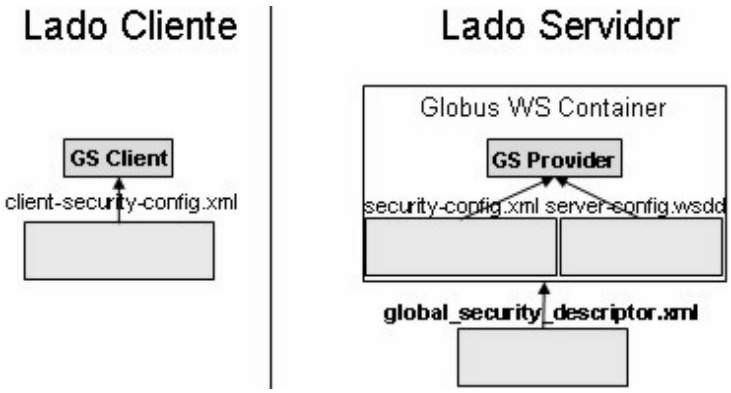

Figura 3.10: Descritores de configuração de segurança do AASDF.

O arquivo "cliente-security-config.xml" do *GS Client* contém a configuração do tipo de autenticação e de autorização que será utilizada na comunicação com o *GS Provider*.

O *Globus WS Container (GWSC)* contém o arquivo `$GLOBUS_LOCATION/etc/globus_wsrf_core/globus-security-descriptor.xml`, para configuração dos mecanismo de autenticação e autorização no nível de *container*. Caso este arquivo esteja em outro diretório, é necessário informá-lo ao container através do parâmetro '*container-Doc*', para que seja utilizada a segurança no nível de *container*. Pode-se informar o mecanismo de autorização que o *container* deve utilizar, com uso do parâmetro '-z', informando-se os valores '*self*', '*host*', '*none*' ou a identidade esperada ('*identity*'). Há também o parâmetro '-*nosec*', que pode ser passado para o container para desabilitar

a segurança no nível de transporte.

GS Providers contêm um arquivo WSDD localizado em "`$GLOBUS_LOCATION/etc/$service_gar_dir/server-config.wsdd`". Este arquivo contém os parâmetros, 'allowedMethods', que informa quais os métodos que podem ser acessados por meio de *GS Clients*. Esta informação é útil para um correto controle de acesso às operações de *GS Providers*, pois as as operações públicas do GS podem ser executadas, mesmo que não estejam formalmente definidos no WSDL.

GSs contêm também outro arquivo de segurança chamado "security-config.xml". Este arquivo contém o parâmetros '*auth-method*', indicando quais os tipos de autenticação, mostradas acima, são requeridos. Há outro parâmetro '*authz*' que indica o tipo de autorização requerido. Também há o parâmetro chamado '*run-as*', que indica qual a identidade que o serviço assumirá durante a execução. O GS pode executar com a identidade do *GS Client*, com sua própria identidade, com a do *container*. Caso o serviço também não tenha uma identidade, será assumida a identidade do *container*. O container utiliza a identidade da "credencial de computador".

O uso do arquivo "globus-security-descriptor.xml" no *GS Provider* é opcional, assim como o arquivo "cliente-security-config.xml" para configuração da segurança no *GS Client*.

Resumo do Capítulo 3

Este Capítulo apresentou o *Globus Toolkit* e seus principais serviços de segurança.

A segurança neste *middleware* é feita principalmente utilizando-se a infraestrutura GSI. GSI é formada por diversos serviços que possuem finalidades específicas como, MCMS, que gerencia credenciais, CAS, que realiza controle de acesso à recursos, DS, para delegação de credenciais, e AASDF, no auxílio a autenticação e autorização na grade.

No próximo Capítulo serão mostrados alguns trabalhos correlatos, apresentando as principais arquiteturas e mecanismos de segurança para grades computacionais, baseadas no *Globus Toolkit*.

CAPÍTULO

4

Mecanismos Adicionais de Segurança para Grades

"A segurança é indivisível. Ou existe igual para todos ou não há segurança para ninguém."

MIKHAIL GORBACHEV

Existem diferentes arquiteturas para o tratamento de questões de segurança em grades computacionais, tais como a *Security Architecture for Open Grid Services (SAOGS)*, a *caGrid Security Architecture (caGSA)* e a *Authentication and Authorization Architecture for Grid Computing (3AGC)*. Também há mecanismos de segurança para Grades que expandem as funcionalidades da GSI tais como, *GridShib*, *Virtual Organization Membership Service (VOMS)*, *PrivilegE and Role Management Infrastructure Standards (PERMIS)*, Akenti e *Multipolicy Authorization Framework (MAF)*.

4.1 *Security Architecture for Open Grid Services (SAOGS))*

A Arquitetura de Segurança para GSs Abertos *(Security Architecture for Open Grid Services - SAOGS)* (NAGARATNAM; other, 2002) especifica inúmeros requisitos de segurança para utilização de GSs. Esta arquitetura é dividida em camadas, que fornecem integração com os mecanismos de segurança existentes para Grades, interoperabilidade entre diferentes mecanismos e múltiplos domínios administrativos, e o relacionamento de confiança entre os usuários da Grade, para que os serviços e recursos possam ser acessados a partir de outros domínios.

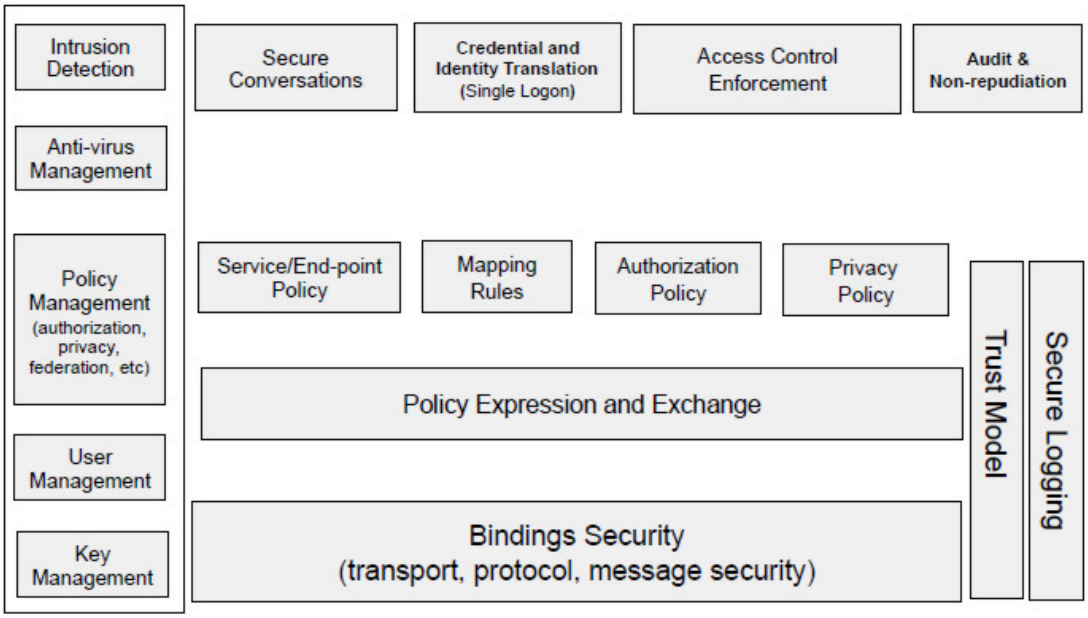

Figura 4.1: Arquitetura de Segurança para GS Abertos. Fonte: (NAGARATNAM; other, 2002).

SAOGS lida com os principais problemas de segurança envolvendo grades computacionais, como os apresentados na Figura 4.1:

1. *Authentication*, que provê o uso de múltiplos mecanismos de autenticação;
2. *Delegation*, que permite que uma entidade delegue a outra seus direitos de acesso sobre recursos;
3. *Single Logon*, que permite ao usuário autenticado, realizar o logon em outros

computadores automaticamente;

4. *Credential Lifespan and Renewal*, que notifica o usuário quando sua credencial delegada está expirando;

5. *Authorization*, que permite o controle de acesso aos GSs;

6. *Privacy*, que provê o uso de políticas de privacidade para o usuário;

7. *Confidentiality*, que protege a confidencialidade das mensagens no nível de transporte e de aplicação;

8. *Message Integrity*, que assegura que não ocorram modificações não autorizadas nas mensagens;

9. *Policy Exchange*, que permite a troca de políticas no estabelecimento do contexto seguro de transferência de dados;

10. *Secure Logging*, que registra todos os eventos a fim de fornecer auditoria e não repúdio, inclusive os envolvendo os mecanismos de segurança;

11. *Assurance*, que permite definir níveis de segurança.

12. *Manageability*, que é responsável pela gerência de identidades, políticas e chaves;

13. *Firewall traversal*, que fornece mecanismos para troca de informações entre dominios protegidos por firewall; e

14. *Securing the OGSA infrastructure*, que especifica os requisitos de segurança da Infraestrutura Aberta para Segurança de Grades *(Open Grid Security Infrastructure - OGSI)*.

4.2 CaGrid Security Architecture (caGSA)

A Grade de Informática Biomédica para cancer *(cancer Biomedical Informatics Grid - caBIG)* (LANGELLA, 2005) envolve instituições e indivíduos voluntários para pesquisas

sobre câncer por meio de compartilhamento de dados e ferramentas. A arquitetura de Segurança para Grades caBIG *(caGrid Security Architecture - caGSA)* fornece confidencialidade, para que apenas o destinatário tenha ciência das informações, integridade, para garantir que as informações não sejam alteradas, disponibilidade, segurança física, controle de acesso e rastreabilidade aos dados manipulados por caBIG.

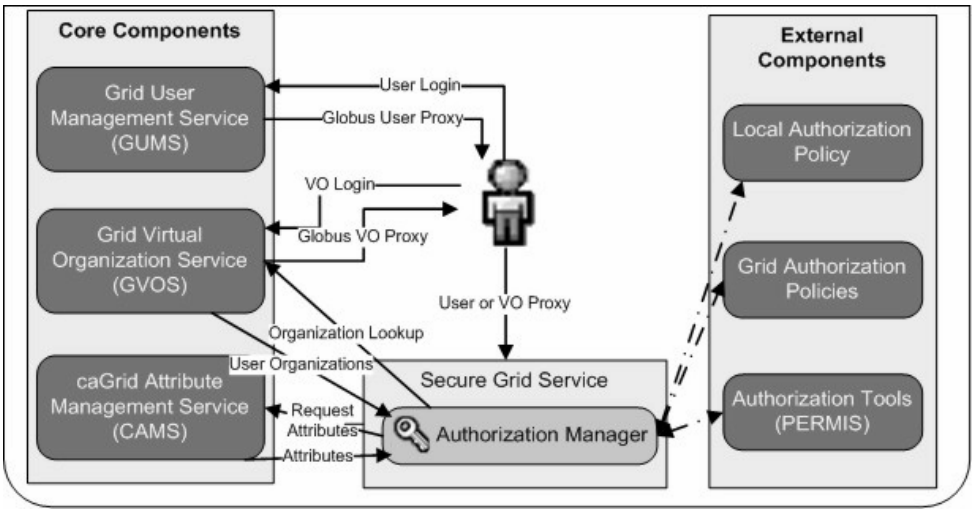

Figura 4.2: Grade de Informática Biomédica para câncer. Fonte: (LANGELLA, 2005).

caGSA fornece autorização a GS no nível de operações e possui os serviços mostrados na Figura 4.2:

- *Grid User Management Service (GUMS)* para mapeamento de identidade de grade, que é necessário quando se utilizada diferentes mecanismos de autenticação entre a VO original e a VO que terá o recurso utilizado;
- *caGrid Attribute Management Service (CAMS)* que gerencia os atributos com os papéis do usuário, visando autorização;
- *Grid Virtual Organization Service (GVOS)* que gerencia um conjunto de VOs permitindo que o usuário de uma VO requisite um proxy de outra;
- *Authorization Manager* que é chamado a partir da implementação do GS para, com auxílio de mecanismos externos, retornar uma decisão de autorização sobre um recurso.

4.3 *Authentication and Authorization Architecture for Grid Computing (3AGC)*

A Arquitetura de Autenticação e Autorização para Grade Computacional *(Authentication and Authorization Architecture for Grid Computing - 3AGC)* (JUNG et al., 2005) fornece fina granularidade no controle de acesso a GSs, permitindo associar usuários a operações. Sua funcionalidade é dinâmica, pois não exige o reinício do GS quando as políticas de segurança são modificadas. 3AGC utiliza o conceito de Aspectos para o controle de acesso as operações dos GSs, entrecortando os métodos do *Globus WS Container (GWSC)* que realizam a chamada ao *GS Provider*.

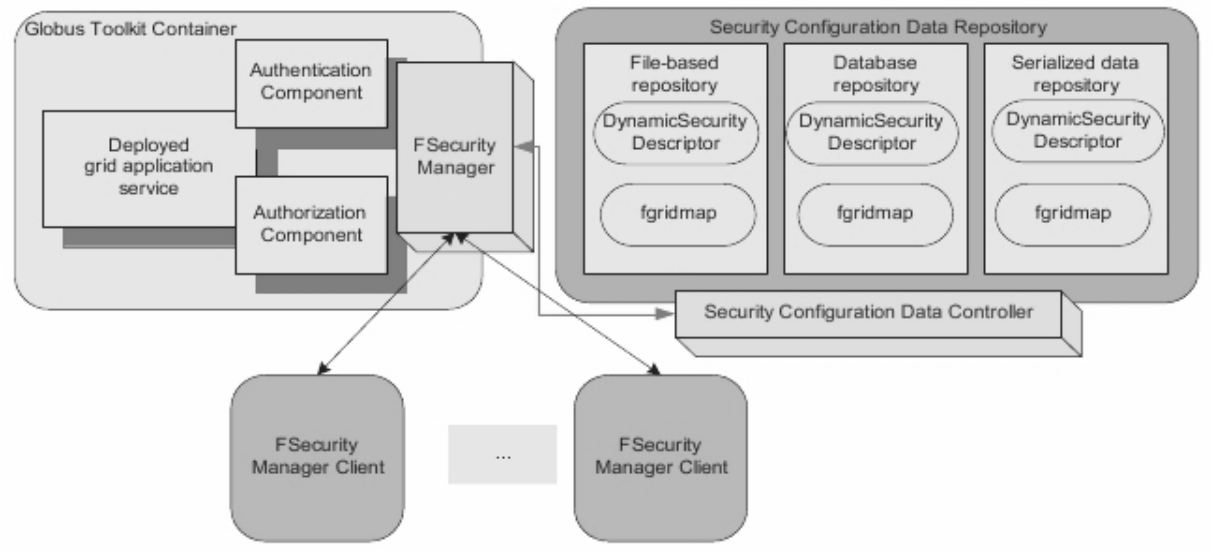

Figura 4.3: Arquitetura de Autenticação e Autorização para Grade Computacional. Fonte: (JUNG et al., 2005).

Esta arquitetura é composta pelos módulos da Figura 4.3:

- *FSecurity Manager* que intercepta as requisições de autenticação e autorização e evita que o GS seja reiniciado caso as políticas de segurança sejam alteradas. Este modulo também fornece fina granularidade de autorização relacionando usuários e operações de GS, ao invés do uso de gridmap-file que especifica quem pode usar os GSs;

- *Security Configuration Data Controller* que gerencia as informações de políticas de segurança armazenadas em *eXtesible Markup Language (XML)*, arquivo ou banco de dados;

- *FSecurity Manager Client* que permite que administradores de GS controlem e gerenciem as políticas de segurança remotamente, via Web.

4.4 *GridShib*

Shibboleth (WELCH et al., 2005) é uma autoridade de distribuição de atributos *(Attribute Authority - AA)* para controle de acesso a recursos web entre instituições educacionais, compartilhados via *browsers*. *Shibboleth* também fornece anonimato dos usuários com o uso de pseudônimos, sendo que apenas a AA sabe a qual identidade o pseudônimo se refere. *GridShib* (WELCH et al., 2005) é uma infraestrutura de autorização que integra o *Shibboleth* ao GT, permitindo que um provedor de recursos do *Globus Toolkit* requisite, de forma segura, atributos do usuário ao provedor de identidade do *Shibboleth*. Os atributos de autorização são expressos em assertivas *Security Assertion Markup Language (SAML)* e possuem um curto tempo de vida.

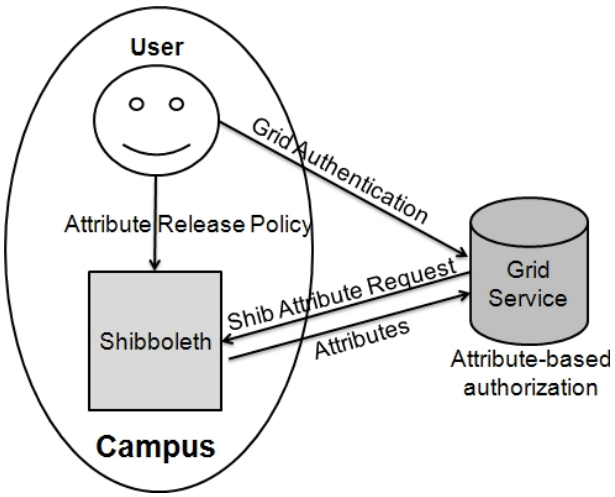

Figura 4.4: Interação entre o *Shibboleth* e os recursos da Grade, através do *GridShib*. Fonte: (WELCH et al., 2005).

GridShib é formado pelos componentes da Figura 4.4: 1) *Assertion Transmission*,

que permite transmitir as asserções do *Shibboleth* para o serviço da grade e faz decisão de autorização em tempo de execução na Grade; 2) *Attribute Authority*, para permitir a descoberta de AA apropriadas para o usuário; 3) *Distributed Attributed Administration*, para gerenciar subconjuntos de atributos fornecidos pelo serviço *Shibboleth*; 4) *Pseudonymous Interaction*, para estender às Grades a interação de pseudonimo fornecida pelo *Shibboleth*; e 5) *Authorization*, que fornece um mecanismo que é integrado ao *Globus Toolkit* para utilizar os atributos.

4.5 *PrivilegE and Role Management Infrastructure Standards (PERMIS)*

A Infraestrutura Padrão de Gerenciamento de Papéis e Privilégios *(PrivilegE and Role Management Infrastructure Standards - PERMIS)* (CHADWICK; OTENKO, 2002) pode ser utilizada para controle de acesso a recursos em sistemas que utilizam *Public Key Infrastructure (PKI)* para autenticação. PERMIS utiliza certificados de atributo *(Atribute Certificate - AC)* na autorização, a fim de ligar os papéis de usuários a um ou mais atributos contendo os direitos destes papéis. Este sistema de autorização utiliza o controle de acesso baseado no RBAC hierárquico, em que papéis podem herdar os direitos de outros papéis. Os atributos dos certificados são descritos em XML com uma estrutura (*schema*) definida por PERMIS. As políticas de autorização e os papéis do AC para usuário são armazenados em um repositório *Lightweight Directory Access Protocol (LDAP)*[1].

PERMIS é formado pelos subsistemas de alocação e de verificação de direitos, como pode ser visto na Figura 4.5. O subsistema de alocação armazena as políticas em um diretório LDAP, assina digitalmente as políticas de autorização e devolve ACs com os direitos dos usuários. O verificador de direitos consulta este diretório e realiza a autenticação e autorização dos usuários. Este verificador possui os módulos *Access Control Enforcement Function (AEF*) e *Access Control Decision Function (ADF)*. No uso do PERMIS, os seguintes procedimentos acontecem: o usuário acessa um recurso

[1] `http://www.ldap.org.br`.

