UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
MESTRADO EM ENGENHARIA DE TELEINFORMÁTICA

JOÃO CARLOS SOUSA DO VALE

MYGSI – UMA PROPOSTA DE SEGURANÇA PARA GRADES PEER-TO-PEER

FORTALEZA
2006

JOÃO CARLOS SOUSA DO VALE

MYGSI – UMA PROPOSTA DE SEGURANÇA PARA GRADES PEER-TO-PEER

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Engenharia de Teleinformática, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Teleinformática.

Orientador: Prof. Dr. Mário Fiallos Aguilar.
Co-orientadora: Profa. Dra. Rossana Maria de Castro Andrade.

FORTALEZA
2006

MYGSI – Uma Proposta de Segurança para Grades Peer-to-Peer

João Carlos Sousa do Vale

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Engenharia de Teleinformática, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Teleinformática.

Composição da Banca Examinadora:

Prof. Dr. Mário Fiallos Aguilar (Orientador)
Universidade Federal do Ceará - UFC

Profa. Dra. Rossana Maria de Castro Andrade (Co-Orientadora)
Universidade Federal do Ceará – UFC

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará - UFC

Prof. Dr. Antonio Miguel Vieira Monteiro
Instituto Nacional de Pesquisas Espaciais - INPE

Aprovada em ____ de _____ de 2006

AGRADECIMENTOS

Agradeço primeiramente a Deus, pelas oportunidades e desafios que tenho encontrado em minha vida.

Aos meus pais pela educação, amor e atenção que me foi dado.

À minha esposa Paula, pela paciência e apoio nesta etapa da minha vida. Que soube compreender minhas ausências durante a escrita desta dissertação.

Agradeço ao professor Mário Fiallos, pelo seu acompanhamento, orientação, dedicação e compromisso durante o desenvolvimento do trabalho. E também pela sua paciência devido aos meus atrasos às nossas reuniões.

Agradeço também a professora Rossana por sua co-orientação, apoio e oportunidade de trabalhar no CENAPAD-NE e de ingressar no Projeto Pauá. Não posso deixar de agradecer também a companhia da equipe: Janine, Felipe, Juracy e Sílvia.

Ao apoio e ajuda de minha amiga de mestrado Ana Flávia.

Agradeço ao Paulo Benício pelas lições de vida, exemplo de competência e profissionalismo e de pessoa. E também por sua amizade e companheirismo.

Aos amigos que conquistei durante o projeto Kanindé: Rodrigo, João Paulo, David Sena, Marcelo Iury e à equipe do CT/XML. E pela oportunidade de financiamento do mestrado pelo Instituto Atlântico.

Também agradeço o suporte do pessoal de Campina Grande: ao professor Walfredo, Érica, Ayla, Bozo, Nazareno e toda a equipe do LSD pelas dúvidas sanadas do OurGrid.

Agradeço, enfim, a todos os que, direta ou indiretamente, me auxiliaram nos trabalhos.

"Eu não me envergonho de corrigir meus erros e mudar as opiniões, porque não me envergonho de raciocinar e aprender."

(Alexandre Herculano)

RESUMO

MyGSI é uma proposta de arquitetura de segurança para ambientes de grades *peer-to-peer*. Utilizando mecanismos de autenticação, controle de acesso e delegação de direitos de acesso, MyGSI permite a troca de informações e o compartilhamento de recursos de forma segura através de três módulos: MyAuth, MyAC e MyDel. MyAuth é o módulo responsável pela autenticação utilizando a infra-estrutura de chaves públicas. MyAC é o módulo responsável pelo controle de acesso e permite o gerenciamento descentralizado de políticas de controle de acesso. MyDel é o módulo responsável pelo processo de delegação de direitos de acesso através de correntes de certificados. MyGSI foi desenvolvido na linguagem JAVA e integrado na grade OurGrid. O processo de integração de MyGSI com o OurGrid, alguns cenários de uso e os resultados desta integração também são apresentados nesta dissertação.

Palavras-chave: grade computacional, *peer-to-peer*, segurança, autenticação, controle de acesso.

ABSTRACT

MyGSI is a proposal of security architecture for peer-to-peer grid environments. MyGSI uses authentication mechanisms, access control and delegation of access rights. MyGSI allows the exchange of information in secure mode, and is composed of three modules: MyAuth, MyAC and MyDel. MyAuth uses public key infrastructure to deal with authentication. MyAC deals with access control, allowing a decentralized access control policies management. MyDel deals with the delegation of access rights implemented through certified chains. MyGSI was developed in JAVA and was integrated to OurGrid successfully. Some examples and results of this integration are also presented.

Keywords: grid computing, peer-to-peer, security, authentication, access control.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACL – *Access Control List*

API - *Application Program Interface*

BoT – *Bag of Tasks*

CA – *Certificate Authority*

CAS - *Community Authorization Service*

CRL - *Certificate Revocation Lists*

DAC - *Discretionary Access Control*

DC – Departamento de Computação

DES - *Data Encryption Standard*

DETI – Departamento de Engenharia de Teleinformática

DHT – *Distributed hash table*

DMTF – *Desktop Management Task Force*

DN - *Distinguished Name*

GAA – Generic Authorization and Access

GDF – *Grid Description File*

GGF - *Global Grid Forum*

GRAM - *Grid Resource Alocation Management*

GSI - *Grid Security Infrastructure*

GT2 - *Globus Toolkit* versão 2

GuM – *Grid Machine*

GuMP – *Grid Machine Provider*

HPF - *High Perfomance Fortran*

ICP – Infra-estruturas de Chaves Pública

IDL – *Interface Definition Language*

IETF – *Internet Engineering Task Force*

ISO - *International Orgazination for Standardization*

J2SE - *Java 2 Plataform Standard Edition*

JDF – *Job Description File*

KDC – *Key Distribution Center*

LSD – Laboratório de Sistemas Distribuídos

MAC - *Mandatory Access Control*

MPI - *Message Passing Interface*

MDS - *Monitoring and Discovery Service*

MyAC - *My Access Control*

MyAuth – *My Authorization*

MyDel - *My Delegation*

MyGSI – *My Grid Security Infrastructure*

OGSA - *Open Grid Services Achitecture*

P2P – *Peer-to-peer*

PDP – *Policy Decision Point*

PeGAC - *Peer-to-peer Grids Access Control*

PEP – *Policy Enforcement Point*

P2PSLF - *Peer-to-Peer Security Layer Framework*

PKI – *Public Key Infrastructure*

PRIMA - *Privilege Management and Authorization*

PVM - *Parallel Virtual Machine*

RBAC - *Role-Based Access Control*

RMI - *Remote Method Invocation*

RSA - *Rivest, Shamir, Adleman*

SDF – *Site Description File*

SSH – *Secure Shell*

SSL – Secure Sockets Layer

TLS – *Transport Layer Security*

UFC – Universidade Federal do Ceará

UFCG – Universidade Federal de Campina Grande.

URI – *Universal Resource Identification*

VO – *Virtual organization*

WSDL – *Web Services Definition Language*

XACML - *eXtensible Access Control Markup Language*

XML - eXtensible Markup Language

SUMÁRIO

Introdução

i. Considerações Iniciais

Uma grade computacional é caracterizada por um conjunto arbitrariamente grande de recursos heterogêneos distribuídos através de diversos domínios administrativos [FOSTER, 2003]. Recursos esses, formados por quaisquer dispositivos com algum poder computacional (estações de trabalhos, instrumentos científicos, entre outros). Na medida em que as grades são cada vez mais utilizadas como plataforma de execução de aplicações científicas e mesmo comerciais, a necessidade da construção de estruturas deste tipo aumenta.

Criar uma grade computacional, entretanto, não é uma tarefa trivial. Além de prover uma infra-estrutura complexa, é necessário resolver uma série de questões políticas surgidas com o paradigma da computação em grade. É necessário definir quem pode acessar quais recursos, e sob que condições. Estas definições tornam-se um problema também complexo em um ambiente descentralizado, arbitrariamente grande e formado de partes não-confiáveis.

As abordagens dos sistemas tradicionais, baseadas na definição estática de usuários por administradores não é suficiente para prover, em grades, fácil acesso a uma grande quantidade de recursos para os usuários. Novos mecanismos de estabelecimento de relações de confiança são necessários para lidar com o cenário da computação em grades. Neste contexto, a segurança se torna um aspecto crítico e um serviço indispensável para o funcionamento da grade.

A complexidade na implementação de uma infra-estrutura de segurança em ambientes de grade é devida à distribuição de processamento e ao ambiente heterogêneo da solução. Isto implica em novos requisitos de segurança, dentre os quais: *single sign on*, autenticação mútua, compartilhamento de credenciais e delegação a processos [FOSTER et al. 1998].

Em ambientes seguros, a disponibilidade de serviços de autenticação e autorização deve ser explorada. O objetivo da autenticação e autorização é o de prover um sistema computacional confiável, definindo as entidades participantes e as ações, métodos e operações que são permitidas.

Um dos esforços de construção de uma grade computacional a nível nacional é o Projeto Pauá ou Grade Pauá. Financiado pela HP-Brasil, a Grade Pauá é a implementação de uma grade geograficamente distribuída, utilizando, de forma transparente e não-dedicada, o tempo de máquinas instaladas em *clusters* de instituições que compõem o projeto, utilizando-se das tecnologias MyGrid / OurGrid desenvolvidas pela Universidade Federal de Campina Grande [ANDRADE et al. 2003; CIRNE et al. 2003].

OurGrid implementa uma grade *peer-to-peer* para execução de aplicações Bag-of-Tasks enquanto que o MyGrid escalona as aplicações do usuário, utilizando todos os recursos acessíveis por esse usuário no seu domínio administrativo [ANDRADE et al 2005].

Dentro do Projeto Pauá, existiu o subprojeto Kanindé cujo objetivo foi modelar e implementar protótipos de mecanismos de segurança para infra-estrutura em Grade Computacional, dentre os quais destacamos o modelo de autenticação, autorização e delegação a ser proposto.

ii. Objetivos da Dissertação

a. Definir, modelar e implantar mecanismos de autenticação, autorização e delegação em grades computacionais P2P, através de:
    - comparação entre os modelos de segurança existentes;
    - adaptação de aspectos funcionais de segurança para o modelo em grades computacionais.
b. Implementar um ambiente seguro que permita aos usuários da grade computacional:
    - possuir mecanismos válidos para autenticação;
    - estabelecer mecanismos de confiabilidade entre múltiplos domínios de processamento;
    - realizar transmissão segura de dados e informações para processamento;
    - permitir a delegação total ou parcial dos direitos de acesso.
c. Criar um ambiente de testes multi-institucional que permita a:
    - utilização de uma rede de computadores geograficamente distribuídas entre os participantes da grade;
    - verificação dos protocolos e mecanismos utilizados.

d. Divulgar a funcionalidade dos serviços de segurança propostos para a comunidade científica.

iii. Notação Utilizada

O texto utiliza uma notação padronizada, para facilitar a leitura, obedecendo as seguintes regras:

- termos em *itálico* indicam palavras ou expressões em língua estrangeira (normalmente inglesa). Algumas traduções estão feitas para facilitar a legibilidade do texto e obedecendo a utilização comum;
- termos em **negrito** são usados para apresentar alguma definição ou idéia principal. Isto é feito, normalmente, quando se julgar conveniente destacar um conceito específico;
- a utilização de notas de rodapé é feita sempre que aspectos particulares ou comentários são necessários.

iv. Organização do Trabalho

Este trabalho está organizado em cinco capítulos, descritos a seguir.

Capítulo 1 Conceitos Básicos

Neste capítulo introdutório são apresentados os conceitos fundamentais relacionados com sistemas *peer-to-peer*, ambientes de grades computacionais e a integração destas tecnologias nas grades *peer-to-peer*.

Capítulo 2 Segurança

No segundo capítulo são explorados os conceitos de segurança, tipos, ameaças de ataques e os requisitos de segurança para sistemas distribuídos e grades computacionais. Os principais mecanismos de segurança para o gerenciamento de recursos para grades *peer-to-peer* também são detalhados neste capítulo.

Capítulo 3 MyGSI - My Grid Security Infrastructure

Neste capítulo são ilustrados os principais aspectos da arquitetura de segurança que se propõe para ambientes de grades *peer-to-peer*.

O desenvolvimento de um protótipo e sua comparação com sistemas de segurança para grades também estão presentes neste capítulo.