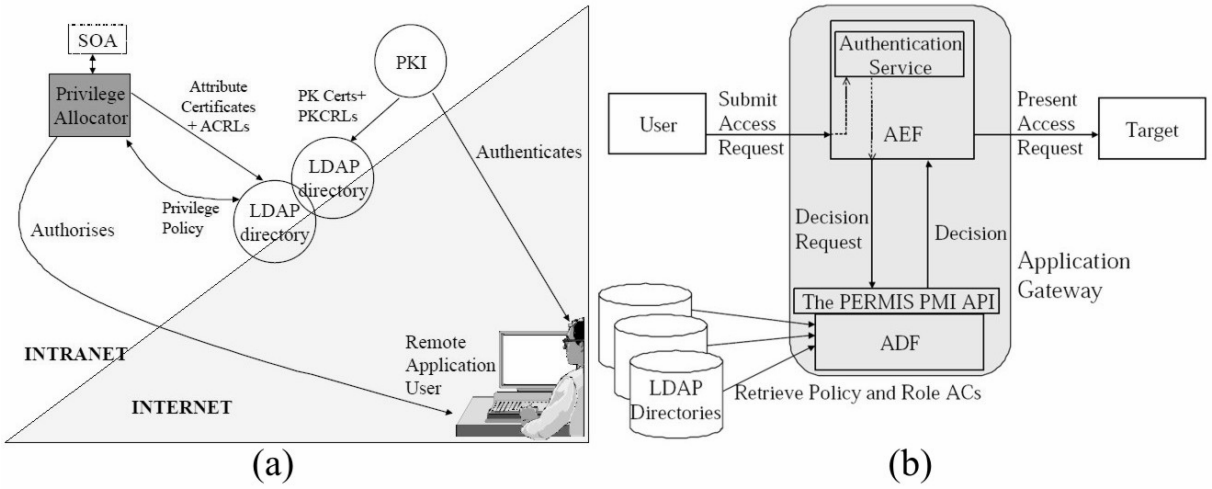

Figura 4.5: Sistema PERMIS. Fonte: (CHADWICK; OTENKO, 2002). (a) Subsistema de Alocação de Direitos. (b) Subsistema de verificação de direitos.

via uma aplicação; AEF autentica o usuário e então questiona ADF se o usuário tem permissão de realizar a ação desejada no recurso; ADF acessa o diretório LDAP e retorna as políticas de autorização e os papéis do AC para usuário; então AEF realiza a decisão de autorização com base nestas informações.

4.6 *Akenti*

O sistema de autorização *Akenti* (THOMPSON et al., 1999) utiliza certificados X.509 para o controle de autenticação, ACs para o controle de autorização e certificados de condição de uso *(Use-Condition Certificate - UCC)* para expressar as políticas de segurança. Akenti foi desenvolvido para o controle de acesso a sites e recursos web, e não objetiva gerenciar VOs como a maioria dos sistemas de autorização voltados para Grades. No ambiente com *Akenti*, cada provedor de recursos é responsável por associar as identidades dos usuários às suas permissões. No *Akenti* deve existir pelo menos um UCC por usuário para acesso ao recurso. *Akenti* registra o *log* dos acessos dos usuários e mantém *cache*[2] dos ACs e UUCs buscados de outros sites.

A Figura 4.6 mostra a arquitetura do Akenti. O principal elemento desta arquite-

[2]Armazenamento temporário de dados que possuem uma grande probabilidade de serem utilizados novamente.

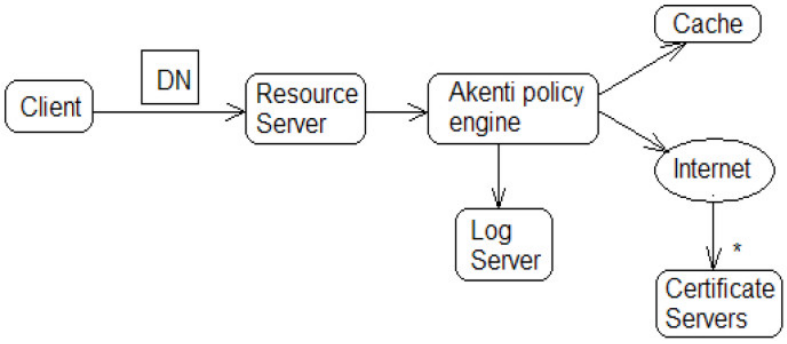

Figura 4.6: Modelo de Autorização do Akenti. Fonte: (THOMPSON et al., 1999).

tura é o *Akenti Policy Engine (APE)*. APE é responsável por verificar a autenticidade e validade dos UCC, registrar o *log* dos acessos dos usuários e manter um cache dos ACs e UUCs buscados de outros sites. Inicialmente, o usuário autentica-se no provedor de recursos e este chama o APE, que retorna a decisão de autorização com base nos papéis do usuário e das políticas do sistema. Deve existir pelo menos um UCC por usuário; caso nenhum seja apresentado ao APE, este nega o acesso.

4.7 *Multipolicy Authorization Framework (MAF)*

O Framework de Autorização para Políticas Múltiplas *Multipolicy Authorization Framework - MAF)* (LANG et al., 2006) suporta vários mecanismos de autorização e múltiplas políticas de autorização como GridMap, *Access Control List (ACL)*, CAS e SAML, etc. Este *framework* é baseado no padrão *eXtensible Access Control Markup Language (XACML)*. MAF é usado por desenvolvedores de sistemas de autorização que o estendem e o usam em seus sistemas personalizados.

Para uma aplicação utilizar o framework de autorização é necessário que ela crie o próprio mecanismo de decisão, herdando o *MasterPDP* do framework, mostrado em detalhes na Figura 4.7. O *Point Administration Point (PAP)* é específico de cada mecanismo de autorização, sendo que as políticas de acesso são definidas de várias formas e linguagens. O framework de autorização fornece a implementação do PAP para alguns comuns mecanismos de autorização, como pode ser visto na Figura 4.7. A implementação do *Policy Enforcement Policy (PEP)* faz parte do framework e é comum

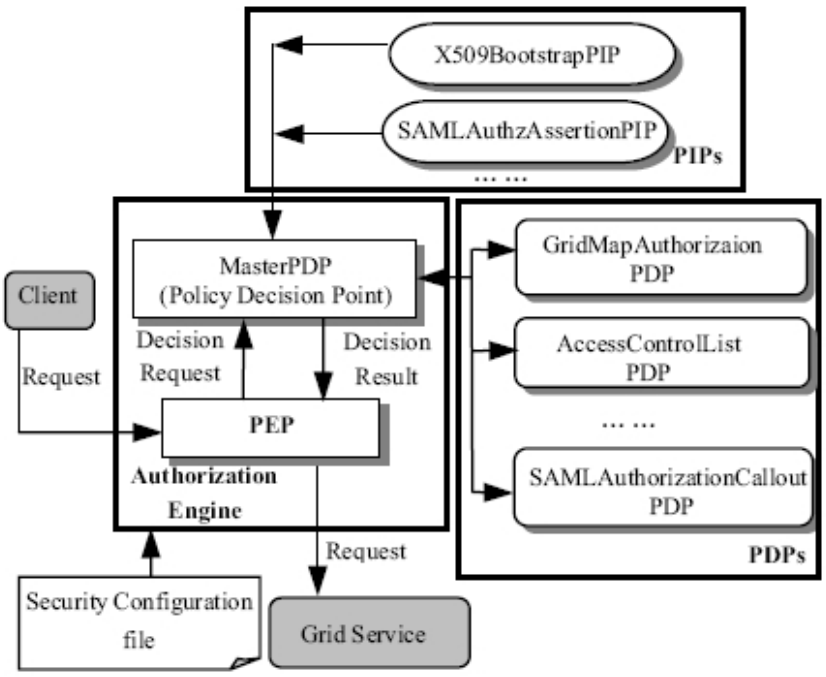

Figura 4.7: Framework de Autorização. Fonte: (LANG et al., 2006).

a todos os mecanismos.

4.8 *Virtual Organization Membership Service (VOMS)*

O Serviço de Sociedade para Organizações Virtuais *(Virtual Organization Membership Service - VOMS)* (ALFIERI et al., 2003) é um sistema de autorização baseado em AC. Estes certificados contêm papéis do usuário para auxiliar o controle de acesso, funcionando independente dos mecanismos de autenticação. O usuário gera os ACs com a interface *voms-proxy-init*, sendo que as informações de autorização destes certificados são entendidas apenas por servidores VOMS. Provedores de recursos as ignoram caso não as entendam, mantendo assim a compatibilidade com versões anteriores. Um usuário pode contactar inúmeros servidores VOMS e obter vários ACs. VOMS mantém todos os registros em um banco de dados. Quando eles são apagados, apenas uma *flag* é configurada, não realizando a exclusão. Desta forma, podem-se informar quais grupos o usuário estava em um momento específico.

VOMS é formado por 4 partes, como observado na Figura 4.8: 1) *User Server*, que

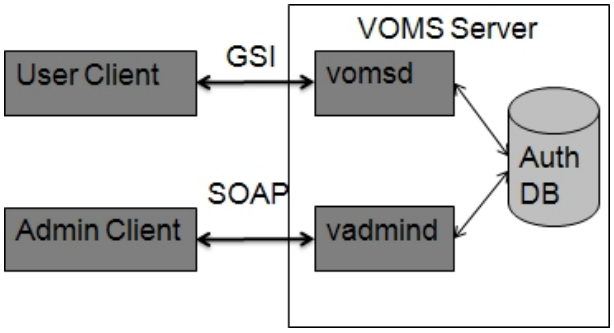

Figura 4.8: Serviço de Sociedade para Organizações Virtuais. Fonte: (ALFIERI et al., 2003).

recebe as requisições dos usúarios; 2) *User Client*, que contacta o servidor, apresenta o certificado do usuário e obtém uma lista de grupos, papéis e direitos; 3) *Administrator Client*, usado para gerenciar os usuários, grupos e papéis; e 4) *Administrator Server*, que aceita a requisição dos usuários e atualiza o banco de dados do VOMS. O módulo Administrador é usado para gerenciar as informações do Banco de dados e fornece as funcionalidades básicas para os clientes, os métodos para administrar o banco de dados, e controle de *log* e de responsabilidade dos acessos dos usuários.

Resumo do Capítulo 4

Este Capítulo apresentou algumas arquiteturas que estendem OGSA, incluindo SAOGS, caGSA e 3AGC, além de mecanismos de segurança que estendem as funcionalidades de GSI como *GridShib*, PERMIS, *Akenti*, MAF e VOMS.

A arquitetura SAOGS fornece autorização sobre GS, caGSA fornece autorização sobre as operações de GS e 3AGC que utiliza aspecto para autorização de GS.

O mecanismo *GridShib* é usado para fornecer controle de acesso e privacidade aos usuários, VOMS e PERMIS utilizam certificados de atributos para gerenciar os recursos da grade, *Akenti* utiliza certificados de condição de uso para especificar políticas de segurança e MAF suporta vários sistemas de autorização para controle de acesso a recursos.

No próximo Capítulo será apresentada a arquitetura proposta para a segurança de GSs.

CAPÍTULO

5

Uma Arquitetura de Segurança para *Grid Services*

"Segurança é, eu diria, nossa maior prioridade (...), se não resolvermos estes problemas de segurança, daí as pessoas recuarão."

BILL GATES

Grid Service Security Architecture (GSSA) é uma arquitetura de segurança que visa fornecer um controle de acesso às operações disponibilizadas por *GS Providers*, permitindo especificar quais operações os usuários têm o direito de executar, baseados em seus papéis. Isso é feito de forma que o *GS Provider* não precise ter ciência da segurança introduzida e sem que os desenvolvedores de aplicações clientes *(GS Clients)* precisem preocupar-se com estas questões, podendo focar-se nas funcionalidades do sistema. GSSA fornece também um controle sobre o acesso aos recursos dos *GS Providers* utilizado o mesmo mecanismo de autorização citado

anteriormente.

Além disso, GSSA especifica outras atribuições importantes para o controle de acesso. Isso inclui a identificação do usuário através de mecanismos de autenticação, a fim de associar suas identidades a seus diretos, a transferência segura das mensagens trocadas entre GSs, o registro de *logs* da autenticação e do acesso às operações, para permitir uma investigação e responsabilizar os usuários por suas ações e identificar possíveis falhas de segurança, e a interoperabilidade entre *WS Clients* e *GS Providers* utilizando uma infraestrutura de segurança.

Estas funcionalidades podem ser melhor observadas na Figura 5.1.

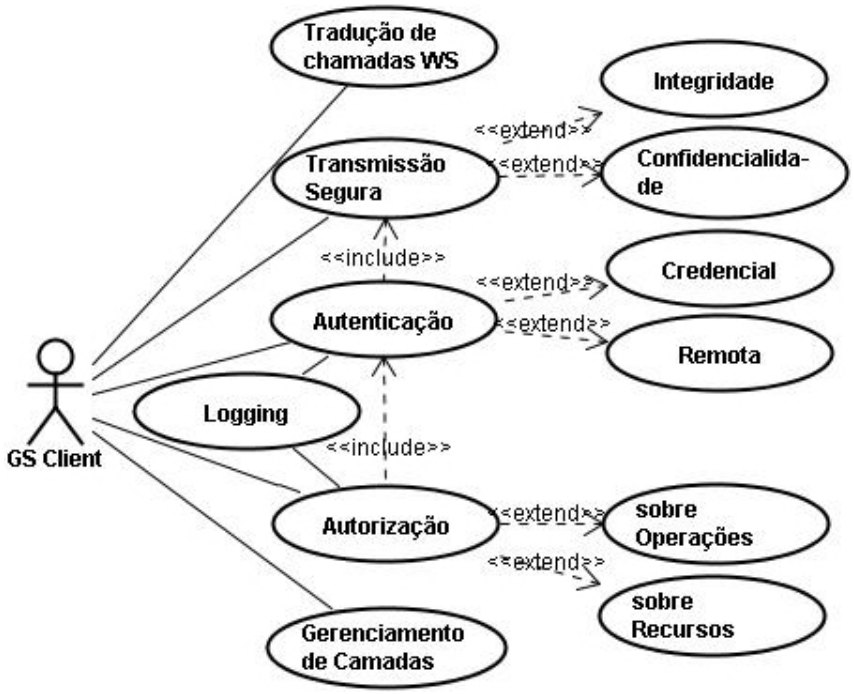

Figura 5.1: Funcionalidades da GSSA.

Esta arquitetura foi projetada baseada nos padrões previstos no padrão OGSA e é apresentada na figura 5.2. Como WSRF fornece uma implementação em Java dos GS padronizados por OGSA, as implementações da arquitetura GSSA devem ser desenvolvidas utilizando este *framework*.

A autenticação é responsável por identificar um usuário univocamente e garantir o não repúdio através de registro dos acessos, de forma que o emissor de uma men-

sagem, ou quem realizou alguma tarefa num sistema computacional, não pode negar sua autoria, haja vista que apenas ele possuía os mecanismos para autenticar-se no sistema.

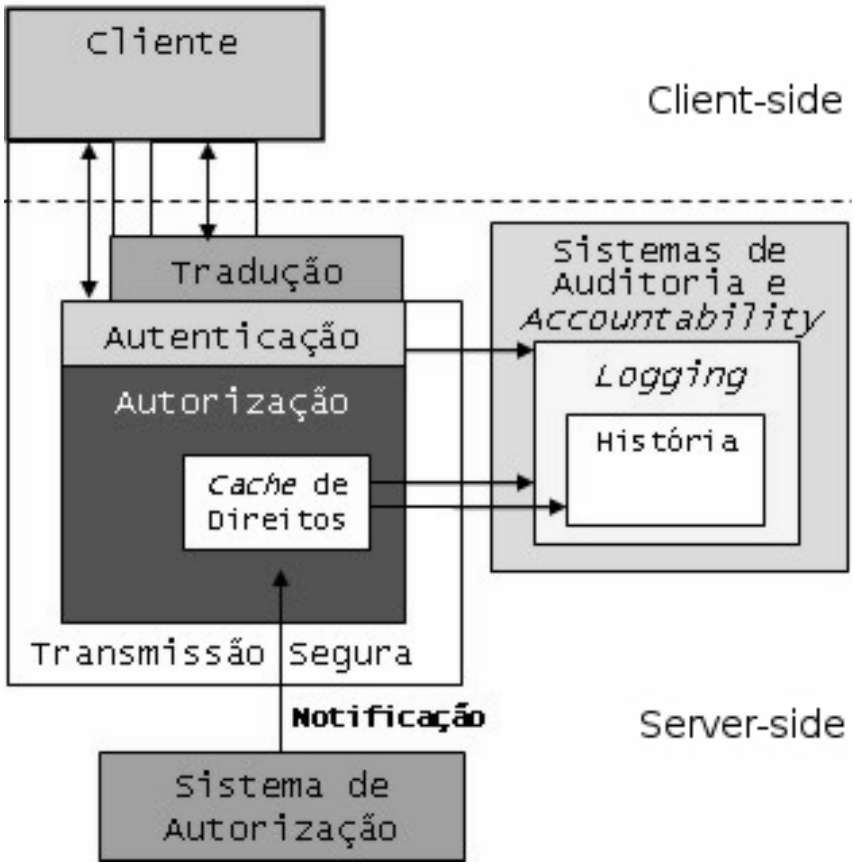

Figura 5.2: Arquitetura de segurança proposta para GS.

A camada **Autenticação** é responsável por identificar o usuário na grade computacional. Isto é feito com o uso de credencias *proxy*, no modelo *Single Sign-On (SSO)*, em que o usuário participa do procedimento de autenticação uma única vez. Mesmo que o usuário não possua as credenciais no *GS Client*, esta camada fornece uma interface baseada em nome e senha, de forma que o usuário possa obter as credenciais necessárias a esta autenticação. Após a autenticação, esta camada realiza uma consulta às permissões do usuário no "Sistema de Autorização" sempre que for necessário. Estes direitos são armazenados no módulo "*Cache* de Direitos" da camada "Autorização".

Autorização consiste em garantir que apenas usuários autorizados utilizem recursos protegidos de um sistema computacional. Os recursos computacionais podem ser arquivos, páginas web, serviços, etc. A autorização define quais recursos os usuários podem utilizar, ou quais ações os usuários podem executar sobre estes recursos. O processo de autorização acontece após a autenticação do usuário no sistema.

A camada **Autorização** é a mais importante da arquitetura, pois esta é quem de fato realiza o controle de acesso às operações e aos recursos dos GSs. A estrutura ***Cache* de Direitos** é consultada todas as vezes que o usuário precisa executar uma operação. Quando há mudança nos direitos do usuário, "*Cache* de Direitos" é notificado (mecanismo **Notificação**) e realiza a busca e o armazenamento dos novos direitos. Antes de "*Cache* de Direitos" ser alterado, o módulo **História** armazena os valores correntes em arquivo, possibilitando assim determinar os direitos que o usuário tinha em um momento específico da operação do sistema.

Um sistema de autorização gerencia as políticas de autorização, recebe requisições e retorna uma decisão quando requisitado. Este sistema gerencia os usuários, os recursos e as ações envolvidas.

Sistema de Autorização deve ser implementado baseado na OGSA para suportar o padrão *WS Notification*. O uso deste padrão permite que o "Sistema de Autorização" informe ao mecanismo que implementa GSSA, quando as políticas referentes a um ou mais usuários forem modificadas, para que este mecanismo requisite uma nova decisão de autorização contendo os atuais direitos do usuário. Assim, a sincronia dos direitos do usuário é mantida, o que não ocorre comumente nos sistemas de autorização.