Capítulo 4 Estudo de Caso - MyGSI e OurGrid

No quarto capítulo detalha-se o processo de integração de MyGSI com o OurGrid, descrevendo o processo de implantação dos mecanismos de segurança e a validação dos objetivos propostos pela arquitetura.

Capítulo 5 Conclusões e Trabalhos Futuros

Neste capítulo são descritos os benefícios obtidos ao utilizar MyGSI. Também são apresentados trabalhos futuros de melhorias na implementação de MyGSI.

Capítulo 1

Conceitos Básicos

Introdução

Neste primeiro capítulo é feita uma introdução geral aos conceitos envolvidos na computação *peer-to-peer* e na computação distribuída das grades computacionais, envolvendo aspectos históricos, características básicas até tópicos relacionados com aplicações que utilizam essas tecnologias. Também são apresentados detalhes sobre a integração das grades computacionais e a tecnologia *peer-to-peer*, destacando a importância do processo de obtenção de recursos das grades. O objetivo é caracterizar as grades *peer-to-peer*, conceituar as principais entidades de obtenção de recursos e sua participação no processo, para abordar nos próximos capítulos seus aspectos de segurança.

1.1. Sistemas *Peer-To-Peer*

1.1.1. Definição

A computação *peer-to-peer* (P2P) tem se difundido com o surgimento e a utilização massiva de sistemas de compartilhamento de arquivos P2P, como Kazaa, Gnutella, entre outros [KAZAA 2005; GNUTELLA 2005].

Diferente do modelo cliente servidor, a computação P2P independe de uma organização central ou hierárquica e seus integrantes possuem o mesmo poder computacional em servir e utilizar recursos e serviços, dispensando o uso de servidores dedicados. A centralização da informação e do poder de processamento em sistemas cliente-servidor afeta a escalabilidade e a vulnerabilidade dos sistemas.

Schollmeier define a rede P2P como uma rede em que os participantes compartilham uma parte de seus recursos de *hardware* (poder de processamento, capacidades de armazenamento e de conexão, entre outros) para prover o serviço e conteúdo da rede [SCHOLLMEIER et al. 2001]. Estes recursos são acessíveis por outros nós (*peers)*, sem passar por intermediários. Os participantes são provedores e consumidores de recursos da rede. No modelo cliente–servidor, o cliente não compartilha seus recursos, existindo somente uma entidade central que provê todo o conteúdo e serviços, representada pela presença do servidor, como mostra a Figura 1.

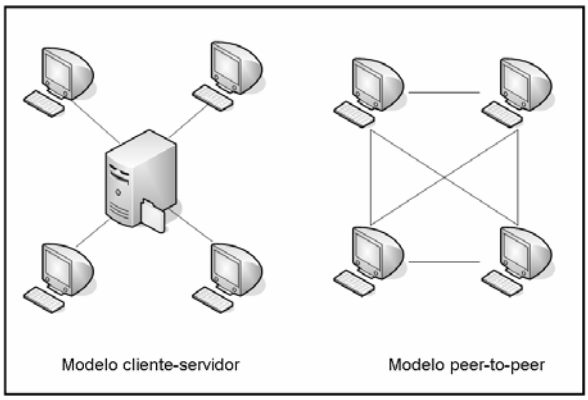

Figura 1. Sistema cliente-servidor e *peer-to-peer*.

1.1.2. Histórico

O conceito P2P não é recente, nas décadas de 60 e 70, quando a *internet* ainda era conhecida como ARPANET, os computadores atuavam como servidores e clientes ao mesmo tempo. Durante o decorrer dos anos esta característica desapareceu devida à especialização de funções, necessidade de robustez e alta disponibilidade, problemas de segurança e limitação de recursos computacionais ou capacidades dos enlaces de comunicação. Isto caracteriza o cenário que se tem hoje: máquinas mais robustas prestando serviços a vários clientes nas “bordas” da rede [CALLADO et al. 2004].

Durante o final da década de 70 e nos anos 80, houve uma concentração de pesquisas no uso de sistemas distribuídos em aplicações específicas. As idéias de autonomia, transparência, execução concorrente, tolerância à falhas e heterogeneidade de plataformas foram organizadas no modelo distribuído e colocadas em prática.

Nos anos 90 com o surgimento do Napster e de outras aplicações de compartilhamento de arquivos, as redes P2P tornaram-se populares [NAPSTER 2005].

1.1.3. Características

As redes P2P são formadas por um conjunto de nós que utilizam uma infra-estrutura de rede física. Estes nós podem possuir capacidades iguais ou não de fornecer e consumir recursos da rede.

Os nós são conectados de forma aleatória, não havendo restrição sobre o número de participantes da rede. A conexão de um nó se estabelece através de outro nó que já pertença à rede. Eles podem se conectar e sair da rede a qualquer momento sem prévio conhecimento dos demais membros. Além da conexão variável, o endereçamento dos nós e a taxa de transmissão entre eles também pode variar de acordo com a rede.

Redes P2P suportam um maior crescimento no número de usuários, de equipamentos conectados e também possibilitam a utilização da capacidade de processamento e de armazenamento das máquinas ociosas.

Na rede P2P a presença de um servidor central é opcional para a sua coordenação, isto reduz o custo ou investimento adicional em *hardware* de alto desempenho.

A Tabela 1 (adaptação de [GAO 2004]) enumera algumas diferenças entre o modelo P2P (descentralizado) e o modelo Cliente-Servidor (centralizado).

Características	Modelo Cliente-Servidor	Modelo P2P
Propriedade da informação	Servidor	Nó
Agrupamento do conteúdo	Por domínio	Por grupos de nós
Backup da informação	Replicação em outros servidores	Potencialmente, cada nó pode atuar como “espelho”
Ponto de contato primário	Servidor central	Um grupo de nós, um supernó
Comunicação entre os clientes	Coordenação centralizada pelo servidor central.	Coordenada pelos próprios nós
Escalabilidade	Limitada pelo servidor	Limitada pela rede
Ingresso de novo participante	Através do servidor	Através de qualquer nó
Onde adicionar conteúdo	No servidor	Em qualquer nó
Roteamento de tráfego	Via roteador	Via nó roteador
Gargalo	Servidor	Rede

Tabela 1 : Modelo cliente-servidor x P2P.

1.1.4. Arquitetura

As redes P2P podem ser classificadas em dois grupos: **puras** ou **híbridas** [SCHOLLMEIER et al. 2001].

- **Puras**

Neste tipo não existe entidade central, e desta forma qualquer entidade da rede pode ser removida sem afetar o serviço oferecido.

- **Híbridas**

Estes sistemas caracterizam-se pela presença de entidades centrais de indexação de nós, não fornecendo outro tipo de serviço ou conteúdo. O conteúdo é negociado e trocado diretamente entre os nós.

Além desta classificação, outras três classificações são bastante utilizadas na literatura para as redes P2P. A primeira destas classificações divide a tecnologia P2P em descentralizada e semicentralizada [P2P ARCHITECT PROJECT 2003].

- **Descentralizada**

Neste tipo de sistema não há um nó central, e todos os nós são autônomos e responsáveis pela troca e controle (gerenciamento) de recursos. Os nós se comunicam de maneira direta ou através de vizinhos comuns, isto caracteriza uma melhor escala de informações de controle, porém aumenta o tráfego de rede.

- **Semicentralizada**

Nestes sistemas há diferença de relevância entre os nós, havendo um ou mais nós centrais, também denominados de supernós, para informações de controle e, possivelmente, para tráfego de dados. A indisponibilidade ou falha de um supernó afeta apenas os nós inferiores conectados a ele. O supernó é um nó com maior capacidade de processamento e serve como ponto de encontro dos outros nós. Já os nós inferiores, podem ter um nível maior ou menor de autonomia, mas são equivalentes entre si.

Diferente da primeira, a segunda classificação tem como critério o tipo de busca utilizada, dividindo a tecnologia P2P em busca centralizada, busca por inundação e busca por tabela *hash* distribuída. [TOWSLEY 2003]:

- **Busca Centralizada**

Rede P2P com um ponto central (possivelmente espelhado por outros pontos, dando a impressão de serem vários) de busca e nós que consultam o ponto central para trocar informações diretamente entre si.

- **Busca por Inundação**

Rede com nós totalmente independentes. A busca normalmente é limitada à vizinhança mais próxima do nó que a iniciou. Desta forma a busca é escalável, mas não é completa.

- **Busca por Tabela *Hash* Distribuída (DHT)**

Rede em que os nós têm autonomia e utilizam uma tabela *hash* para separar o espaço de busca entre eles.

A terceira classificação mistura características das duas primeiras classificações, agupando-as em: centralizada, descentralizada e estruturada, e descentralizada e não-estruturada [LV et al. 2002].

- **Centralizada**

Rede que mantém um índice central com informações atualizadas (similar à busca centralizada da classificação anterior). Sistemas de compartilhamento de arquivos como Napster e de troca de mensagens utilizam esta arquitetura.

- **Descentralizada e Estruturada**

Rede que não possui um servidor centralizado de diretório de informações, mas que tem uma estruturação significativa entre os nós. A topologia da rede é controlada e os recursos são posicionados em locais que posteriormente tornam fácil a sua localização. Este tipo de arquitetura, em geral, utiliza a busca baseada em DHT. Esta arquitetura é utilizada pelos sistemas Chord [STOICA et al. 2003], Pastry [ROWSTRON 2001] e Tapestry [ZHAO et al. 2004].

- **Descentralizada e Não Estruturada**

Rede que não possui servidor centralizado, nem controle preciso sobre a topologia e localização/busca de recursos. Compreende os dois tipos de buscas (descentralizado e

semicentralizado) da primeira classificação e a busca por inundação da segunda. A rede Gnutella utiliza esta arquitetura.

1.1.5. Aplicações

Sistemas P2P estão presentes em várias classes de aplicações, dentre as quais se podem destacar: troca de mensagem instantânea, compartilhamento de arquivos e computação distribuída.

A. Mensagem Instantânea

Nas aplicações P2P de mensagem instantânea, as mensagens são entregues diretamente ao usuário, sem a necessidade de servidores para armazenamento como acontecem com as mensagens de e-mail.

Os contatos do usuário são armazenados em listas utilizadas para identificar se o usuário está ativo ou inativo. Devida sua popularização, existem vários aplicativos de mensagem instantânea, entre eles o MSN Messenger [MSN MESSENGER 2005], AIM [AOL 2005] e o Yahoo! Messenger [YAHOO 2005].

B. Compartilhamento de Arquivos

Sistemas de compartilhamento de arquivos são aplicações que possuem um grande destaque utilizando tecnologia P2P. De maneira geral, elas oferecem:

- **áreas de armazenamento potencialmente ilimitadas** para o usuário;
- **alta disponibilidade** do conteúdo armazenado, com a adoção de políticas de replicação múltipla de conteúdo, ou seja, um conteúdo pode ser armazenado em mais de um nó na rede;
- **anonimato** na troca de informação entre os usuários da rede P2P, sem a identificação explícita dos autores dos recursos, dos nomes dos nós envolvidos na comunicação e do usuário receptor de informações;
- **mecanismos de gerenciamento eficientes** de localização e recuperação de conteúdos armazenados na rede.

O consumo de largura de banda de rede, segurança e a capacidade de pesquisa são os principais desafios no compartilhamento de arquivos [MILOJICIC et al. 2005]. Pode-se citar como sistemas de compartilhamento de arquivos: o Napster, utilizado para compartilhar

arquivos de músicas; o Gnutella, considerado a primeira solução P2P pura; o Freenet [CLARKE et al. 2005], rede descentralizada que adota comunicação cifrada de seus usuários, preservando o anonimato na publicação e coleta de informações na *internet* e o Kazaa [KAZAA 2005] que utiliza supernós para melhorar o desempenho da rede.

C. Computação Distribuída

A aplicação da tecnologia P2P na computação distribuída caracteriza-se no aproveitamento de recursos computacionais ociosos.

Condor [LITZKOW 1988] e MOSIX [BARAK 1999] são exemplos de sistemas que aproveitam o alto desempenho computacional que pode ser obtido de um conjunto de máquinas. As grades computacionais também aproveitam da flexibilidade e escalabilidade dos sistemas P2P para distribuir o processamento em recursos ociosos.

Além das classes de aplicações mencionadas, sistemas com necessidade de escalabilidade, descentralização, distribuição de recursos e relacionamento transiente de nós podem utilizar a tecnologia P2P.