Logging consiste na criação de arquivos texto com informações sobre eventos num sistema, envolvendo o usuário, o seu comportamento e o comportamento do próprio sistema. *Logging* auxilia **sistemas de auditoria**, permitindo que os administradores conheçam o comportamento do sistema no passado, facilitando o diagnóstico de problemas, procurando saná-los. Além disso, favorece também **sistemas de *accountability***, permitindo que os administradores punam os membros de uma organização

por má conduta no sistema, uma vez que estes prestam contas sobre o uso dos recursos do sistema, mesmo sem estarem cientes.

O módulo ***Logging*** cria registros na autenticação e em cada chamada das operações do *GS Provider*, sejam elas permitidas ou negadas. O arquivo de *log* deve conter uma *flag* representando o tipo de evento realizado, o identificador do usuário logado, o horário da realização do evento, o evento executado e uma descrição do mesmo. Os tipos de eventos podem ser *login*, *logout*, acesso bem sucedido, acesso mal sucedido e exceção.

Mecanismos de tradução como (HE et al., 2006), (MOROHOSHI et al., 2007), permitem o acesso às operações de *Grid Services (GS)* a partir de *Web Services (WS)*. Contudo, estes mecanismos não levam em consideração questões de segurança neste acesso, nem as implicações dos mecanismo de segurança já existentes na Grade Computacional, ou mesmo o uso destes mecanismos.

A camada **Tradução** foi prevista para que os *WS Clients* possam acessar as operações de GS Provider*s*, de forma segura. Para isso, deve-se levar em consideração que GSs possuem recursos persistentes e acessíveis, enquanto WSs não. Então, é conveniente existir uma forma de representar no *WS Client* qual o recurso a ser utilizado. Uma vez que o *WS Client* acesse a implementação da arquitetura, toda a infraestrutura aplicada a *GS Clients*, pode ser utilizada por *WS Clients*. Isso é feito por intermédio de um *WS Provider* e um *GS Client*.

Os objetivos principais da técnica de criptografia são garantir a integridade e a confidencialidade de mensagens. Uma mensagem é considerada íntegra quando esta é idêntica à recebida e não foi alterada no caminho. Já confidencialidade é a garantia que apenas quem tem autorização possa visualizar a mensagem.

A camada **Transmissão Segura** é responsável por garantir a transmissão segura, fim-a-fim, das mensagens entre *GS Clients* e o *GS Provider* gerenciado (WELCH et al., 2003a), garantindo a confidencialidade e a integridade destas mensagens. Pode-se optar por transmitir as mensagens com confidencialidade e/ou integridade, além de escolher se deseja segurança no nível de transporte ou de aplicação. Esta camada

garante a transferência segura das mensagens entre o mecanismo que implementa GSSA e o *GS Provider*, usando GSI. No uso de *WS Clients*, as mensagens podem ser transportadas até o mecanismo, por mensagens *Simple Object Access Protocol (SOAP)* trafegadas sobre *HyperText Transfer Protocol Secure (HTTPS)*, garantindo a segurança no nível de transporte, ou utilizando o padrão WS Security que cifra as mensagens SOAP, garantindo a segurança no nível de aplicação (JANA et al., 2006), (SHIRASUNA et al., 2004), (TANG et al., 2006).

A arquitetura permite o desativamento das camadas e módulos, para que a aplicação cliente utilize a segurança adequada aos em seus requisitos. As únicas restrições impostas para as implementações da arquitetura são: a camada "Autenticação" é automaticamente utilizada ao se configurar o uso da camada "Autorização"; e a operação de autenticação sempre deve ser executada sobre um canal que forneça confidencialidade e integridade, mesmo que a camada "Transmissão Segura" esteja desabilitada.

O uso da arquitetura proposta introduz mecanismos adicionais de segurança para GSs, utilizando como base a infraestrutura de segurança já existente.

Resumo do Capítulo 5

Este Capítulo apresentou a arquitetura de segurança chamada GSSA que estende a arquitetura OGSA.

As camadas de GSSA fornecem as funcionalidades Autenticação, Autorização, Transmissão Segura, *Logging* e Tradução de chamadas de WS para GS.

No próximo Capítulo será apresentada a implementação dessa arquitetura de segurança proposta.

CAPÍTULO

6

Implementação da GSSA

"Se você pensa que segurança custa caro, experimente um acidente."

STELIOS HAJI-LOANNOU

Para a avaliação e a validação da GSSA, um *Proxy* de Segurança chamado *Grid Service Security Proxy (GSSP)* foi implementado em Java, utilizando o WSRF. Atuando na interface das chamadas, GSSP é mecanismo intermediário ao qual a aplicação cliente se conecta e que executa as operações do *GS Provider* gerenciado. O *Proxy* é usado para que não seja necessário modificar o código fonte do *GS Provider* no controle de acesso às operações, já que este pode não estar disponível, o que simplifica o uso da arquitetura de segurança desenvolvida. GSSP também exige apenas pequenas modificações na aplicação cliente, como o acréscimo de uma biblioteca (*GSSALib*) para a configuração de parâmetros de forma transparente.

Todas as camadas da arquitetura podem ser desabilitadas, utilizando-se uma operação disponibilizada pelo GSSP porém, com algumas ressalvas. Normalmente, as camadas são desabilitadas por questões de desempenho, mas a camada "Tradução",

por exemplo, pode ser desabilitada para não permitir o uso de WSs no acesso ao *GS Provider* gerenciado. A operação "*void setEnabled(String layer, Boolean status)*" é usada para este fim, na qual informa-se o nome da camada e o status (habilitada ou desabilitada) da mesma.

A Figura 6.1 mostra a comunicação entre o *GS Client* e a Aplicação, por intermédio de um GSSP. Esta comunicação será melhor detalhada no decorrer do Capítulo.

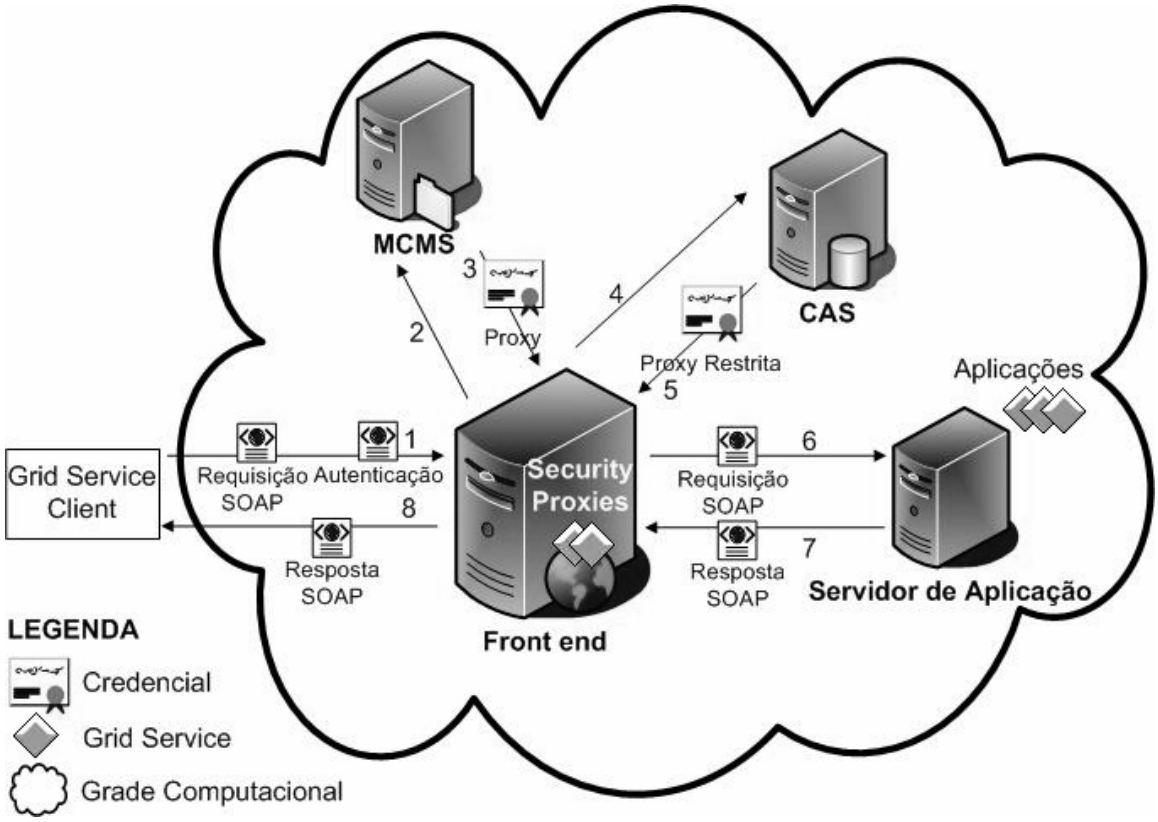

Figura 6.1: Infraestrutura de segurança utilizando o GSSP.

GSSP não permite desabilitar a camada de autenticação se a camada de autorização estiver ativa; caso esta tentativa ocorra, uma Exceção é lançada. Desabilitar a camada "Transferência Segura" não influi na autenticação, pois o arquivo de configuração "security-config.xml" de GSSP, prevê que esta operação deve ser feita sempre sobre um canal seguro com integridade e privacidade.

A Figura 6.2 mostra detalhadamente os componentes do *Authentication and Authorization Security Descriptor Framework (AASDF)* envolvidos do acesso ao *GS Provider*.

Neste ambiente, o *GS Client* (no computador 1) objetiva acessar as operações do *GS Provider* (no computador 5).

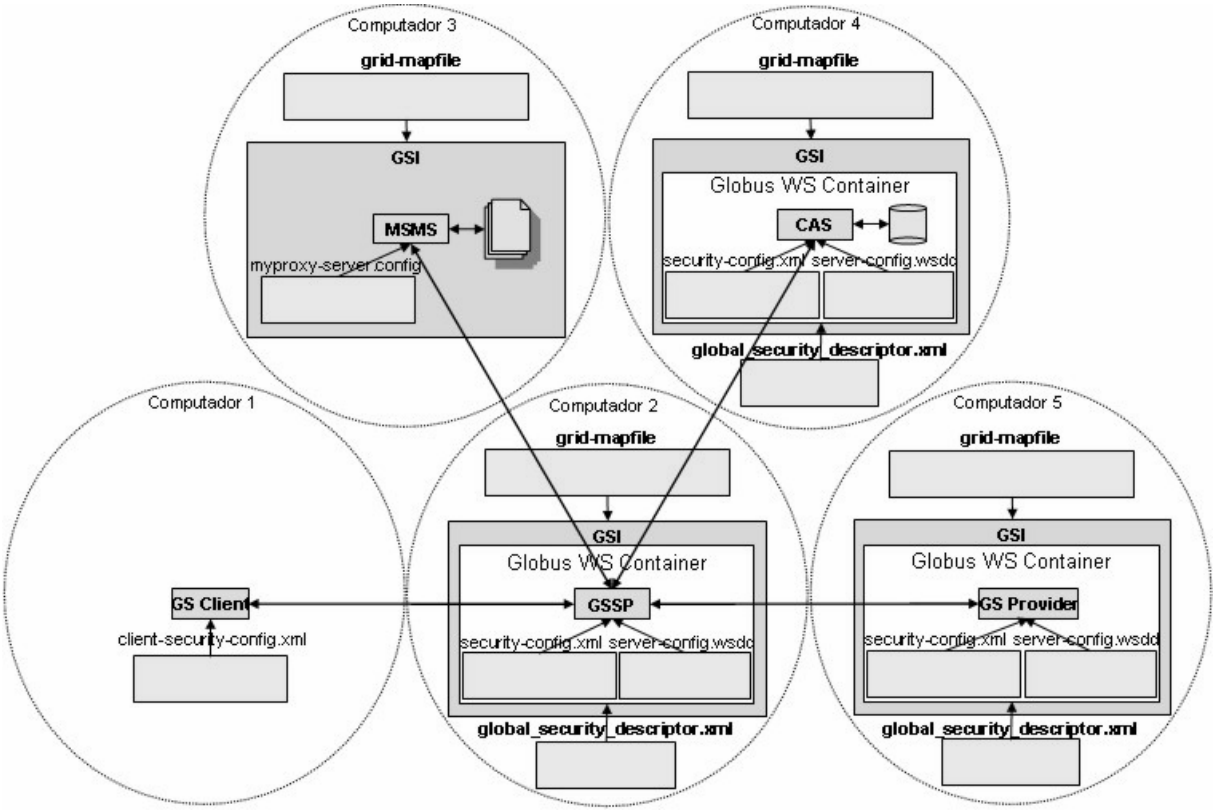

Figura 6.2: Mecanismos de Segurança do Ambiente de Grade.

O arquivo "*client-security-config.xml*" do *GS Client* é passado para a classe *Stub*, utilizando o método "*_setProperty(String name, Object value)*" com os parâmetros "`Constants.CLIENT_DESCRIPTOR_FILE`" e a URL do arquivo de configuração.

O descritor de segurança do CAS é configurado para funcionar com qualquer configuração de autenticação, e com autorização do tipo 'none', indicando que esta não é requerida. *Globus WS Container (GWSC)* é configurado para aceitar qualquer tipo de autenticação, e a autorização é do tipo 'gridMap', informando que o usuário poderá chamar o GS se estiver autorizado no *grid-mapfile*. O parâmetro '*run-as*' do GSSP é configurado como '*caller-identity*', de forma que GSSP assume a identidade usada pelo *GS Client*, não introduzindo, assim, qualquer problema de autorização, com a utilização deste *Proxy*. GSSP também é configurado para aceitar qualquer tipo de

autenticação, mas o método *Init* do GSSP que é usado para, entre outras coisas, realizar a autenticação do usuário, é configurado para ser feito sempre sob autenticação utilizando privacidade e integridade.

Quando o GS Client acessa GSSP, este normalmente utiliza autorização do tipo host.

6.0.1 Camada Autenticação

Antes da realização da autenticação, *GSSALib* verifica se existe um arquivo chamado "*$USER.key*", contendo as informações necessárias à construção do objeto *EndPoint-Reference (EPR)* representados em XML, como a URL do GS e a chave de acesso ao recurso. Caso o arquivo não exista, isto significa que o usuário não está autenticado, desta forma, *GSSALib* realiza o procedimento de autenticação e armazena o EPR retornado deste procedimento no "*$USER.key*", para uso posterior.

A operação de autenticação é executada uma única vez pela biblioteca *GSSALib*, incorporada ao *GS Client*. Para isso, contudo, é necessário que esta biblioteca tenha sido carregada com a credencial do usuário, ou usuário e senha, para a realização do procedimento de autenticacão. *Init* realiza a autenticação do usuário na Grade, obtém do *Community Authorization Service (CAS)* os direitos deste usuário, cria um *WS Resource* com estes direitos, e gera uma chave que referencia o recurso para ser possível acessá-lo posteriormente. Como é necessário utilizar uma instância de GSSP na aplicação cliente para executar as operações do *proxy*, não há como executá-las caso a autenticação não tenha ocorrido com sucesso, pois esta instância é realizada durante a execução da operação *init*.

Caso a aplicação cliente do usuário tenha a credencial *proxy*, a autenticação pode ser feita por meio da operação *EndPointReference init(X509Certificate cert)*.

Caso a aplicação cliente não tenha suas credenciais na autenticação, *GS Client* chama a operação *EndPointReference init(String, String)* do GSSP com parâmetros *username*, referente ao usuário da grade computacional, e *passphrase*, que é a senha que protege o certificado *proxy* do usuário armazenado no repositório (passo 1 da

Figura 6.1) do MCMS. Então, *init* contacta MCMS (passo 2) e obtém uma credencial *proxy* assinada por MCMS e gerada a partir da credencial previamente armazenada no repositório (passo 3), com uso do método *myProxyLogon*, acessado através do *framework Java WS Core* (GAWOR, 2004), (GLOBUS TEAM, 2009b). O MCMS é utilizado para que não seja necessário copiar manualmente as credenciais dos usuários para o computador onde GSSP está hospedado, simplificando as tarefas de autenticação do GSSP na Grade.

Após a obtenção da credencial *proxy*, GSSP verifica a assinatura e validade da credencial do usuário e cria um arquivo contendo esta credencial em `/tmp/x509up_u$ID_USER`, onde *$ID_USER* representa o identificador do usuário no Sistema Operacional. O Globus Toolkit consulta inicialmente a credencial do usuário neste local quando este tenta acessar a um recurso ou serviço, então, o usuário está autenticado na grade.

Após a autenticação, a operação *init* contacta o serviço CAS e requisita uma Credencial *Proxy* Restrita (CPR), que é uma decisão de autorização (passo 4), utilizando o método *casProxyInit*. CAS verifica se o usuário está autenticado na Grade e retorna a CPR contendo os direitos do usuário (passo 5) expressos em assertivas SAML, gerados com base nas políticas previamente cadastradas. GSSP verifica a assinatura da Credencial *Proxy* Restrita e sua validade e realiza o *parser* da credencial utilizando a API OpenSAML[1], que permite obter os parâmetros SAML da credencial, em alto nível.

GSSP utiliza o gerenciador de recursos de GSs *(Resource Home - RH)*, no desenvolvimento do mecanismo ***Cache* de Direitos** e foi implementado seguindo o padrão de projeto denominado "Factory" (GAMMA et al., 2005). A operação *init* faz com que o RH mapeie a CPR em um *WS Resource*, contendo a identidade do usuário e um vetor com as operações permitidas ao usuário e o tipo de ação (e.g., ler, escrever, executar) que o usuário pode executar sobre a operação (no Apêndice A há um exemplo destas informações). Uma sequência aleatória de caracteres *hash*, que funciona como identificador ou chave, é gerada para referenciar este recurso. RH possui uma estrutura de

[1] `https://spaces.internet2.edu/display/OpenSAML/Home`.

dados interna na qual são armazenada o *hash* e o respectivo recurso. Ao término de *init*, um objeto EPR é retornado a *GSSALib* (passo 8 da Figura 6.1). Estas atividades podem ser melhor visualizadas observando-se a Figura 6.3.