A Tabela 2, a Tabela 3, a Tabela 4 e a Tabela 5, utilizam os critérios da seção 1.1.4 para classificar alguns dos sistemas mencionados.

Sistema P2P	Puro	Híbrido
Mensagem instantânea (msn, aim, yahoo)		X
Napster		X
Kazaa		X
Gnutella	X	
Freenet	X	
OurGrid		X
Chord, Pastry, Tapestry	X	

Tabela 2. Sistemas P2P classificados como puro ou híbrido.

Sistema P2P	Descentralizado	Semicentralizado
Mensagem instantânea (msn, aim, yahoo)		X
Napster		X
Kazaa		X
Gnutella	X	
Freenet	X	
OurGrid		X
Chord, Pastry, Tapestry	X	

Tabela 3. Sistemas P2P classificados em descentralizado ou semicentralizado.

Sistema P2P	Busca centralizada	Busca por inundação	Busca por DHT
Mensagem instantânea (msn, aim, yahoo)	**X**		
Napster	X		
Kazaa	X		
Gnutella		X	
Freenet		X	
OurGrid	X		
Chord, Pastry, Tapestry			X

Tabela 4. Sistemas P2P classificados de acordo com o tipo de busca.

Sistema P2P	Descentralizado e não estruturado	Descentralizado e estruturado	Centralizado
Mensagem instantânea (msn, aim, yahoo)			X
Napster			X
Kazaa			X
Gnutella	X		
Freenet	X		
OurGrid			X
Chord, Pastry, Tapestry		X	

Tabela 5. Sistemas P2P classificados em descentralizado e não estruturado, descentralizado e estruturado e centralizado.

1.2. Grades Computacionais

1.2.1. Definição

Grade computacional é uma infra-estrutura computacional que tem por objetivo prover uma plataforma virtual para computação e gerenciamento de dados, de forma semelhante à *internet* integrando recursos para a formação de uma plataforma virtual para publicação de informação. Já existem vários esforços, que tornam real a utilização das grades como infra-estruturas, capazes de prover organizações dinâmicas de recursos para suportar a execução de aplicações distribuídas, de larga-escala e que façam o uso intensivo destes recursos [BERMAN 2003].

Para a computação em larga-escala, as grades computacionais devem atender aos requisitos de trabalhar com recursos heterogêneos, dinâmicos e distribuídos, permitindo a utilização de ciclos e armazenamento potencialmente infinitos, como também o acesso a variados instrumentos científicos, sem se preocupar com a localização geográfica de tais dispositivos. Independente da quantidade de recursos a serem utilizados pelo usuário, a grade é responsável pela organização, coordenação e utilização destes recursos de maneira uniforme [BERMAN 2003]. Denomina-se de **organização virtual** (VO) o conjunto de indivíduos e/ou instituições definidas pelas regras de compartilhamento e coordenação de recursos numa grade [FOSTER, 2003].

Pode-se mencionar como características das grades computacionais [ROURE et al. 2003]: heterogeneidade, escalabilidade e adaptabilidade.

- **Heterogeneidade**

Uma grade envolve multiplicidade de recursos que são heterogêneos por natureza e podem atravessar diferentes domínios.

- **Escalabilidade**

Crescimento de poucos a milhões de recursos. Conseqüentemente, aplicações que requerem um número elevado de recursos geograficamente distribuídos devem ser tolerantes à latência e explorarem os recursos acessados localmente. Além disso, o aumento da escala também envolve atravessar uma grande quantidade de limites organizacionais, enfatizando a heterogeneidade e a necessidade de requisitos de segurança como autenticação e confiabilidade.

- **Adaptabilidade**

Com a diversidade de recursos, aumenta-se a probabilidade de alguns recursos falharem. Gerenciadores de recursos ou as próprias aplicações devem tratar este comportamento dinamicamente, de maneira que extraiam o máximo de desempenho dos recursos e serviços disponíveis.

1.2.2. Evolução

Durante a década de 80, a pesquisa em computação paralela preocupou-se no desenvolvimento de algoritmos, programas e arquiteturas para computadores paralelos. A

limitação de processamento das máquinas incentivou os estudos na distribuição de processamento, além dos limites da máquina, para o processamento de aplicações maiores.

Bibliotecas como Parallel Virtual Machine (PVM), Message Passing Interface (MPI), OpenMP e linguagens como High Perfomance Fortran (HPF) são desenvolvidas para dar suporte de comunicação às aplicações escaláveis [GROPP et al. 2003]. As aplicações paralelas concentraram-se no desenvolvimento de poderosos mecanismos de gerenciamento de comunicação entre processadores (década de 90).

Paralelamente ao descrito nos parágrafos anteriores, áreas multidisciplinares de pesquisa necessitam de um modelo de colaboração geograficamente distribuído. Estas características de distribuição e coordenação são fundamentais para o surgimento das grades computacionais.

Em 1995 a grade I-WAY é apresentada no *Supercomputing* e considerada como a primeira grade computacional moderna, provendo uma infra-estrutura de acesso, aplicando segurança e coordenando recursos através de dezessete sítios distribuídos geograficamente e conectados [BERMAN 2003].

Com a demonstração da grade I-WAY, outros projetos e aplicações em grades computacionais surgem, dentre os quais se destacam: Globus [GLOBUS PROJECT 2005] e Legion [GRIMSHAW 1997] que provêem uma infra-estrutura de grade computacional básica, o projeto Condor [CONDOR PROJECT 2005] com escalonamento de alta transferência de dados, enquanto o AppLes [BERMAN et al 1996], Mars [GEHRING 1996] e Prophet [WEISSMAN 1999] trabalham o escalonamento dedicado. NetSolve [NETSOLVE 2005] e Ninf [NINF 2005] são projetos que utilizam a computação remota via o modelo cliente-servidor, enquanto que o projeto OurGrid [ANDRADE et al 2005] implementa grades P2P.

Desde o final da década de 90, a comunidade científica reúne-se no Global Grid Fórum (GGF) com o objetivo de padronizar as grades computacionais. Baseados na integração do Globus com a tecnologia de serviços *web* (*web services*), tentam desenvolver o Open Grid Services Achitecture (OGSA) para a definição de serviços centrais como gerenciamento de sistemas e automação, gerenciamento de distribuição de carga e desempenho, segurança, gerenciamento de serviços/disponibilidade, gerenciamento lógico de recursos, serviços de *cluster*, gerenciamento de conectividade e gerenciamento de recursos físicos.

1.2.3. Gerações

A evolução das grades pode então ser mapeada em três gerações: primeira, segunda e terceira geração [ROURE et al. 2003].

- **Primeira Geração** (década 90)

Caracterizada por projetos que tem como objetivo conectar sítios de supercomputação através de redes de alta velocidade, este processo também é conhecido como metacomputação. Destacam-se nesta geração os projetos FAFNER [FAFNER 2005] e I-WAY [FOSTER et al. 1996].

- **Segunda Geração**

Caracterizada por *middlewares*[1] que fornecem alguns requisitos de grade: serviços de comunicação, de informação, de nomes, de armazenamento, segurança, tolerância à falhas, gerenciamento de recursos e escalonamento, interfaces com o usuário. Além disso, são sistemas que utilizam a tecnologia de objetos distribuídos, corretores de recursos e escalonadores, sistemas completamente integrados e sistemas P2P. Sistemas como Globus [FOSTER 1997], OurGrid [ANDRADE et al 2005], Legion [GRIMSHAW 1997], escalonadores como Condor [CONDOR PROJECT 2005], Nimrod-G [ABRAMSON et al. 2001] e sistemas integrados como Cactus [ALLEN et al. 2001], DataGrid [DATA GRID PROJECT 2005] e Unicore [ALMOND 1999] pertencem a essa geração.

- **Terceira Geração**

Soluções que adotam o modelo orientado a serviços e a adoção de metadados[2]. Determinam novas maneiras de descrição da grade e de tornar a sua computação autônoma [IBM AUTONOMIC COMPUTING 2005]. Como arquiteturas orientadas a serviços utilizam padrões *web services* (OGSA [FOSTER et al. 2003]).

---

[1] Middleware: camada de software localizada entre o sistema operacional e a aplicação, provendo uma variedade de serviços requerida pela aplicação para funcionar corretamente. Em uma grade, o middleware é usado para esconder a natureza heterogênea e prover uma visão homogênea do ambiente ao usuário e aplicações através de um conjunto padronizado de interfaces para uma variedade de serviços.

[2] Metadado é uma abstração do dado, capaz, por exemplo, de indicar se uma determinada base de dados existe, quais são os atributos de uma tabela e suas características, tais como: tamanho e/ou formato.

1.2.4. Arquitetura

As grades possuem uma arquitetura dividida em camadas, conforme mostra a Figura 2: de infra-estrutura, recursos e conectividade, coletiva e de aplicações do usuário [FOSTER, 2003].

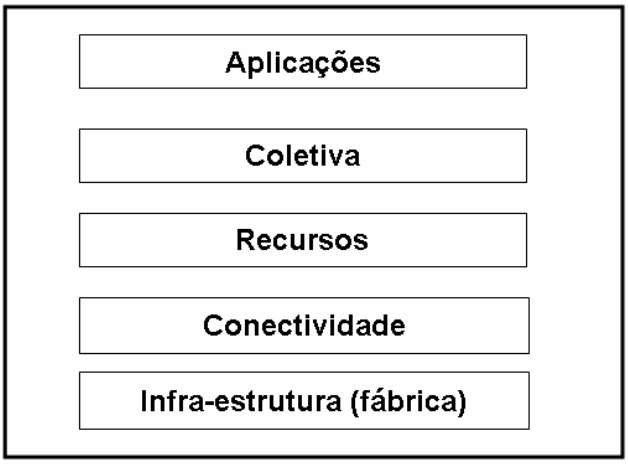

Figura 2. Arquitetura em camadas das grades [FOSTER, 2003].

Na camada de infra-estrutura ou fábrica, estão localizados os dispositivos físicos ou recursos que os usuários da grade querem compartilhar e acessar, incluindo computadores, sistemas de armazenamento, redes e vários tipos de sensores. Um recurso pode ser uma entidade lógica, tal como um sistema de arquivos distribuído ou um *cluster*.

Acima da camada de infra-estrutura, tem-se a camada de recursos e conectividade. Os protocolos e mecanismos de comunicação e autenticação, necessários para as transações específicas da grade, encontram-se na camada de conectividade. Os protocolos de comunicação permitem a troca de dados entre os recursos. São sobre os serviços de comunicação que os protocolos de autenticação operam, provendo mecanismos de segurança criptográfica para a verificação das identidades dos usuários e recursos.

A camada de recursos contém protocolos que exploram a comunicação e autenticação, permitindo a inicialização, monitoramento, e controle das operações dos recursos compartilhados. A execução de uma mesma aplicação em diferentes plataformas computacionais depende desta camada.

A camada coletiva contém protocolos, serviços e APIs[3] (*Application Program Interface)* que implementam interações entre coleções de recursos. Os componentes da camada coletiva podem implementar uma vasta variedade de tarefas sem precisar de novos componentes da camada de recursos. Como exemplo de serviços coletivos, tem-se os serviços de diretório e serviços de descoberta e alocação de recursos; serviços de monitoramento e diagnóstico; serviços de replicação de dados e os serviços de política e controle de usuários para mapear quem na comunidade é permitido acessar os recursos.

No topo da arquitetura, encontram-se as aplicações dos usuários, que utilizam componentes de outras camadas. Como tarefas utilizadas por aplicações de usuários, cita-se:

- obtenção de credenciais de autenticação (protocolos da camada de conectividade);
- consultas para determinar a disponibilidade de recursos (serviços coletivos);
- submissão de tarefas para determinados recursos (protocolo de recursos);
- monitoramento da execução das tarefas e detecção de falhas (protocolo de recursos).

1.3. Grades Peer-To-Peer

1.3.1. Introdução

Não existe uma definição para grades P2P, o conceito de grades P2P surge da integração dos sistemas de grades computacionais e da tecnologia P2P. As grades são exemplificadas como infra-estruturas que permitem o acesso a computadores de alto desempenho e aos seus conjuntos de dados. Por sua vez, a tecnologia P2P permite a formação de comunidades *"ad hoc[4]"* composta por dispositivos que funcionam tanto como clientes, como servidores de arquivos. A utilização destes dois tipos de sistemas resulta nos sistemas de grade P2P [FOX et al. 2003].