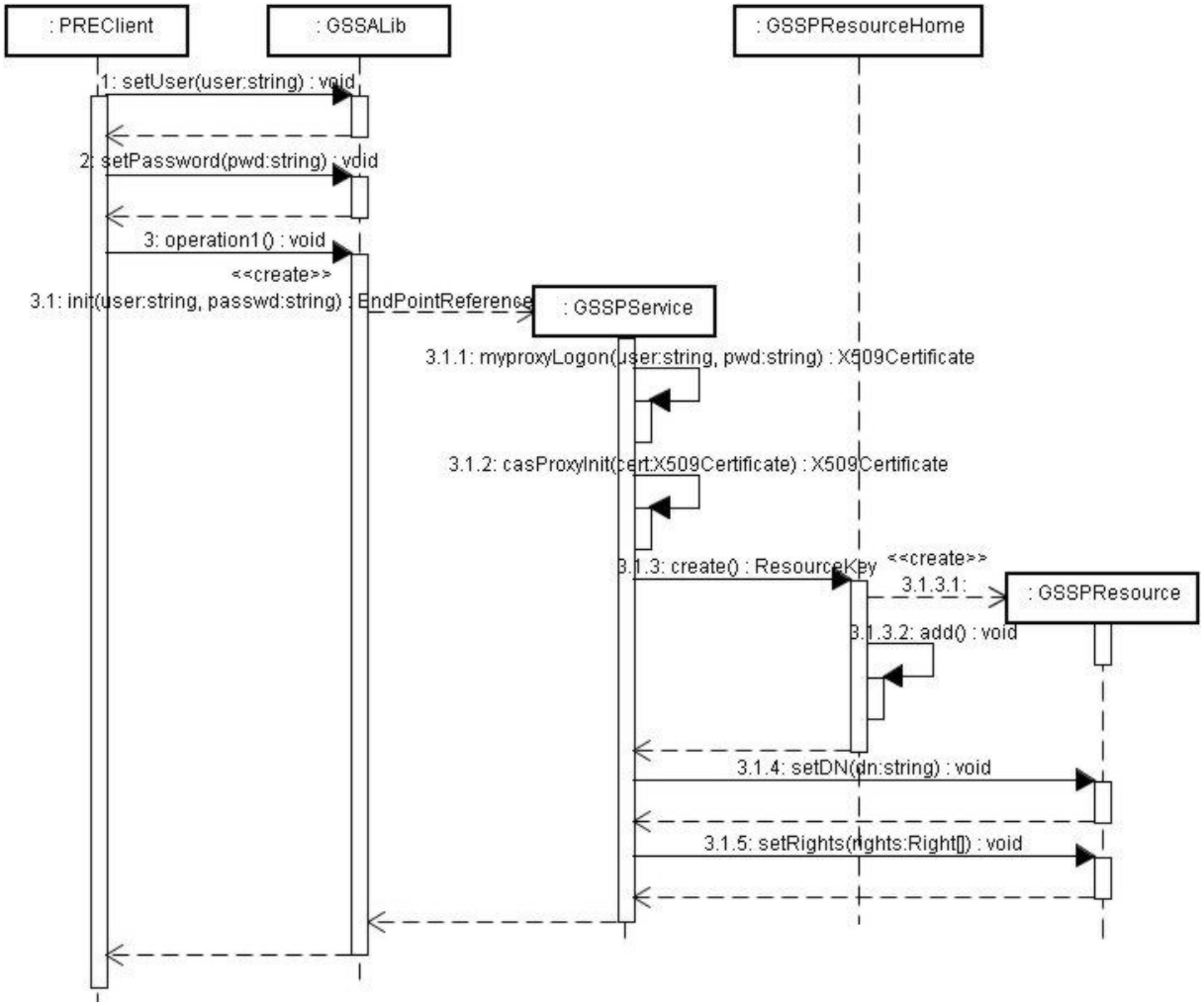

Figura 6.3: Fluxo de execução da funcionalidade autenticação.

Por uso do Padrão *WS ResourceLifetime*, é dado ao recurso um tempo de vida igual ao dos direitos da CPR, fazendo com que o RH destrua este recurso automaticamente no tempo especificado. O GSSP fornece uma operação que explicitamente destrói o recurso, baseado no EPR informado, destruindo assim os direitos do usuário, como apresentado na Figura 6.4. Após a destruição do recurso, o arquivo "*resource.key*" que contém a chave do usuário é destruído.

Isso é importante no caso do *GS Client* necessitar remover os direitos do usuário

(e.g., realização de *logoff*), antes do término do tempo de vida da credencial. Caso tente-se acessar o recurso após este ser destruído, a Exceção *NoSuchResourceException* é lançada por RH.

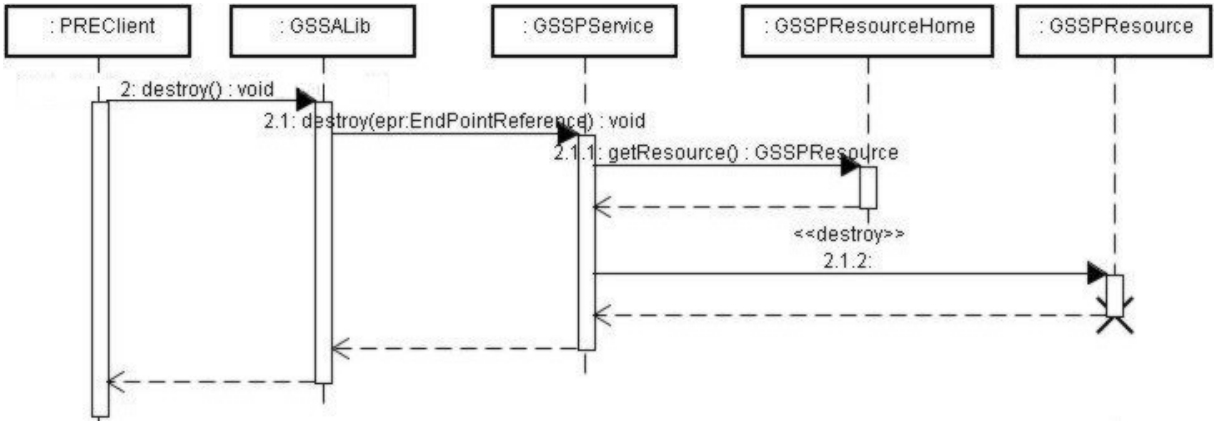

Figura 6.4: Fluxo de execução da funcionalidade *destroy*.

A biblioteca **GSSALib** é necessária na comunicação do *GS Client* com o GSSP, para tornar o uso do *proxy* transparente para o *GS Client*, não precisando este se preocupar com as questões de autenticação e autorização envolvidas no processo. Assim, não é necessário mudar as chamadas às operações no código fonte desta aplicação. Esta biblioteca é quem executa a operação *init* e gerencia o objeto EPR que identifica o recurso. Nas chamadas às operações, esta biblioteca conecta-se ao GSSP ao invés de conectar-se ao *GS Provider*. Não é necessário alterar o *GS Client* para suportar o tratamento da exceção *NotPermittedException*, que é gerada quando o usuário não possui direito de chamar uma operação, pois esta estende a *RemoteException*, que já é prevista e tratada quando se implementa GSs.

6.0.2 Camada Autorização

Para a camada "Autorização", GSSP realiza o controle de acesso no momento da chamada de uma operação do *GS Provider*, verificando se a 4-upla *(usuário, tipo_operação, ação, operação)* está presente no *WS Resource* do usuário em GSSP. Baseando-se no modelo de autorização proposto por XACML, na abordagem desenvolvida (ver Figura 6.5), GSSP funciona como um Ponto de Aplicação de Políticas *(Policy Enforce-*

ment Point - PEP), requisitando uma decisão de autorização de CAS e aplicando-a. Desta forma, CAS funciona como Ponto de Administração de Políticas *(Policy Administration Point - PAP)*, administrando os usuários, os recursos e as ações, e de Decisão de Políticas *(Policy Decision Point - PDP)*, gerando decisões de autorização com base nas políticas do sistema.

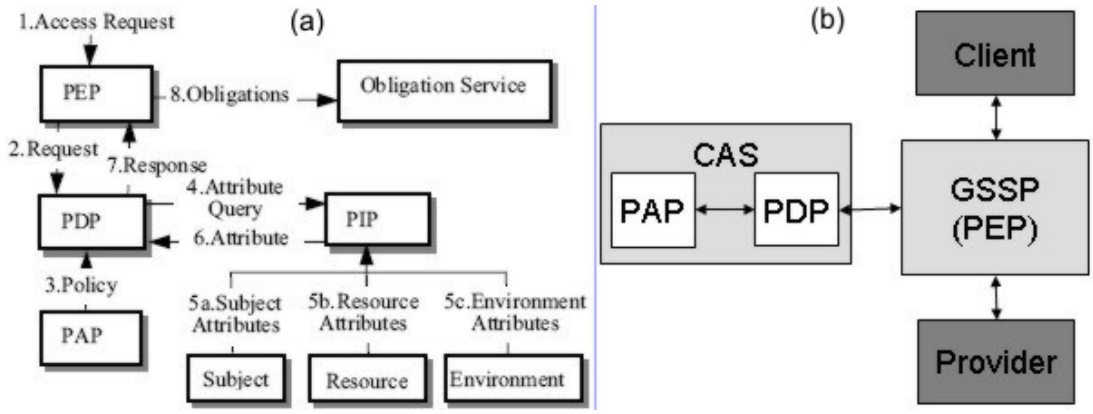

Figura 6.5: Especificação XACML. (a) Elementos da especificação (LANG et al., 2006). (b) Ambiente GSSP mapeado na especificação.

Para implementar o módulo **História**, o parâmetro *maxOccurs* dos recursos é configurado no WSDL do GSSP com valor '1'. Com este valor, os valores anteriores contidos nos recursos não são preservados. Se o valor deste parâmetro fosse, por exemplo, '*unbbounded*', os valores do recurso seriam preservados e desta forma não seria possível identificar qual o valor que um recurso teria em um momento específico, sem modificar o arquivo original gerado pelo RH. O arquivo gerado é salvo com o nome formado pelo nome de usuário, concatenado com o momento em que o este adquiriu os direitos e concatenado com o momento da destruição do recurso.

Na autorização, quando *GS Client* faz chamada a uma operação, *GSSALib* usa o EPR para informar qual é o recurso a ser manipulado e faz a chamada à operação correspondente no GSSP (passo 1 da Figura 6.1). GSSP verifica se o usuário tem a permissão de executar a operação pretendida; caso tenha, a operação do *GS Provider* é chamada (passo 6), e o retorno é enviado de volta ao GSSP (passo 7), caso não tenha, uma exceção *NotPermittedException* é lançada. Por fim, a mensagem com a exceção ou com o retorno da execução da operação é enviada ao *GS Client* pelo GSSP

(passo 8).

A Figura 6.6 mostra em mais detalhes como se dá a autorização nas chamadas às operações.

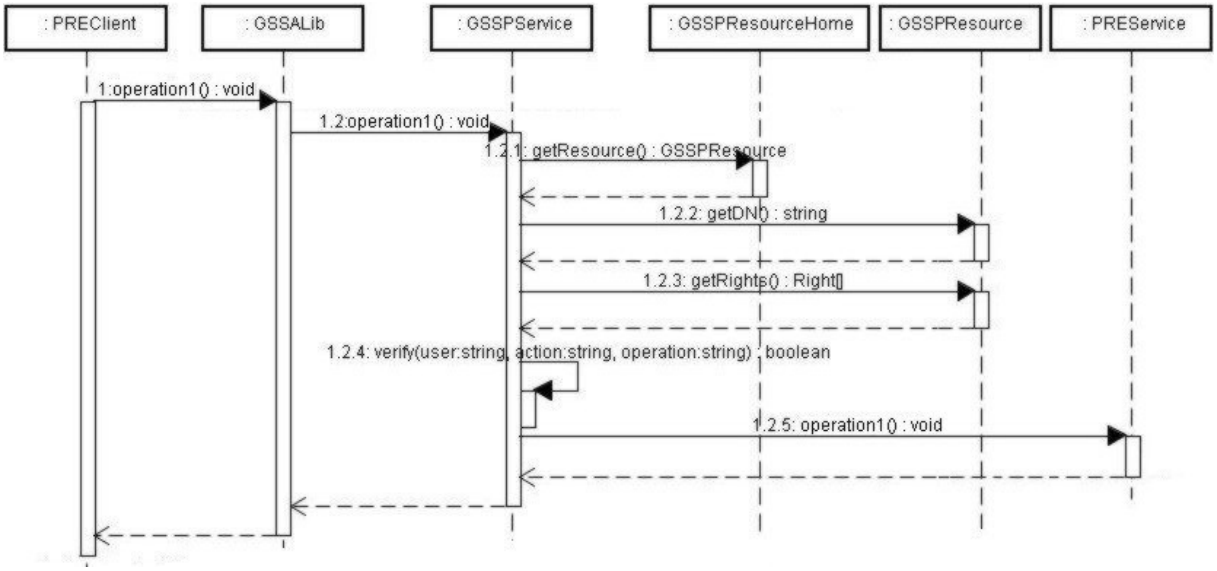

Figura 6.6: Fluxo de execução da autorização sobre as operações.

A Figura 6.7 mostra como se dá a autorização no acesso aos recursos de GS pelo *GS Provider* gerenciado.

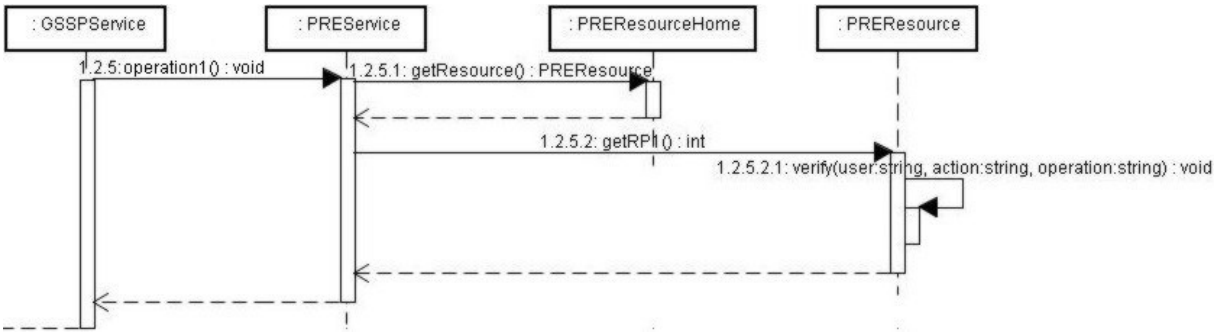

Figura 6.7: Fluxo de execução da autorização sobre os recursos de GS.

O controle de autorização do GSSP foi desenvolvido utilizando Programação Orientada a Aspectos *(Aspect Oriented Programming - AOP)* em Java chamado de *AspectJ* (LADDAD, 2003). AOP auxilia a Programação Orientada a Objetos *(Object Oriented Programming - OOP)*, entrecortando (*crosscuting*), em tempo de compilação, os métodos das classes, visando principalmente o reuso. Os principais elementos da AOP são

os Pontos de Junção (*JoinPoints*), que são quaisquer pontos identificáveis na execução de um programa (e.g., chamada de método, instância de objeto), Pontos de Corte (*PointCuts*), que permitem selecionar um ou mais *JoinPoints* por uso de expressões regulares, e Aviso (*Advice*), que é o código a ser executado em *JoinPoints* selecionados por um *PointCut*, permitindo especificar se o código será introduzido antes (*before*), depois (*after*) ou durante (*around*) a execução do método.

No Apêndice C.2 há um exemplo dos valores dos recursos que são armazenados em arquivo.

Framelet de Autorização

O *framelet* (*framework* de pequeno porte) de autorização permite 2 tipos de especialização. A primeira, utilizando Aspecto, permite configurar quais as classes e métodos serão entrecortados (*crosscuts*). A outra, na classe de mapeamento de direitos em *WS Resources*, permite a utilização de decisões de autorização em diversos formatos. Para isso, o *framelet* fornece os Aspectos abstratos chamados *AuthorizationAspect*, *ServiceAuthorizationAspect*, *ResourceAuthorizationAspect*, e a classe abstrata chamada *MappingResource*, como pode ser observado na Figura 6.8.

O Aspecto *AuthorizationAspect* do *framelet* de autorização especifica um *pointcut* abstrato, que indica que o Aspecto que o implementar deve indicar quais as classes e métodos que devem ser entrecortados, e especifica também um *advice* abstrato do tipo *before*, que indica que o Aspecto que o especializar deve proceder a verificação de direitos e realizar o controle de acesso, antes da chamada do método entrecortado. Os Aspectos *ServiceAuthorizationAspect* e *ResourceAuthorizationAspect* estendem *AuthorizationAspect* para o controle de acesso a operações e a recursos, respectivamente.

A implementação do *advice before* verifica se há a 4-upla (usuário, tipo_autorização, ação, objeto) no *WS Resource* selecionado pelo *Cache* de Direitos, com uso da chave informada na autenticação. Se existir, o usuário pode executar a operação e o código inserido pelo Aspecto, não realiza nenhuma restrição à chamada realizada por GSSP ao *GS Provider*. Se não existir, o código inserido pelo Aspecto lança uma exceção

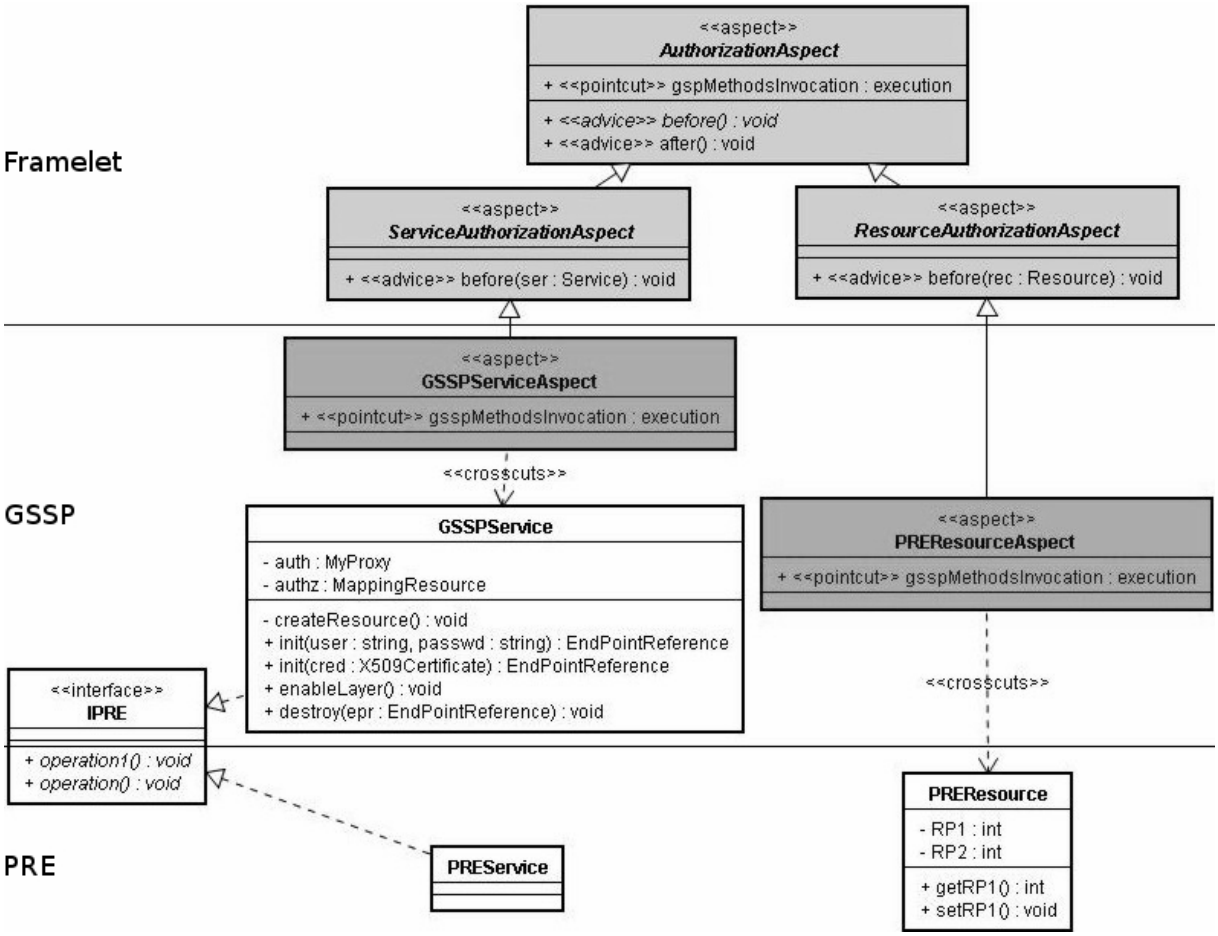

Figura 6.8: Componente do *Framelet* para o controle de acesso das operações e recursos.

NotPermittedException, não chamando a operação do *GS Provider*.

Em *ServiceAuthorizationAspect*, o usuário é representado pela identidade *(Distinguished Name - DN)* da credencial, a ação é "*read*", e o objeto é composto pela URL do *GS Provider* concatenado com a operação que se deseja acessar (e.g., `http://localhost:8080/wsrf/service/PRE|void,inserirRecado,String`). Já em *ResourceAuthorizationAspect*, a ação pode ser "*get*" ou "*set*", e o objeto é composto pela URL do *GS Provider* concatenado com o nome do recurso (nome do método que acessa o recurso, excluindo-se o prefixo "*get*" ou "*set*"). O objeto é obtido por meio de um objeto *thisJoinPoint*, que permite obter o nome do método que o Aspecto está entrecortando, assim como os tipos, nomes dos parâmetros e tipo de retorno.

ServiceAuthorizationAspect foi especializado para o Aspecto *GSSPServiceAuthoriza-*

tionAspect para realizar o controle de acesso às operações da classe *GSSPService*. Este Aspecto funciona da seguinte forma: GSSP disponibiliza na classe *GSSPService*, todas as operações que implementam a lógica de negócio do *GS Provider* gerenciado (*PREService*), e disponibiliza também operações para configurar o uso das camadas de segurança e para autenticação. Este aspecto insere na compilação, o código para controle de acesso antes da chamada de cada operação em GSSP que faz acesso ao *GS Provider*, excluindo assim as operações próprias do GSSP.

ResourceAuthorizationAspect foi especializado para o Aspecto *PREResourceAuthorizationAspect* para realizar o controle de acesso aos recursos da classe *PREResource*. Este Aspecto funciona da seguinte forma: PRE disponibiliza na classe *PREResource* todos os métodos necessários à manipulação dos Recursos. Este aspecto insere na compilação o código para controle de acesso antes da chamada de cada método em PREResource que faz acesso ao recurso, seja para buscá-lo ou modificá-lo.

No Apêndice B há o código fonte do *framelet*, e das instâncias criadas para a realização do controle de acesso em GSSP.

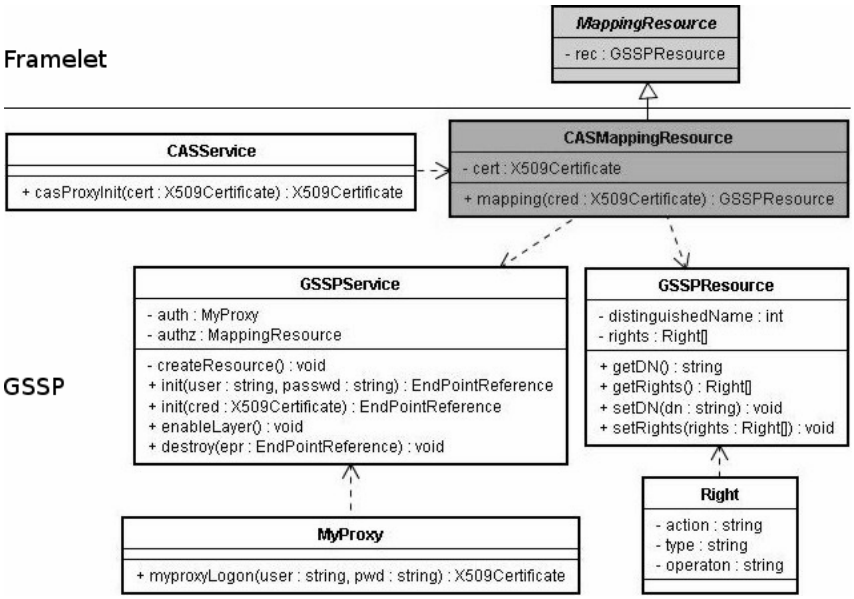

Figura 6.9: Componente do *Framelet* para o de mapeamento de direitos.

Em GSSP este *framelet* foi também especializado para a Classe *CASMappingResource*, a fim de permitir o mapeamento de direitos decisões, expressos em assertivas

SAML, em *WS Resources*, como pode ser visto na Figura 6.9. Para isso, *CASMappingResource* contacta CAS, obtém a credencial e representa as informações da credencial nos atributos do Recurso de GSSP (*GSSPResource*).

6.0.3 Camada Sistema de Autorização

Na implementação de GSSA utilizou-se CAS como *Authorization System*. No entanto, qualquer sistema desenvolvido baseado na OGSA poderia ser utilizado.

CAS foi modificado para suportar **notificações** de eventos e gerar uma notificação quando ocorre uma mudança nas políticas dos usuários. A este novo sistema deu-se o nome de *eXtended CAS (XCAS)*.

eXtended CAS

CAS possui 2 recursos chamados "*serverDN*", que representa o DN do Servidor e "*VoDescription*", que representa a descrição da VO. Foi acrescentado mais um recurso que é um vetor de booleanos, onde cada posição refere-se a um usuário cadastrado em CAS. Este novo recurso é controlado pelo gerenciador de notificações.

Quando as informações de um usuário são apagadas do CAS, este usuário perde todos os seus direitos de acesso; então, GSSP destrói explicitamente o recurso deste usuário. Quando um usuário é removido/incluído em um grupo, é alterado o recurso em CAS referente a este usuário, porque o usuário perde/adquire os direitos do grupo envolvido. Quando um grupo é removido, são alterados os recursos dos usuários deste grupo, porque os usuários perdem os direitos que pertenciam a este grupo. Quando uma política é removida/inserida, são alterados os recursos dos usuários, porque ele irá adquirir/perder os direitos do grupo relacionado.

Quando o recurso é modificado, gera-se implicitamente uma notificação para os assinantes. GSSP realiza uma assinatura (*subscribe*) no XCAS e quando este *proxy* recebe uma notificação procede a busca de uma nova CPR com os novos direitos do usuário, salva os valores correntes do recurso em arquivo (módulo *História*) e atualiza os dados do recurso do usuário.

6.0.4 Módulo *Logging*

Na implementação do módulo *Logging*, é realizado um armazenamento em arquivos de *log* sobre os acessos do usuário e as tentativas ou realizações de autenticações.

Logging em *Java WS Core (JWSC)* é baseado na *API Commons Logging* da Apache, conhecida como log4j. O *logging* no GWSC pode ser configurado em `$GLOBUS_LOCATION/container-log4j.properties`, e o *logging* nos *GS Providers* pode ser configurado em `$GLOBUS_LOCATION/log4j.properties`.

Não são registradas informações sobre depuração (e.g., a cada chamada de método ou atribuição de valores) e avisos (e.g., aviso de perda de precisão em *cast* de pontos flutuantes para inteiros) interessantes em análise de erros de programação.

O Aspecto *SecurityAspect* do *framelet* de autorização de GSSP (ver Apêndice B) insere o código para controle de *log* em tempo de compilação, após (por meio do *advice* chamado *after*) a chamada de cada operação de GSSP e também após a chamada ao método *init*. Um arquivo de log, cujo nome refere-se ao nome do usuário, é registrado contendo um valor *flag* (0, 1, 2, 3 ou 4) referente à operação realizada (*login*, *logout*, acesso bem sucedido, acesso mal sucedido e exceção não esperada), o DN da credencial do usuário, o horário atual, a operação que está sendo utilizada, e uma mensagem ou exceção capturada.

Um exemplo de arquivo de *logging* gerado por GSSP pode ser visto no Apêndice C.1.

6.0.5 Camada Transmissão Segura

A implementação da camada "Transmissão Segura" permite a GSSP transportar de maneira segura as mensagens entre *GS Client* e *GS Provider*. A biblioteca *GSSALib* configura a conexão com o GSSP para ser segura no nível de transporte e GSSP configura-se para transferir os dados de forma segura até o *GS Provider*. Assim, consegue-se garantir segurança no tráfego das mensagens fim-a-fim.

Para isso, deve-se configurar o método *_setProperty(String name, Object value)* nas classes *Stubs* do *GS Client*, e do módulo cliente do GSSP.

Estes parâmetros podem receber as seguintes informações (ver Seção 3.1.4):

- Quando o primeiro parâmetro deste método for *Constants.GSI_SEC_CONV* indica que será utilizado o método de autenticação *GSISecureConversation*. Neste caso, como segundo parâmetro pode-se ter *Constants.ENCRYPTION*, indicando que a mensagem será cifrada, e/ou *Constants.SIGNATURE*, indicando que a mensagem será assinada.

- Se o primeiro parâmetro for *Constants.GSI_XML_SIGNATURE* será utilizado o método de autenticação *GSISecureMessage*. Então, o segundo parâmetro pode ser Boolean.TRUE ou Boolean.FALSE, indicando se será utilizada autenticação de chave pública.

- Para se configurar a autorização do lado cliente utiliza-se o primeiro parâmetro como *Constants.AUTHORIZATION*, e o segundo parâmetro sendo *NoAuthorization. getInstance()*, indica autorização será do tipo *None*, *SelfAuthorization.getInstance()* indica que será do tipo *Self*, e HostAuthorization.getInstance() indica o tipo *Host*.

A operação para habilitar/desabilitar camadas foi sobrecarregada para "*void setEnabled(String layer, String security, Boolean status)*", a fim de permitir que a camada "Transmissão Segura" indique se será feito o controle de integridade e/ou de privacidade, por meio do parâmetro *security*.

A segurança da comunicação entre os *WS Clients* e GSSP pode ser provida via HTTPS quando dispositivos móveis são empregados. Mesmo que dispositivos móveis não suportem o padrão *WS Security*, tecnologias baseadas neste padrão podem ser utilizadas por WS executados em *Desktops*, usando uma *Java Virtual Machine (JVM)* mais robusta.

6.0.6 Camada Tradução

A implementação da camada "Tradução" foi realizada com o uso do módulo *GSSATranslating*, como pode ser observado na Figura 6.10, que é formado por um *WS Provider* e um *GS Client*. As requisições do *WS Client* são enviadas ao *WS Provider*, que disponibiliza todas as operações disponíveis de GSSP, inclusive as de controle de segurança.

Na implementação de cada operação do *WS Provider* está a implementação de um *GS Client* que conecta-se ao GSSP. O uso deste módulo foi necessário porque não é possível disponibilizar WSs no GWSC, então *WS Provider* precisou ser disponibilizado em outro *container*. O *WS Container* utilizado foi o *Apache Axis*, executado sobre o *Apache Tomcat*, que é um servidor aplicações *web* desenvolvidas em Java.

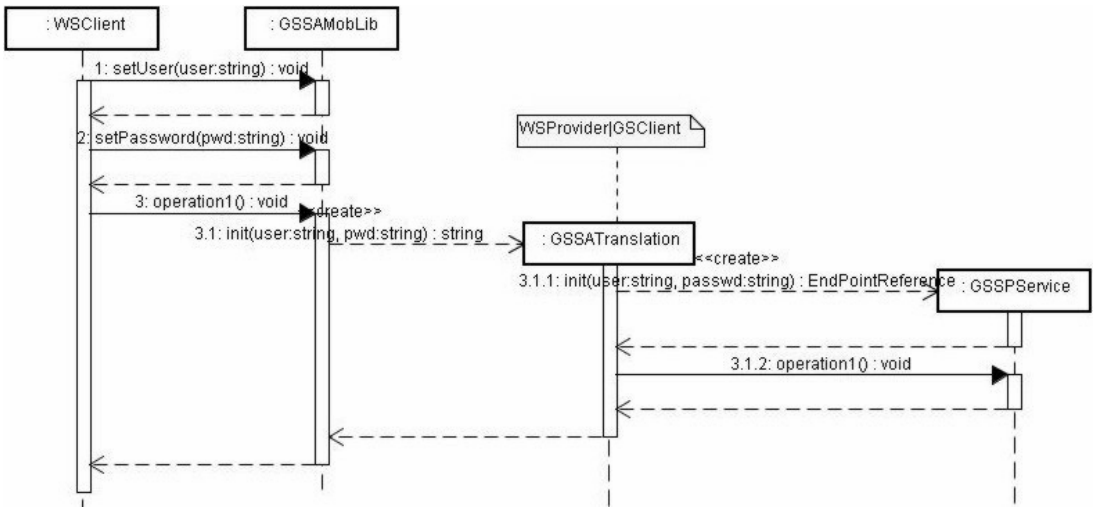

Figura 6.10: Fluxo de execução da Tradução das chamadas WS para acesso à *Grid Services*.

O procedimento de autenticação de WS Clients acontece de forma similar à autenticação de *GS Clients*, utilizando-se o arquivo "*$USER.key*", contendo as informações do objeto EPR. Este arquivo é gerenciado por *GSSAMobLib*, que o interpreta apenas como texto, e por *GSSATranslating*, que tranforma o seu conteudo em um objeto EPR, que será utilizado pelo *GS Clients* a fim de instanciar o GSSP.

O *GSSAMobLib* funciona de forma similar a *GSSALib*, gerenciando a chave que referencia os direitos do usuário. Como o *WS Client* pode estar sendo executado a partir de um dispositivo e o *Java Micro Edition (JME)* utilizado nestes, não tem suporte para compreender objetos EPRs, *GSSALib* contém apenas a chave do recurso como uma *String*. *GSSALib* informa a chave ao *GSSATranslation*, que gerencia o objeto EPR para acesso aos recursos com os direitos do usuário.

Resumo do Capítulo 6

Este Capítulo apresentou a arquitetura GSSA e suas camadas que fornecem as funcionalidades Autenticação, Autorização, Transmissão Segura, *Logging* e Tradução.

Também apresentou em detalhes uma implementação de GSSA chamada GSSP. Nesta implementação, MCMS foi utilizado no auxílio à autenticação, AASDF no suporte a autorização, CAS como sistema de autorização, um *framelet* em *AspectJ* para o controle de acesso e *logging*, e GSI foi utilizado para a transferência segura dos dados.

No próximo capítulo serão apresentados os trabalhos relacionados, juntamente com um comparativo com a arquitetura proposta.

CAPÍTULO

7

Trabalhos Relacionados

"Mesmo aquilo que é considerado bom pode sempre ser melhorado."

JAPONÊS DESCONHECIDO

Os mecanismos existententes para Grades Computacionais baseadas no middleware Globus Toolkit ainda apresentam algumas dificuldades ou deficiências como as que serão apresentadas a seguir.

A arquitetura GSSA proposta fornece um controle de acesso mais refinado sobre os GSs, do que a ***Security Architecture for Open Grid Services (SAOGS)***, permitindo, além de especificar quem pode acessar um GS, indicar também quais das operações podem ser chamadas. Este trabalho apresenta também uma implementação da arquitetura GSSA proposta utilizando o *framework* WSRF, enquanto SAOGS apenas indica alguns possíveis padrões de WS que podem ser utilizados para atingir os requisitos da arquitetura, mas não apresenta uma implementação para validá-la e analisar o seu custo de implementação e implantação.

CaGrid Security Architecture (caGSA) realiza um controle de autorização sobre

operações de GS, ficando a cargo do *GS Provider* realizar a autorização, mas este deve ser modificado para tanto. Na arquitetura GSSA proposta, o *GS Provider* não tem ciência da infraestrutura de segurança. Outra diferença é que caGSA utiliza o *PrivilegE and Role Management Infrastructure Standard (PERMIS)* como sistema de autorização. Já a implementação da arquitetura GSSA proposta utiliza o CAS, que faz parte do núcleo do *Globus Toolkit*, não necessitando instalar sistemas adicionais.

Authentication and Authorization Architecture for Grid Computing (3AGC) utiliza AOP para o controle de autenticação e autorização na Grade, enquanto a arquitetura GSSA proposta utiliza AOP para o controle de *logging* e autorização e, além disso, fornece outras funcionalidades de segurança como, transferência segura de dados e tradução de chamadas oriundas de WSs para acesso a GSs. 3AGC intercepta as chamadas de autenticação e autorização do GWSC, impondo o uso da arquitetura. Esta abordagem, contudo, é dependente da implementação do GWSC que, caso seja modificado ou substituído por outro *container*, requer a readequação da solução. A arquitetura GSSA proposta também se apresenta como uma opção de segurança para desenvolvedores de *GS Clients* mas não é uma imposição.

GridShib visa sobretudo a privacidade da identidade dos usuários, ao invés de preocupar-se em atender os requisitos básicos de segurança no acesso aos GSs. A arquitetura de segurança proposta fornece estes requisitos como autenticação, autorização, transferência segura de dados e *logging*, além de permitir a interoperabilidade entre GSs e WSs.

As políticas em ***PrivilegE and Role Management Infrastructure Standards (PERMIS)*** são disponibilizadas quando o sistema inicia e, caso alguma mudança ocorra nestas políticas, elas só serão refletidas ao usuário quando o sistema for reiniciado, o que é inadequado para um ambiente no qual as políticas podem mudar constantemente. Uma alternativa a este problema seria realizar consultas ao PERMIS periodicamente. O uso de notificações prevista pela arquitetura GSSA proposta permite a sincronia das políticas de segurança de sistemas de autorização baseados em GS, com um baixo custo de comunicação resolvendo a limitação comumemente

encontradas nestes sistemas. A arquitetura GSSA proposta não precisa utilizar ACs porque o tempo de vida da identidade do usuário é definido pela credencial do usuário, e o tempo de vida dos seus direitos é definidos pelos *WS Resources*.

Akenti implementa o mecanismo de *cache* dos certificados, mas é necessário realizar uma análise das informações destes certificados todas as vezes que o controle de acesso for requisitado. Na arquitetura GSSA proposta a análise das informações do certificado é feito uma vez na inicialização do sistema e somente será necessário analisá-lo novamente caso as políticas dos usuários sejam modificadas. A arquitetura GSSA proposta usa certificados *proxy* para fornecer autenticação e autorização de maneira simples visando a manutenabilidade, ao invés de ter que manipular diversos tipos de certificados.

Multipolicy Authorization Framework (MAF) lida somente com questões de autorização, ao contrário da arquitetura GSSA proposta que lida com outras questões de segurança em grades computacionais, além de ser utilizada por desenvolvedores de aplicações clientes, de maneira que eles não precisem se preocupar com a infraestrutura de segurança nas aplicações.

A arquitetura GSSA proposta também realiza o *cache* dos papéis dos usuários, assim como ***Virtual Organization Membership Service (VOMS)***, além disso, esta arquitetura faz *cache* dos acessos e da autenticação, permitindo alem de saber qual papel o usuário tinha em um momento específico, saber o que ele fez no sistema neste mesmo momento.

Resumo do Capítulo 7

Este Capítulo apresentou as deficiências dos mecanismos de segurança adicionais, incluindo SAOGS, caGSA e 3AGC, além de mecanismos de segurança que estendem as funcionalidades de GSI como *GridShib*, PERMIS, *Akenti*, MAF e VOMS, para Grades Computacionais baseadas no Globus Toolkit.

A arquitetura SAOGS não fornece autorização sobre operaçoes de GSs, caGSA fornece autorização sobre as operações mas necessita que os estes sejam alterados

e 3AGC utiliza aspecto para autorização de GS controlando os mecanismos do *container*, sendo dependente da implementação desse *container*.

O mecanismo *GridShib* não se preocupa com os principais requisitos de segurança, PERMIS não consegue manter a sincronia das políticas do usuário com um baixo custo de comunicação, *Akenti* necessita analisar as informações dos certificados a cada controle de acesso, MAF é voltado a desenvolvedores de sistemas de autorização, e VOMS não realiza *cache* das autenticações e acessos aos recursos.

No próximo Capítulo será abordado o estudo de caso realizado em uma aplicação da telemedicina.

CAPÍTULO

8

Estudo de Caso

"A teoria sempre acaba, mais cedo ou mais tarde, assassinada pela experiência."

ALBERT EINSTEIN

O processo de ensino-aprendizagem do Curso de bacharelado em Medicina da Universidade Federal de São Carlos (UFSCar) (Conselho de Ensino e Pesquisa da Universidade Federal de São Carlos, 2005) utiliza princípios do aprendizado baseado em problemas *(Problem Based Learning - PBL)* (BLIGH, 1995), (RHEM, 1998). Desta forma, os estudantes, ao longo do curso, precisam atuar em cenários diversificados, tais como domicílios, creches, escolas, instituições para idosos e pessoas com necessidades especiais, Unidades de Saúde da Família, entre outros. Surgiu assim, a necessidade de um sistema que fornecesse suporte às atividades realizadas pelos alunos, nos mais diversos lugares, permitindo que o sistema fosse acessível pela Web e por dispositivos móveis.