---

[3] API: É a interface entre um componente ou sistema de suporte. Uma API é formada pelo conjunto de protótipos de funções, dados, tipos, regras de uso e especificações que um componente servidor torna disponível para os seus programas clientes.

[4] Ad -Hoc: Comunicação sem a necessidade de uma gestão de rede ou roteador.

1.3.2. Características

Na configuração das grades P2P, existe o chamado núcleo da rede formado por servidores estáveis e robustos. O núcleo da rede é caracterizado por uma organização estruturada que possui algum grau de hierarquia em sua topologia para dar suporte aos serviços de grade. Nas “bordas” da rede, têm-se características P2P suportando as interações dinâmicas locais dos clientes e gerenciando a estrutura transiente dos recursos [FOX et al. 2003].

A Figura 3 ilustra uma grade P2P, tem-se grupos de nós gerenciados localmente (representados pelos recursos) em um sistema suportado por supernós.

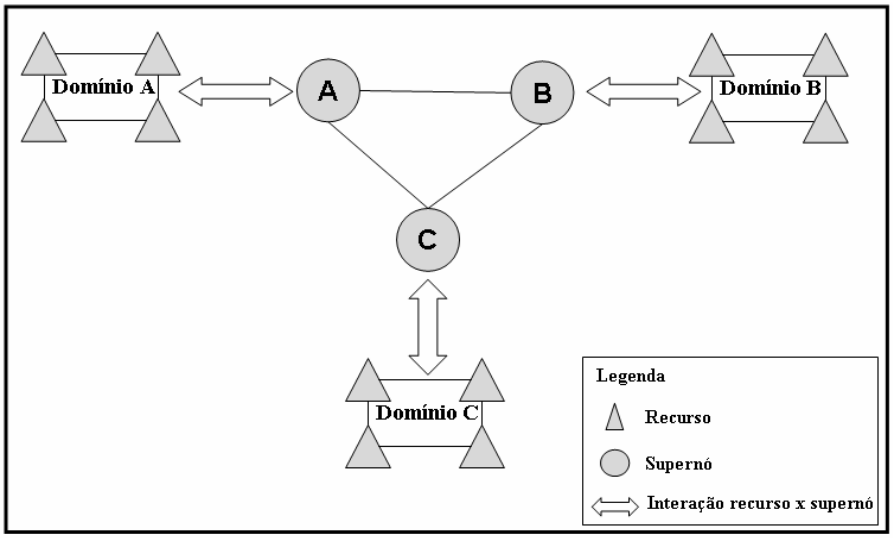

Figura 3. Grade P2P.

Os nós da grade são componentes menos robustos, não-confiáveis, sem um controle total e sujeitos a instabilidade da rede, de *software* e de *hardware*.

Por outro lado, os supernós devem ser confiáveis, executando em ambientes controlados e seus *softwares* devem ser configurados para garantir operações confiáveis.

Os dados transientes da grade podem ser armazenados nos recursos, mas os repositórios de informações permanentes devem se localizar nos supernós. Em uma grade P2P os recursos transientes atuam como veículos que processam e apresentam a informação para o usuário, sendo toda a ação e gerenciamento global tomada pelos servidores.

1.3.3. Exemplos de Grades P2P

As grades baseadas em serviços *web* (grades de serviço), utilizando a arquitetura OGSA, e o sistema OurGrid são exemplos de grades P2P [KUNSZT 2003; FOSTER 2004; ANDRADE et al. 2003].

Nas grades de serviço os recursos são virtualizados - encapsulados como serviços e disseminados através de instâncias do serviço - e todas as suas entidades se comunicam através de mensagens, formando um sistema distribuído [XU 2004].

Os objetos distribuídos são definidos em uma linguagem de definição de interfaces (*IDL – Interface Definition Language*) baseada em XML (*eXtensible Markup Language*), denominada Linguagem de Definição de Serviços Web (*WSDL – Web Services Definition Language*), enquanto que os recursos são mapeados através de um identificador universal (*URI – Universal Resource Identification*).

Nas “bordas” da rede temos grupos de nós gerenciados localmente e integrados com os serviços da grade localizados como pontos centrais da rede, conforme mostra a Figura 4.

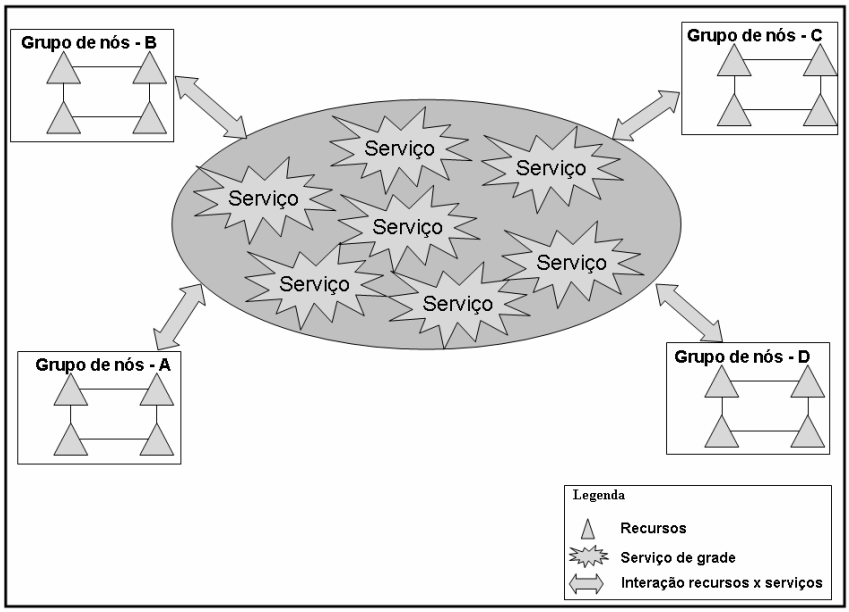

Figura 4. Grade de serviços [FOX et al. 2003].

Por outro lado, a grade OurGrid utiliza provedores de recursos no centro da arquitetura da grade, responsáveis pela negociação de recursos e submissão de tarefas em domínios administrativos distintos ou no próprio domínio local.