Em função do exposto, o Grupo de Computação Ubíqua (GCU) da UFSCar desen-

volveu o Portfólio Reflexivo Eletrônico (PRE) (SANTOS et al., 2008) que faz um registro sistemático, sobre a trajetória e as práticas desenvolvidas pelos estudantes nas atividades do Curso de forma contínua, reflexiva, e por meio de dispositivos computacionais. Este portfólio foi utilizado como estudo de caso para validar a arquitetura de segurança proposta.

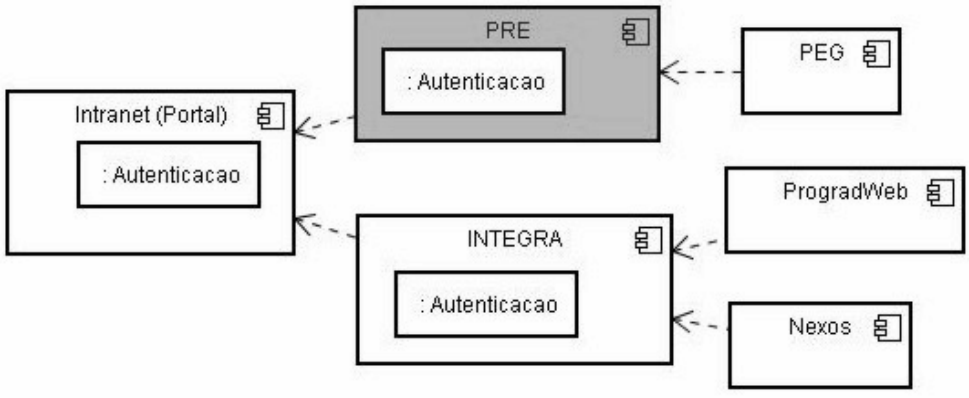

Figura 8.1: Aplicações para suporte a telemedicina da UFSCar.

Intranet é um portal Web, do qual pode-se autenticar e utilizar os diversos sistemas providos para suporte ao curso de Medicina (ver Figura 8.1). Este portal é composto pelos sistemas PRE, Portfólio Eletrônico para Grupos (PEG), que é um portfólio para gerenciamento de grupos de estudantes baseado em atividades comuns, INTEGRA (antigo Sistema Integrado de Gestão Acadêmica - SIGA), que é um sistema de gestão acadêmica para currículos integrados, *ProgradWeb*, que é o sistema de gestão acadêmica (e.g., notas, matrículas) da UFSCar, e Nexos, para planejamento e disponibilização das disciplinas.

É interessante que o sistema gerenciador central (*Intranet*) possua o controle dos usuários, objetivando a autenticação, mas o *Intranet*, o PRE e o INTEGRA possuem bases de usuários próprias, assim como o GSSP. Desta forma, torna-se útil um mecanismo que permita a realização da autenticação dos usuários nestes diversos sistemas da forma *Single Sign-On (SSO)*. No entanto, é necessário que exista um atributo em cada uma destas bases, que seja comum a todos os mecanismos de autenticação. Analisando-se as bases, encontrou-se o número Cadastro de Pessoal Física (CPF), que

foi utilizado para esta integração, como pode ser visto na Figura 8.2.

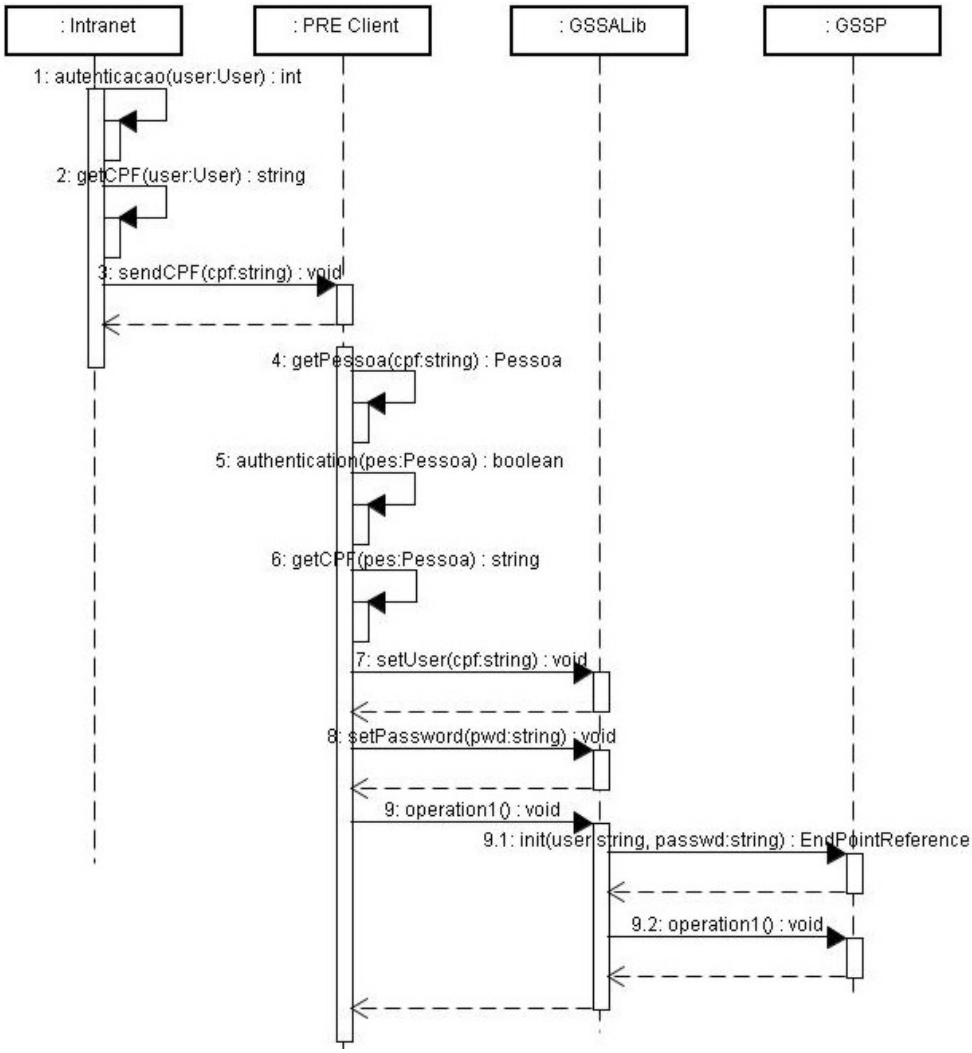

Figura 8.2: SSO no ambiente da telemedicina.

O PRE foi desenvolvido baseado na tecnologia WS para permitir que diferentes aplicações clientes (e.g., aplicação Web ou para dispositivo móvel) utilizem o mesmo Provedor de serviços do Portfólio. O GSSP foi empregado no PRE a fim de possibilitar um controle de acesso refinado às diversas funcionalidades providas por esse Portfólio.

Na Figura 8.3 observa-se as camadas da aplicação Cliente do PRE (*PREClient*), que é acessível via Web.

O PRE possui uma camada de apresentação desenvolvida em *Java Server Faces*

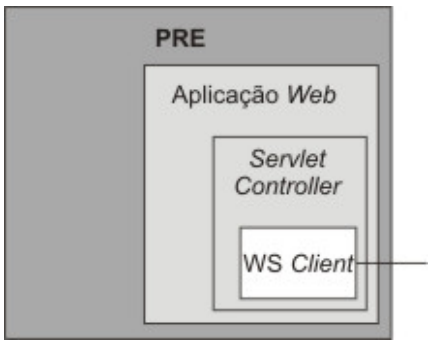

Figura 8.3: Camadas do PRE acessível via Web.

(JSF) e *HyperText Markup Language* (HTML). A camada de controle que é reponsável por prover informações à camada de apresentação utiliza as tecnologias *Java Server Pages* (JSP) e *Servlets*. A classe *ServletController* desta camada utiliza o módulo *GS Client* que é responsável pela chamada às operações do *GS Provider*. Neste módulo foram incluídas a biblioteca *GSSALib* necessária ao uso do GSSP, que faz a autenticação do usuário e permite o acesso às operações disponibilizadas pelo PREProvider, e a configuração do uso das camadas do GSSP.

Os papéis previstos por PRE são: estudante, professor e administrador. Com a finalidade de testar-se o acesso ao Portfólio, definiu-se que os estudantes podem acessar as operações referentes à criação e visualização de documentos, os professores podem acessar as operações de visualização de documentos, e os administradores têm acesso a todas as operações, incluindo as de exclusão de documentos.

Armazenou-se em CAS registros sobre todos os alunos, professores e administradores que acessam o sistema. Depois, estes usuários foram inseridos em grupos, de acordo com os papéis desempenhados no PRE. Cadastraram-se também os recursos e as ações que se pode executar sobre eles. Posteriormente, criaram-se as regras de controle de acesso descritas acima, especificando quais papéis têm direito de executar cada ação específica sobre cada recurso.

Há um exemplo destas informações armazenadas em CAS no Apêndice D.

GSSP foi validado com esta aplicação no Portfólio. O *Proxy* funcionou de forma eficaz, controlando o acesso às operações do PRE.

Resumo do Capítulo 8

Este Capítulo apresentou o uso do GSSP para o controle de segurança de uma aplicação da telemedina.

Uma aplicação da telemedicina chamada PRE faz parte do ambiente de telemedicina gerenciado pelo portal *Intranet*. Esta aplicação é usada para a gestão da vida acadêmica dos alunos do Curso de Medicina da UFSCar, e teve suas operações controladas, baseadas nos papéis estudante, professor e administrador, previstos pela aplicação.

No próximo Capítulo será visto os resultados quantitativos obtidos com o empregado do estudo de caso.

CAPÍTULO

9

Análise dos Resultados

"Você nunca sabe que resultados virão da sua ação. Mas se você não fizer nada, não existirão resultados."

MAHATMA GANDHI

Para os testes, utilizou-se uma Grade Computacional envolvendo o domínio administrativo do Departamento de Computação (DC) da UFSCar, baseada no *middleware Globus Toolkit* 4.

9.1 Ambiente de Testes

Esta Grade é formada por 2 clusters fixos chamados *Xeon* e P3, e uma Grade com baixa disponibilidade chamada *Personal*. Cada um destes ambientes é gerenciado por uma *Virtual Organization (VO)*.

As interligações entre elas podem ser vistas na Figura 9.1. A elipse com borda pontilhada simboliza a relação de confiança entre as VOs, utilizando a abordagem

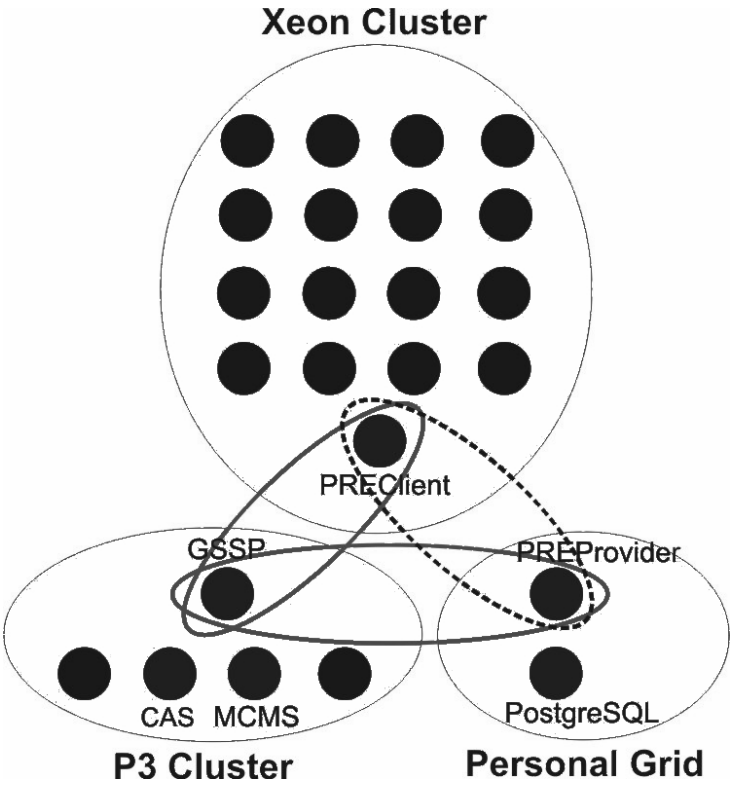

Figura 9.1: Grade Computacional DCGrid.

padrão (*PREClient* acessando *PREProvider* diretamente). Nesta abordagem o *cluster P3* não foi utilizado.

As elipses com borda cheia simbolizam a relação de confiança entre as VOs, utilizando a abordagem do *Proxy*. *Personal* possui uma relação de confiança com P3 e P3 possui relação de confiança com *Xeon*; deste modo, é possível acessar os recursos de outro domínio. Deve-se observar que não há relação de confiança entre *Personal* e *Xeon*.

Xeon possui 16 computadores escravos e um computador coodenador. Todos os computadores possuem processador *Xeon Hiper Theading (HT)* de 2.6 GHz de frequência de CPU e 2 Gb de memória RAM. Os computadores utilizam Sistema Operacional (SO) *Rocks Cluster*[1] 4.2.1 (Cydonia) com *kernel* 2.6.9-42 e GT 4.0.3. Os computadores estão ligados por uma rede Ethernet IEEE 803.2 de 100 Mb/s.

P3 possui 4 computadores escravos e um coordenador. Todos os computadores

[1] `http://www.rocksclusters.org`.

possuem processador Pentium III de 1 GHz, sendo que os escravos possuem 512 Mb de RAM e o coordenador possui 1 Gb. Estes computadores utilizam SO *Rocks Cluster* 4.3 (Mars Hill) com *kernel* 2.6.9-55 e GT 4.0.4. Eles estão interligados por uma rede Ethernet IEEE 803.2 de 100 Mb/s.

Personal envolve 2 computadores, 1 configurado como escravo e coordenador e outro configurado apenas como escravo. O coordenador possui processador *Pentium D core 2 duo* de 3 GHz cada núcleo, 2 Gb de RAM, SO *Slackware*[2] 12.0 com *kernel* 2.6.21, e GT 4.0.6. O computador escravo possui processador *Celeron* de 1,6 GHz, 1 Gb de RAM, SO *Slackware* 12.1 com *kernel* 2.6.24 e GT 4.0.8. Os computadores se comunicam por uma rede Ethernet IEEE 803.2 de 1 Gb/s.

9.2 Configuração do Ambiente de Testes

Nos testes, o computador coordenador de Personal hospedou a aplicação cliente do PRE (*PREClient*), o computador coordenador de P3 hospedou GSSP, o computador coordenador de Xeon hospedou a implementação do portfólio (*PREProvider*) e um computador escravo de Xeon hospedou o banco de dados *PostgreSQL* utilizado pelo *PREProvider*. Os serviços MCMS e CAS foram instalados em computadores escravos de P3. Desta forma, tanto o *GS Provider* quanto GSSP precisaram utilizar a rede ao requisitar os serviços.

CAS foi configurado para confiar em credenciais de usuário provenientes de *Personal*, e MCMS foi configurado através do arquivo de autorização *myproxy-server.config* para permitir armazenar e retornar credenciais provenientes de *Personal* e *P3*.

Foram controladas 10 operações fornecidas por PRE, sendo que todas estas possuiam acesso a banco de dados. Mediu-se o tempo de cada operação e fez-se uma média de tempo de requisição para esta aplicação.

Para um melhor desempenho do *Globus WS Container (GWSC)* onde GSSP está hospedado, a *Java Virtual Machine (JVM)* foi configurada para executar com 512MB de memória *heap*, por meio do parâmetro '-Xmx512M', e para executar com alta pri-

[2] `http://www.slackware.com`.

oridade, utilizando-se o parâmetro '-server'. Para que estes parâmetros não precisem ser passados manualmente ao iniciar o GWSC, configura-se a variável de ambiente, chamada $GLOBUS_OPTIONS, com estes valores.

9.3 Análise de Fraudes e Desempenho

Em relação a fraudes no sistema, considerou-se a utilização do mecanismo autenticação sem transmissão segura, o uso do mecanismo de autorização sem realização de autenticação, e/ou a não utilização de uma camada habilitada.

Com GSSP, o acesso não autorizado é impedido porque, para isso acontecer, seria necessário saltar linhas da execução do código sequencial da implementação da operação em GSSP inseridos pelo Aspecto, o que não é possível. Usuários não autenticados também não podem acessar as operações do GS Provider gerenciado, pois esta autenticação ocorre somente na instância do GSSP e, caso esta não seja realizada ou haja qualquer problema, não é permita a criação do objeto referente ao GSSP, necessário à execução das operações. Não há como burlar o mecanismo de *logging* pois, para isso, também seria necessário saltar as linhas da execução do código sequencial da implementação da operação em GSSP inseridos pelo Aspecto.

Seria possível burlar os mecanismos de autenticação e a configuração de transferência segura realizados no cliente, estendendo-se as classes de *GSSALib* que realizam estas tarefas, não executando a operação *Init* e não configurando os parâmetros de *Stub*. Entretanto, estas classes foram criadas como do tipo "*final*", não permitindo serem estendidas.

A Figura 9.2 apresenta os tempos médios de acesso às operações de *PREProvider*.

Sem uso do *proxy*, o tempo médio foi de 19,57 milissegundos (ms), mas com uso das camadas de autorização e autenticação o tempo médio fica em 26 ms. Com o incremento de transferência segura o tempo médio subiu para 30,22 ms e com o acréscimo do mecanismo *Logging* o tempo médio foi 31,77 ms. Desta forma, é visto que o tempo de acesso às operações do PRE com uso da abordagem *proxy*, que é apenas uma dentre inúmeras possíveis implementações de GSSA, sofre acréscimo de

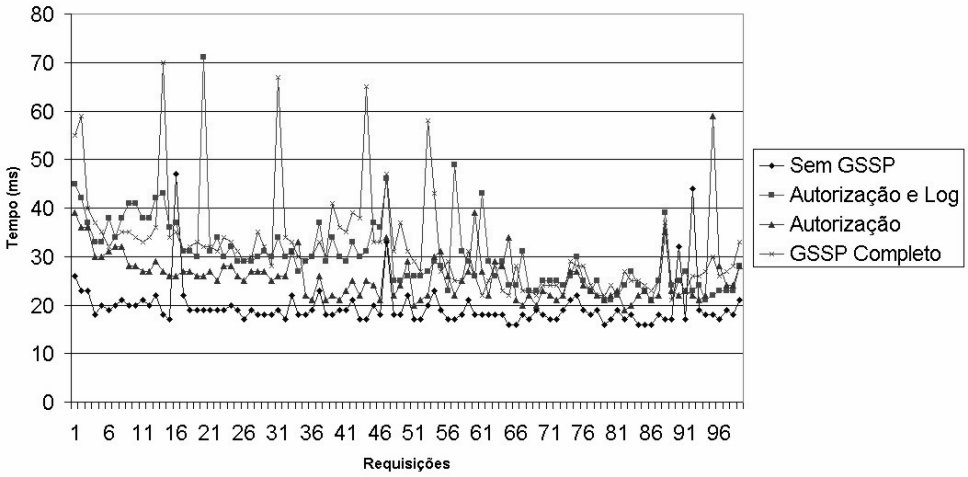

Figura 9.2: Medidas de tempo entre diversas requisições.

cerca de 37% para requisições pontuais.

Já a Figura 9.3 apresenta os tempos médios de acesso às operações, obtidos por meio de requisições simultâneas.

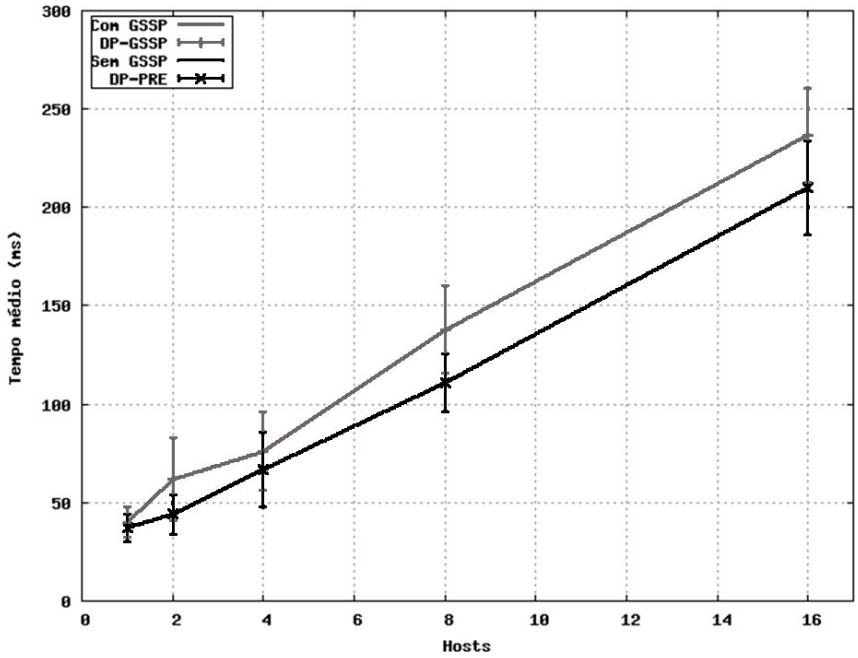

Figura 9.3: Teste de *stress* por meio de requisições simultâneas.

Foram realizadas requisições simultâneas advindas de 1, 2, 4, 8 e 16 computadores. No ambiente com 16 computadores, onde obviamente realizou-se um maior número de requisições simultâneas, sem uso do *proxy*, o tempo médio das requi-

sições foi de 210 ms. Já com o seu uso, o tempo subiu para 237 ms, havendo assim um acrescimo de 11%. Um ponto importante a se observar é que houve um aumento linear do tempo de requisição comparando-se o acesso direto às operações do PRE com o uso da abordagem *proxy*. Desta forma, o *proxy* não apresentou-se como um gargalo na comunicação, sendo considerado escalável.

Resumo do Capítulo 9

Este Capítulo apresentou o ambiente de testes e os resultados obtidos com a solução desenvolvida.

Utilizou-se 2 cluster *Xeon* e P3 e uma Grade de baixa disponibilidade chamada *Personal*, para os testes. Cada um destes ambientes define uma VO. Sem uso do *proxy* o tempo médio de acesso as operações do GS Provider de PRE foi de 19,57 milissegundos (ms), enquanto que com o uso de GSSP o tempo médio ficou em 31,77 ms. Utilizando-se 16 computadores realizando requisições simultâneas, sem utilizar GSSP o tempo médio ficou em 210 ms, contra os 237 ms com uso do GSSP.

O próximo capítulo apresenta as considerações finais e aponta para trabalhos futuros.

CAPÍTULO

10

Considerações Finais

"Eu cheguei a conclusão que eu dei o melhor de mim."

EIICHIRO ODA

Grid Service Security Proxy (GSSP) mostrou-se eficiente no controle de segurança do Portfólio Reflexivo Eletrônico (PRE). De forma geral, GSSP funcionou de forma transparente aos *Grid Services (GSs) Providers*, exigindo mínimas alterações na aplicação do usuário para permitir um controle de acesso às operações dos *Grid Services* de Grades Computacionais baseadas no *middleware Globus Toolkit*.

GSSP introduziu importantes conceitos no acesso a GSs, como o uso de notificações para sincronia dos direitos dos usuários, uso de linguagem orientada a aspectos para controle de acesso e *logging* de GSs independentemente do *Container* utilizado, e controle de acesso aos recursos de *GS Providers*.

Dada a estratégia de implementação adotada, de não interferir com o código dos *Grid Service Providers*, a solução apresentada não garante que os acessos às operações sejam realizados sempre com segurança.

Usando a implementação da *Grid Service Security Architecture (GSSA)*, contudo, a segurança especificada é provida. Com GSSP, o acesso não autorizado e o mecanismo de *logging* é dificultado porque, para isso acontecer, seria necessário a modificação da implementação do GSSP, evitando as chamadas as funcionalidades de segurança, o que não é possível. Usuários não autenticados também não podem acessar as operações do *GS Provider* gerenciado, pois esta operação ocorre somente na instância do GSSP, necessária à execução das operações.

Também é dificultado a burlargem dos mecanismos de autenticação e a configuração de transferência segura realizadas no Cliente, pois as classes de *GSSALib* foram criadas de forma que não podem ser estendidas.

Deste modo, GSSA apresentou-se como uma arquitetura eficiente para segurança de GS e também tornou modular as questões de segurança das aplicações clientes. As análises de desempenho mostraram pouca perda de desempenho no uso do GSSP, tanto em requisições pontuais quanto em requisições simultâneas. O ajuste das camadas a serem utilizadas mostrou também a adaptabilidade da implementação da arquitetura no uso de diferentes níveis de segurança.

10.1 Trabalhos Futuros

Como trabalhos futuros é indicado que o controle de acesso seja realizado sobre os parâmetros das operações, após ser realizado o controle de acesso sobre a operação, pela implementação da arquitetura. Isso permite especificar, por exemplo, que um parâmetro numérico qualquer, possa ou não conter um valor entre valores pré-determinados, ou que o valor deste parâmetro seja menor, maior, ou igual a um valor especificado. Para tipos de dados *strings*, o controle poderia ser, por exemplo, analizando se o valor do parâmetro é igual, começa por, termina por, ou contém a *string* especificada.

Outro possível trabalho seria a modificação do *Community Authorization Service (CAS)* para que este sistema de autorização suporte hierarquia de papéis, verificando se os papéis herdam direitos de outros papéis, e concessão de direitos a usuários

individualmente, permitindo uma maior granularidade dos direitos de acesso. Estas funcionalidades podem ser úteis, no ambiente em que, por exemplo, um papel Enfermeira pode precisar temporariamente herdar alguns dos direitos do papel Médico.

Referências Bibliográficas

ALFIERI, R. et al. VOMS, an authorization system for virtual organizations. **Proceedings of the 1st European Across Grids Conference**, p. 13–14, 2003.

BLIGH, J. Problem-based learning in medicine: an introduction. **Postgraduate Medical Journal**. v. 71, p. 323–326, 1995.

BOSWORTH, A. Developing web services. **Proceedings of the 17th International Conference on Data Engineering**, 2001.

BUTLER, R. et al. A national-scale authentication infrastructure. **IEEE Computer**. v. 33(12), p. 60–66, 2000.

CHADWICK, D. W.; OTENKO, A. The PERMIS X.509 role based privilege management infrastructure. **Proceedings of the 7th ACM Symposium on Access Control Models and Technologies**, 2002.

CIRNE, W.; NETO, E. S. Grids computacionais: da computação de alto desempenho a serviços sob demanda. **XXIII Simpósio Brasileiro de Redes de Computadores**. v. 18, p. 51, 2005.

Conselho de Ensino e Pesquisa da Universidade Federal de São Carlos. **Curso de Medicina da UFSCar**. Available: http://www2.ufscar.br/graduacao/medicina.php.

CZAJKOWSKI, K. et al. **From OGSI to WS Resource Framework: Refactoring and Evolution**. Fujitsu Limited, International Business Machines Corporation and The University of Chicago, 2004. Relatório técnico.

FERREIRA, L. et al. **Introduction to Grid Computing with Globus**, 2. ed. Number 0738427969. North Castle Drive Armonk, NY, USA: Red Books, 2003., 268 p.

FERREIRA, L. et al. **Grid Services Programming and Application Enablement**, 1. ed. Number 0738498033. North Castle Drive Armonk, NY, USA, 2004., 280 p.

FOSTER, I. **A Globus Primer**, 2005a. Relatório técnico.

FOSTER, I. Globus toolkit version 4: Software for service-oriented systems. **International Federation for Information Processing (IFIP'05)**, p. 2–13, 2005b.

FOSTER, I. et al. A security architecture for computational grids. **Proceedings of the 5th ACM Conference on Computer and Communications Security Conference**, p. 83–92, 1998.

FOSTER, I. et al. Grid services for distributed system integration. **IEEE Computer Society**. v. 35(6), p. 37–46, 2002a.

FOSTER, I. et al. The physiology of the grid: An open grid services architecture for distributed systems integration. **Open Grid Service Infrastructure WG**, 2002b.

FOSTER, I.; KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. **International Journal of Supercomputer Applications**. v. 11(2), p. 115–129, 1998a.

FOSTER, I.; KESSELMAN, C. **The Grid: Blueprint for a Future Computing Infrastructure**, 2. ed. Number B001CHXIYK. Morgan Kaufmann Publishers, 1998b., 748 p.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. **International Journal of Supercomputer Applications**. v. 15(3), p. 200–222, 2001.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading, MA: Addison Wesley, 2005.

GAWOR, J. **Java Web Services Core**. Available: http://www.globus.org/alliance/events/sc06/JWSCore.pdf.

GLOBUS SECURITY TEAM, 2009a. **GT 4.0: Authorization Framework**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/authzframe/index.pdf.

GLOBUS SECURITY TEAM, 2009b. **GT 4.0: CAS**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/cas/index.pdf.

GLOBUS SECURITY TEAM, 2009c. **GT 4.0: Delegation Service**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/delegation/index.pdf.

GLOBUS SECURITY TEAM, 2009d. **GT 4.0: GSI-OpenSSH**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/openssh/index.pdf.

GLOBUS SECURITY TEAM, 2009e. **GT 4.0: MyProxy**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/myproxy/index.pdf.

GLOBUS SECURITY TEAM, 2009f. **GT 4.0: Security: Message and Transport Level Security**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/message/index.pdf.

GLOBUS SECURITY TEAM, 2009g. **GT 4.0: Security: Pre-Web Services Authentication and Authorization**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/prewsaa/index.pdf.

GLOBUS SECURITY TEAM, 2009h. **GT 4.0: SimpleCA**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/security/simpleca/index.pdf.

GLOBUS TEAM, 2009a. **GT 4.0: GridFTP**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/data/gridftp/index.pdf.

GLOBUS TEAM, 2009b. **GT 4.0: Java WS Core**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.0/common/javawscore/index.pdf.

GLOBUS TEAM, 2009c. **Installing GT 4.0**. GLOBUS ALLIANCE, http://www.globus.org/toolkit/docs/4.2/4.2.0/admin/install/installingGT.pdf.

GONÇALVES, L. et al. Sistema de monitoramento/gerência de recursos e de segurança para grades computacionais baseadas no globus toolkit. **IV Workshop de Computação em Grid e Aplicações (WCGA'06)**, 2006.

Governo Eletrônico. **Guia de Estruturação e Administração de Cluster e Grid**, 1. ed. Brasília-DF: Secretaria de Logística e Tecnologia da Informação, 2006., 454 p.

HE, H.; GONG, P.; LIN, Z. Creation and applications of grid services for pervasive computing. **Proceedings of the 1st IEEE International Symposium on Pervasive Computing and Applications**. v. 212–217, 2006.

JACOB, B. et al. **Introduction to Grid Computing**, 1. ed. Number 0738494003. North Castle Drive Armonk, NY, USA: RedBooks, 2005., 248 p.

JANA, D. et al. Interoperability and security issues of grid services for ubiquitous computing. **IEEE International Conference on Computer Systems and Applications**, p. 1114–1117, 2006.

JUNG, H. et al. Dynamic and fine-grained authentication and authorization architecture for grid computing. **Lecture Notes in Computer Science**, p. 179 – 186, 2005.

JUNIOR, J. D. R.; KON, F. Segurança em grades computacionais. **Mini-curso de segurança em Grades. Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP)**, 2007.

LADDAD, R. **AspectJ in Action: Practical Aspect-Oriented Programming**. Number 1-930110-93-6. Manning, 2003.

LANG, B. et al. A multipolicy authorization framework for grid security. **Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)**, 2006.

LANGELLA, S., 2005. **caGrid Security Architecture**. Life Sciences Grid Workshop, 5. ed.

MALER, E.; MISHRA, P.; PHILPOTT, R. **Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1**.

MOROHOSHI, H. et al. A bridge linking ubiquitous devices and grid services. **21st International Conference on Advanced Information Networking and Applications (AINA'07)**, p. 747–753, 2007.

NAGARATNAM, N.; other. The security architecture for open grid services, 2002.

NOVOTNY, J.; TUECKE, S.; WELCH, V. An online credential repository for the grid: Myproxy. In: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, 2001., 2001. p. 104–111.

PAPAZOGLOU, M. Service-oriented computing: concepts, characteristics and directions. **Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)**, p. 3–12, 2003.

PEARLMAN et al. The community authorization service status and future. **Computing in High Energy Physics (CHEP'03)**, p. 1 – 9, 2003.

PEARLMAN, L.; FOSTER, I. A community authorization service for group collaboration. **Proceedings of the 3rd International IEEE Workshop on Policies for Distributed Systems and Networks**, p. 50–59, 2002.

RADIC, B.; KAJIC, V.; IMAMAGIC, E. Optimization of data transfer for grid using gridFTP. **Proceedings of the 29th International Conference on Information Technology Interfaces (ITI 2007)**, p. 709–715, 2007.

RHEM, J. **Problem-Based Learning: An Introduction**. Available: http://www.ntlf.com/html/pi/9812/pbl_1.htm.

ROSSET, V.; WESTPHALL, C. M.; FILIPPIN, C. V. Modelo autorização e distribuição de direitos de acesso para sistemas DRM. **IV Congresso Brasileiro de Computação (CBComp). Sistemas Paralelos e Distribuídos**, p. 480–485, 2004.

SANDHUF, R. S. et al. Role-based access control: A multi-dimensional view. **Proceedings of the 10th Annual Computer Security Applications Conference**, p. 54–62, 1994.

SANTOS, H. F. et al. A ubiquitous computing environment for medical education. **Proceedings of the 23st ACM Symposium on Applied Computing (SAC'08)**. v. 2, p. 1395–1399, 2008.

SEBU, L.; CIOCÂRLIE, H. The design of stateful web services based on web service resource framework implemented in globus toolkit 4. **Proceedings of the 8th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)**, 2006.

SHIRASUNA, S. et al. Performance comparison of security mechanisms for grid services. **Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)**, 2004.

SOTOMAYOR, B., 2005. **The Globus Toolkit 4 Programmer's Tutorial**. University of Chicago, http://gdp.globus.org/gt4-tutorial/.

TANEMBAUM, A. S. **Redes de Computadores**, 4. ed. Number 8535211853, 2003., 945 p.

TANG, K. et al. A performance evaluation of web services security. **Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)**, 2006.

THOMPSON, M. et al. Certificate-based access control for widely distributed resources. **Proceeding of the Usenix SecuritySymposium '99**, 1999.

TUECKE, S. et al. **Internet X.509 Public Key Infrastructure (PKI): Proxy Certificate Profile**, 2004. Relatório técnico.

WELCH, V. et al. Security for grid services. **Proceedings of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)**, p. 48–57, 2003a.

WELCH, V. et al. Use of SAML in the community authorization service, 2003b.

WELCH, V. et al. Attributes, anonymity, and access: Shibboleth and globus integration to facilitate grid collaboration. **Proceedings of the 4th Annual PKI R&D Workshop**, 2005.

YU, C.-M.; NG, K.-W. A heterogeneous authorization policy management mechanism for grid environments. **Proceedings of the IEEE International Conference on Multimedia and Ubiquitous Engineering (MUE '07)**, 2007.

APÊNDICE

A

Assertivas SAML da CPR

A seguir estão parte das assertivas SAML contidas na Credencial *Proxy* Restrita obtida do *Community Authorization Service (CAS)*.

Parte do Arquivo: ***/tml/x509up_u1001***

```
1 <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" AssertionID="2104f8d0-8b88
    -11dd-a474-e3b2a27c6a32" IssueInstant="2008-09-26T05:01:07Z" Issuer="O=Grid,OU=
    GlobusTest,OU=simpleCA-monet.dc.ufscar.br,CN=Globus Simple CA" MajorVersion="1"
    MinorVersion="0">
2  <Conditions NotBefore="2008-09-26T05:01:07Z" NotOnOrAfter="2008-09-27T05:01:07Z">
3  </Conditions>
4  <AuthorizationDecisionStatement Decision="Permit" Resource="http://monet/wsrf/
     service/PRE|boolean,validaGrupo,int">
5   <Subject>
6    <NameIdentifier Format="#X509SubjectName" NameQualifier="O=Grid,OU=GlobusTest,OU=
       simpleCA-monet,CN=Claudio Rodolfo Sousa de Oliveira">/O=Grid/OU=GlobusTest/OU
       =simpleCA-monet/CN=Claudio Rodolfo Sousa de Oliveira
7    </NameIdentifier>
8    <SubjectConfirmation>
9     <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:am:X509-PKI</ConfirmationMethod>
```

```
10      </SubjectConfirmation>
11     </Subject>
12     <Action Namespace="ws">read
13     </Action>
14    </AuthorizationDecisionStatement>
15    <AuthorizationDecisionStatement Decision="Permit" Resource="http://monet/wsrf/
          service/PRE|void,inserirRecado,String">
16     <Subject>
17      <NameIdentifier Format="#X509SubjectName" NameQualifier="O=Grid,OU=GlobusTest,OU=
            simpleCA-monet,CN=Claudio Rodolfo Sousa de Oliveira">/O=Grid/OU=GlobusTest/OU
            =simpleCA-monet/CN=Claudio Rodolfo Sousa de Oliveira
18      </NameIdentifier>
19      <SubjectConfirmation>
20       <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:am:X509-PKI</ConfirmationMethod>
21      </SubjectConfirmation>
22     </Subject>
23     <Action Namespace="ws">read
24     </Action>
25    </AuthorizationDecisionStatement>
26    <AuthorizationDecisionStatement Decision="Permit" Resource="String,listarDocumentos
          ,int">
27     <Subject>
28      <NameIdentifier Format="#X509SubjectName" NameQualifier="O=Grid,OU=GlobusTest,OU=
            simpleCA-monet,CN=Claudio Rodolfo Sousa de Oliveira">/O=Grid/OU=GlobusTest/OU
            =simpleCA-monet/CN=Claudio Rodolfo Sousa de Oliveira
29      </NameIdentifier>
30      <SubjectConfirmation>
31       <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:am:X509-PKI</ConfirmationMethod>
32      </SubjectConfirmation>
33     </Subject>
34     <Action Namespace="ws">read
35     </Action>
36    </AuthorizationDecisionStatement>
37    <AuthorizationDecisionStatement Decision="Permit" Resource="Pessoa">
38     <Subject>
39      <NameIdentifier Format="#X509SubjectName" NameQualifier="O=Grid,OU=GlobusTest,OU=
```

```
        simpleCA–monet,CN=Claudio Rodolfo Sousa de Oliveira">/O=Grid/OU=GlobusTest/OU
        =simpleCA–monet/CN=Claudio Rodolfo Sousa de Oliveira
40    </NameIdentifier>
41    <SubjectConfirmation>
42     <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:am:X509–PKI</ConfirmationMethod>
43    </SubjectConfirmation>
44   </Subject>
45   <Action Namespace="resource">get
46   </Action>
47  </AuthorizationDecisionStatement>
48  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
49   <ds:SignedInfo>
50    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC–xml–c14n
        –20010315">
51    </ds:CanonicalizationMethod>
52    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa–sha1">
53    </ds:SignatureMethod>
54    <ds:Reference URI="">
55     <ds:Transforms xmlns:signs="urn:oasis:names:tc:SAML:1.0:assertion">
56      <ds:Transform Algorithm="http://www.w3.org/2002/06/xmldsig–filter2">
57       <dsig–xpath:XPath xmlns:dsig–xpath="http://www.w3.org/2002/06/xmldsig–filter2"
            Filter="intersect">here()/ancestor::signs:Assertion[1]
58       </dsig–xpath:XPath>
59       <dsig–xpath:XPath xmlns:dsig–xpath="http://www.w3.org/2002/06/xmldsig–filter2"
            Filter="subtract">here()/ancestor::ds:Signature[1]
60       </dsig–xpath:XPath>
61      </ds:Transform>
62      <ds:Transform Algorithm="http://www.w3.org/2001/10/xml–exc–c14n#">
63       <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml–exc–c14n#"
           PrefixList="code ds kind rw saml samlp signs #default xsd xsi">
64       </ec:InclusiveNamespaces>
65      </ds:Transform>
66     </ds:Transforms>
67     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
68     </ds:DigestMethod>
69     <ds:DigestValue>S2igiNFK7TF+bT6ETh6tmpQdNyU=</ds:DigestValue>
```

```
70      </ds:Reference>
71    </ds:SignedInfo>
72    <ds:SignatureValue>
73      Gw7urUWxtr+cYBJJQzIWzhR6UTu+bZKxzayVoLe/gHeLvvyrQ6cHIGoxb/ZId8Gtvr7qo2puzqNl
74      ozheSVOx9Td4V6Pnx/IU6VXbbSYxdEpWC0tBCC35vRkJmGvFMBbnBjZoCCZj3qzRAwuvSp3/6CCz
75      ItieLXwLF3a9Y3RWWDc=
76    </ds:SignatureValue>
77    <ds:KeyInfo>
78      <ds:X509Data>
79        <ds:X509Certificate>
80          MIICXDCCAcWgAwIBAgIBATANBgkqhkiG9w0BAQQFADBlMQ0wCwYDVQQKEwRHcmlkMRMwEQYDVQQL
81          EwpHbG9idXNUZXN0MSQwIgYDVQQLExtzaW1wbGVDQS1tb25ldC5kYy51ZnNjYXIuYnIxGTAXBgNV
82          BAMTEEdsb2J1cyBTaW1wbGUgQ0EwHhcNMDgwNzA1MTYyMzIyWhcNMDkwNzA1MTYyMzIyWjBsMQ0w
83          CwYDVQQKEwRHcmlkMRMwEQYDVQQLEwpHbG9idXNUZXN0MSQwIgYDVQQLExtzaW1wbGVDQS1tb25l
84          dC5kYy51ZnNjYXIuYnIxIDAeBgNVBAMTF2hvc3QvbW9uZXQuZGMudWZzY2FyLmJyMIGfMA0GCSqG
85          SIb3DQEBAQUAA4GNADCBiQKBgQDguBkcFTlS3DbjSENOzG/UJgahHDX0qFbEaJZYwMYUsvw0HtJn
86          hPg12QnzfDECg5L1sJakMhnBRwrh4OwA48tU6TQqIlHJ7Gh4ZBXIVrHjNpl348/UJvO4cN0tNerd
87          GgLhuKLmOVmq9HFGflNQPPNkAcHQSg99mO2/XmkJd4GoLwIDAQABoxUwEzARBglghkgBhvhCAQEE
88          BAMCBPAwDQYJKoZIhvcNAQEEBQADgYEAGiOEH3GCWMQ14hZI+KTjN+Jwx/O3Fq0qx5wDC+Ru9E/o
89          Ja14nxEOHqOpBc9yzCG6HlNZNDkEMwMlDqBY4N36h6UmDLl3SfaibqySX/y7Ssd3EgTK7OiHU26C
90          6B6uP7OEL4xVAnxZxZ+d9QyyR9M4X3SxBwd5R5iK6o7IszUd2+o=
91        </ds:X509Certificate>
92      </ds:X509Data>
93    </ds:KeyInfo>
94  </ds:Signature>
95 </Assertion>
```

APÊNDICE

B

Aspecto para Autorização e *Logging*

Código em AspectoJ do *framelet* de autorização, das instâncias do *framelet*, e da implementação do módulo *Logging* de GSSP.

B.1 *Framelet* de Autorização

Aspecto abstrato do *framelet* para o controle de autorização.

Arquivo: ***AuthorizationAspect.aj***

```
1  package br.ufscar.dc.aspectj;
2
3  import br.ufscar.dc.gcu.gssp.provider.impl.GSSPService;
4
5  privileged public abstract aspect AuthorizationAspect {
6      String resourceLocal;
7
8      abstract pointcut gsspMethodsInvocation(IService service);
9
10     abstract before(IService service):gsspMethodsInvocation(service);
11
```

```
12      after(IService service):gsspMethodsInvocation(service) {
13          String method = thisJoinPoint.getStaticPart().getSignature().getName();
14          logger(service.class);
15          service.log.gravar(resourceLocal);
16      }
17  }
```

Aspecto abstrato do *framelet* para o controle de autorização de serviços.

Arquivo: ***ServiceAuthorizationAspect.aj***

```
1   package br.ufscar.dc.aspectj;
2
3   import br.ufscar.dc.gssp.server.GSSPService;
4   import br.ufscar.dc.gssp.server.NotPermittedException;
5   import br.ufscar.dc.pre.server.stubs.PREServiceLocator;
6
7   public aspect ServiceAuthorizationAspect extends AuthorizationAspect {
8
9       abstract pointcut gsspMethodsInvocation(GSSPService ser);
10
11      before(IService service):gsspMethodsInvocation(service) {
12          //return
13          resourceLocal = thisJoinPoint.getSignature().getDeclaringTypeName();
14          //operation name
15          resourceLocal = resourceLocal + "," + thisJoinPoint.getStaticPart().
                getSignature().getName();
16          //arguments
17          String[] array = thisJoinPoint.getArgs();
18          for (int i = 0; i < array.lenght; i++)
19              resourceLocal = + resourceLocal + "," + array[i] ;
20
21          if(!service.verify(service.resource.user, "ws", "read", service.loc.
                getPREAddress() +"/"+ resourceLocal))
22              throw new NotPermittedException("Acesso não autorizado a " +
                    resourceLocal);
23      }
24
```

```
25 }
```

Aspecto abstrato do *framelet* para o controle de autorização de serviços.

Arquivo: ***ResourceAuthorizationAspect.aj***

```
1  package br.ufscar.dc.aspectj;
2
3  import br.ufscar.dc.gssp.server.NotPermittedException;
4  import br.ufscar.dc.pre.server.PREResource;
5  import br.ufscar.dc.pre.server.stubs.PREServiceLocator;
6
7  public abstract aspect ResourceAuthorizationAspect extends AuthorizationAspect {
8
9      abstract pointcut gsspMethodsInvocation(GSSPResource res);
10
11     before(IService resource):gsspMethodsInvocation(resource) {
12
13         String name += thisJoinPoint.getStaticPart().getSignature().getName;
14         resourceLocal += name.substring(3, name.legth);
15
16         if(!service.verify(service.resource.user, "resource", name.substring(1, 3);,
               resource.loc.getPREAddress() +"/"+ resourceLocal))
17             throw new NotPermittedException("Acesso não autorizado a " + resource);
18     }
19 }
```

B.2 Especialização do Aspecto para o PRE

Instância do *framelet* para o controle de autorização das operações do Serviço.

Arquivo: ***GSSPServiceAuthorizationAspect.aj***

```
1  package br.ufscar.dc.aspectj;
2
3  import br.ufscar.dc.proxy.server.GSSPService;
4
5  public aspect GSSPServiceAuthorizationAspect extends ServiceAuthorizationAspect {
6
```

```
7           pointcut gsspMethodsInvocation(GSSPService ser):
8           execution (public * GSSPService.*(..)) && !execution(public GSSPService.new
                (..)) && target(ser);
9
10  }
```

Instância do *framelet* para o controle de autorização do Recurso do PRE.

Arquivo: ***PREResourceAuthorizationAspect.aj***

```
1   package br.ufscar.dc.aspectj;
2
3   import br.ufscar.dc.pre.server.PREResource;
4
5   public aspect PREResourceAuthorizationAspect extends ResourceAuthorizationAspect {
6
7       pointcut gsspMethodsInvocation(PREResource res):
8           execution(public * PREResource.get*(..)) &&(public * PREResource.set*(..))
                && target(res));
9   }
```

APÊNDICE

C

Registro de *Logs* de Segurança

Este apendice apresenta um modelo dos arquivos de *log*, gerados pelo módulo *Logging* de GSSA ,que podem auxiliar sistemas de auditoria e *accountability*.

C.1 Arquivo do módulo *Loggging*

Abaixo está o arquivo de *log* diário gerado por GSSP.

Neste arquivo as colunas representam respectivamente o tipo de evento, o *timestamp* de quando o evento ocorreu, a máquina requisitante, o usuário que pretendeu executar a operação, qual a operação chamada e a descrição da operação.

Arquivo: ***2009-04-10.log***

```
1 0 1239337825 oiticica claudio GSSP/boolean,init,String,String "Autenticação"
2 2 1239337992 oiticica claudio PRE/String,visualizarDocumento,int "Visualiza
    documento"
3 2 1239338111 oiticica claudio PRE/String,listarDocumentos,int "Lista documento"
4 0 1239338243 rambo helio GSSP/boolean,init,String,String "Autenticação"
5 2 1239338387 oiticica claudio PRE/int,criarDocumento,CriarDocumento "Cria documento"
```

```
2 1239338502 rambo helio PRE/String,visualizarDocumento,intg "Visualiza documento"
2 1239338661 rambo helio PRE/String,listarDocumentos,int "Lista documento"
3 1239338837 rambo helio PRE/int,criarDocumento,CriarDocumento "Cria documento"
0 1239338969 monet admin GSSP/boolean,init,String,String "Autenticação"
3 1239339112 rambo helio PRE/void,inserirRecado,String "Insere recado"
2 1239339284 monet admin PRE/boolean,apagarDocumento,int "Apaga documento"
1 1239339405 rambo helio GSSP/boolean,destroy,EndPointReference "Logout"
1 1239339574 monet admin GSSP/boolean,destroy,EndPointReference "Logout"
3 1239339742 oiticica claudio PRE/boolean,apagarDocumento,int "Apaga documento"
2 1239339825 oiticica claudio PRE/boolean,criarDocumentoOffline,String "Cria
    documento offline"
2 1239340017 oiticica claudio PRE/void,inserirRecado,String "Insere recado"
1 1239340325 oiticica claudio GSSP/boolean,destroy,EndPointReference "Logout"
4 1239340735 oiticica claudio PRE/boolean,apagarDocumento,int "Usuário não
    autenticado"
```

C.2 Arquivo do módulo *History*

Abaixo vê-se o arquivo de xml gerado a partir das informações do WS Resource do usuário claudio. Isto significa que este usuário possuiu estes direitos de 01h30m25s do dia 10/04/2009 até às 02h18m55s do dia 10/04/2009.

Arquivo: ***2009-04-10-02-18-55-claudio-2009-04-10-01-30-25.xml***

```
<?xml version="1.0" encoding="utf-8"?>
<GSSPResourceProperties xmlns:tns="http://www.dc.ufscar.br/namespaces/gssp/gcu/
    GSSPService_instance">
        <tns:DN>claudio</tns:DN>
        <tns:Rights[0]>ws</tns:Rights[0]>
        <tns:Rights[0]>read</tns:Rights[0]>
        <tns:Rights[0]>GSSP/boolean,init,String,String</tns:Rights[0]>
        <tns:Rights[1]>ws</tns:Rights[1]>
        <tns:Rights[1]>read</tns:Rights[1]>
        <tns:Rights[1]>PRE/String,visualizarDocumento,int</tns:Rights[1]>
        <tns:Rights[2]>ws</tns:Rights[2]>
        <tns:Rights[2]>read</tns:Rights[2]>
```

```
12          <tns:Rights[2]>PRE/String,listarDocumentos,int</tns:Rights[2]>
13          <tns:Rights[3]>ws</tns:Rights[3]>
14          <tns:Rights[3]>read</tns:Rights[3]>
15          <tns:Rights[3]>PRE/int,criarDocumento,CriarDocumento</tns:Rights[3]>
16          <tns:Rights[4]>ws</tns:Rights[4]>
17          <tns:Rights[4]>read</tns:Rights[4]>
18          <tns:Rights[4]>PRE/boolean,criarDocumentoOffline,String</tns:Rights[4]>
19          <tns:Rights[5]>ws</tns:Rights[5]>
20          <tns:Rights[5]>read</tns:Rights[5]>
21          <tns:Rights[6]>PRE/void,inserirRecado,String</tns:Rights[5]>
22          <tns:Rights[6]>ws</tns:Rights[6]>
23          <tns:Rights[6]>read</tns:Rights[6]>
24          <tns:Rights[6]>GSSP/boolean,destroy,EndPointReference</tns:Rights[6]>
25          <tns:Rights[7]>resource</tns:Rights[7]>
26          <tns:Rights[7]>get</tns:Rights[7]>
27          <tns:Rights>Pessoa</tns:Rights[7]>
28  </GSSPResourceProperties>
```

APÊNDICE

D

Configuração das políticas do CAS

Para o uso do *Community Authorization Service (CAS)*, é necessário antes popular seu banco de dados com as políticas de autorização informando quais pápeis de usuários podem executar ações em objetos ou grupo de objetos.

A seguir, estão parte dos comandos utilizados para o uso do CAS no Portfólio Reflexivo Eletrônico (PRE).

Arquivo: ***cas.sh***

```
1 #!/bin/bash
2
3 CONFIG="http://oiticica:8080/wsrf/services/CASService} -s /O=Grid/OU=GlobusTest/OU=
      simpleCA-monet/CN=host/monet"
4
5 #CA
6 cas-enroll $CONFIG trustAnchor defaultTrustAnchor X509 "/O=Grid/OU=GlobusTest/OU=
      simpleCA-monet/CN=Globus Simple CA"
7 #usuarios
8 cas-enroll $CONFIG user casadmin claudio "/O=Grid/OU=GlobusTest/OU=simpleCA-monet/CN
      =Claudio Rodolfo Sousa de Oliveira" defaultTrustAnchor
9 cas-enroll $CONFIG user casadmin helio "/O=Grid/OU=GlobusTest/OU=simpleCA-monet/CN=
```

```
    Helio Crestana Guardia" defaultTrustAnchor
10  cas-enroll $CONFIG user casadmin wanderley "/O=Grid/OU=GlobusTest/OU=simpleCA-monet/
    CN=Wanderley Lopes de Souza" defaultTrustAnchor
11  #grupos de usuarios
12  cas-group-admin $CONFIG user create casadmin estudante
13  cas-group-admin $CONFIG user create casadmin docente
14  cas-group-admin $CONFIG user create casadmin administrador
15  #inserir usuarios em grupos
16  cas-group-add-entry $CONFIG user administrador claudio
17  cas-group-add-entry $CONFIG user administrador helio
18  cas-group-add-entry $CONFIG user administrador wanderley
19  cas-group-add-entry $CONFIG user docente helio
20  cas-group-add-entry $CONFIG user docente wanderley
21  cas-group-add-entry $CONFIG user estudante claudio
22  #namespace dos objetos
23  cas-enroll $CONFIG namespace casadmin PRE http://monet.dc.ufscar.br:8080/wsrf/
    services/PRE Exact
24  cas-enroll $CONFIG namespace casadmin PEG http://monet.dc.ufscar.br:8080/wsrf/
    services/PEG Exact
25  cas-enroll $CONFIG namespace casadmin GSSP http://monet.dc.ufscar.br:8080/wsrf/
    services/GSSP Exact
26  #objetos
27  cas-enroll $CONFIG object casadmin "boolean,autenticacao,String,String" PRE
28  cas-enroll $CONFIG object casadmin "String,visualizarDocumento,int" PRE
29  cas-enroll $CONFIG object casadmin "String,listarDocumentos,int" PRE
30  cas-enroll $CONFIG object casadmin "int,criarDocumento,CriarDocumento" PRE
31  cas-enroll $CONFIG object casadmin "boolean,criarDocumentoOffline,String" PRE
32  cas-enroll $CONFIG object casadmin "void,inserirRecado,String" PRE
33  cas-enroll $CONFIG object casadmin "boolean,init,String,String" GSSP
34  cas-enroll $CONFIG object casadmin "boolean,destroy,EndPointReference" GSSP
35  cas-enroll $CONFIG object casadmin "Pessoa" PRE
36  cas-enroll $CONFIG object casadmin "PequenoGrupo" PRE
37  #grupo de objetos
38  cas-group-admin $CONFIG object create casadmin Arquivo
39  cas-group-admin $CONFIG object create casadmin Autor
40  cas-group-admin $CONFIG object create casadmin Coordenador
```

```
41 cas-group-admin $CONFIG object create casadmin Foto
42 cas-group-admin $CONFIG object create casadmin PerfilEstudante
43 cas-group-admin $CONFIG object create casadmin Reuniao
44 cas-group-admin $CONFIG object create casadmin Assunto
45 cas-group-admin $CONFIG object create casadmin Ciclo
46 cas-group-admin $CONFIG object create casadmin Docente
47 cas-group-admin $CONFIG object create casadmin Grupo
48 cas-group-admin $CONFIG object create casadmin Pessoa
49 cas-group-admin $CONFIG object create casadmin Atividade
50 cas-group-admin $CONFIG object create casadmin Conteudo
51 cas-group-admin $CONFIG object create casadmin Estudante
52 cas-group-admin $CONFIG object create casadmin PerfilDocente
53 cas-group-admin $CONFIG object create casadmin Recado
54 #adicionar o objeto ao grupo de objetos
55 cas-group-add-entry $CONFIG object Autor object PRE "boolean,autenticacao,String,
       String"
56 cas-group-add-entry $CONFIG object Recado object PRE "void,inserirRecado,String"
57 cas-group-add-entry $CONFIG object Conteudo object PRE "criarDocumento,
       CriarDocumento"
58 cas-group-add-entry $CONFIG object Conteudo object PRE "criarDocumentoOffline,String
       "
59 cas-group-add-entry $CONFIG object Conteudo object PRE "String,listarDocumentos,int"
60 cas-group-add-entry $CONFIG object Conteudo object PRE "String,visualizarDocumento,
       int"
61 #tipo de servico
62 cas-enroll $CONFIG serviceType casadmin ws
63 cas-enroll $CONFIG serviceType casadmin resource
64 #acões permitidas ao serviço
65 cas-action $CONFIG add ws read
66 cas-action $CONFIG add resource get
67 cas-action $CONFIG add resource set
68 #grupo de ações
69 cas-group-admin $CONFIG serviceAction create casadmin GSSA
70 #adicionando a ação ao grupo
71 cas-group-add-entry $CONFIG serviceAction GSSA ws read
72 cas-group-add-entry $CONFIG serviceAction GSSA resource get
```

```
73 cas-group-add-entry $CONFIG serviceAction GSSA resource set
74 #adicionando direitos
75 cas-rights-admin grant $CONFIG estudante object PRE "boolean,autenticacao,String,
       String" serviceAction ws read
76 cas-rights-admin grant $CONFIG estudante object PRE "resource1" serviceAction
       resource get
77 cas-rights-admin grant $CONFIG professor object PRE "resource2" serviceAction
       resource set
```