Ilustrada na Figura 5 e semelhante à grade baseada em serviços, a abordagem P2P da grade OurGrid é aplicada nos domínios locais, sendo a conexão transiente dos dispositivos e a execução das tarefas controladas através de um *middleware* denominado MyGrid [CIRNE et al. 2003].

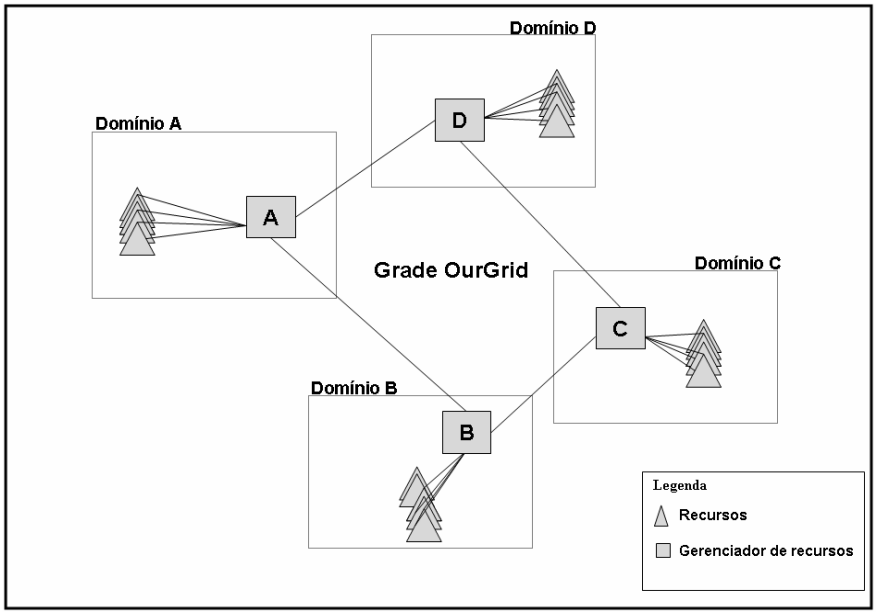

Figura 5. Arquitetura OurGrid [CIRNE et al. 2003].

1.3.4. Obtenção de Recursos em Grades P2P

Antes de explorar o processo de obtenção de recursos em grades P2P, deve-se definir e detalhar um pouco das principais entidades envolvidas neste processo: os **recursos**, o **escalonador** e o **gerenciador de recursos**.

A. Recursos

Tradicionalmente, o termo recurso tem sido interpretado denotando uma entidade física, tal como computador, rede, ou sistema de armazenamento. Nos ambientes de grades computacionais, utiliza-se o termo em um sentido muito mais genérico, para denotar qualquer capacidade que pode ser compartilhada e explorada no ambiente. Esta definição é aplicável à uma arquitetura orientada a serviço, dentro da qual ambos os recursos tradicionais e serviços virtualizados podem diferenciar na função que provêem para os usuários, mas se assemelham na forma que entregam suas funções através da rede [CZAJKOWSKI 2004].

Os recursos computacionais de armazenamento e os serviços incorporados na grade são denominados recursos da grade. Estes são caracterizados pela heterogeneidade e grande variedade de configurações e capacidades [JOHNSTON 2003].

B. Escalonador

Um dos principais objetivos da computação distribuída é permitir que uma comunidade de usuários execute suas tarefas em um conjunto de recursos compartilhados. Em um ambiente distribuído, o número de tarefas que devem ser executadas freqüentemente é superior ao número de recursos disponíveis. O escalonador é quem decide como alocar as tarefas aos recursos [THAIN 2003].

O escalonador é a entidade responsável pela distribuição de tarefas aos recursos computacionais que satisfazem os requisitos da tarefa submetida. Além disso, é também responsável pela inicialização dos serviços e seu gerenciamento nos recursos computacionais remotos [JOHNSTON 2003].

Os escalonadores decidem quais recursos são alocados para quais tarefas. Esta associação é feita através de políticas de decisão do escalonador. A política pode ser explicitamente especificada através de bases de regras externas, de interfaces programadas ou implicitamente implementadas através de algoritmos [KRAUTER 2002].

Como exemplos de escalonadores de grade, destacam-se os sistemas AppLeS [BERMAN et al 1996], Condor-G [FREY et al. 2001] e o Nimrod-G [ABRAMSON et al. 1995].

C. Gerenciador de Recursos

A habilidade de descobrir, alocar e negociar o uso dos recursos acessíveis, via rede, faz parte dos requisitos de uma grade. Embora existam muitas formas de obtenção de recursos, o termo gerenciamento de recursos é utilizado para descrever todos os aspectos do processo: localização de um recurso, disposição para uso, utilização e monitoramento de seu estado [CZAJKOWSKI 2004].

Gerenciamento de recursos em sistemas computacionais tradicionais diferencia-se bastante do gerenciamento em grades. Nos sistemas tradicionais, seus gerenciadores de recursos são locais, possuem um controle total do recurso e, desta forma, podem implementar os mecanismos e políticas necessárias para o uso efetivo do recurso de maneira isolada. O que

distingue o gerenciamento de recursos em ambientes de grade, do gerenciamento de sistemas locais, é o fato de que os recursos gerenciados se expandem por domínios administrativos distintos. Esta distribuição pode apresentar problemas diante da heterogeneidade à medida que os recursos são configurados e administrados.

O gerenciador de recursos controla os estados dos recursos e sua disponibilidade, preocupando-se com a conclusão das operações nos recursos e implementando vários tipos de interfaces devida a heterogeneidade dos recursos [CZAJKOWSKI 2004]. É através dele que as tarefas do escalonador são transmitidas aos recursos [CHIEN 2003]. O problema da heterogeneidade dos recursos pode ser superado com a definição de protocolos padrões de gerenciamento [CZAJKOWSKI et al. 1998; CZAJKOWSKI et al. 2001] e de mecanismos para expressar recursos e requisitos de tarefas [RAMAN 1998].

Do ponto de vista do usuário, o gerenciamento e escalonamento de recursos devem ser transparentes; sua interação com eles deve ser restrita através de mecanismos de submissão de tarefas da sua aplicação.

No sistema Globus, o gerenciamento de recursos é realizado através do protocolo Grid Resource Alocation Management (GRAM) e do Serviço de Monitoramento e Descoberta (MDS - Monitoring and Discovery Service) [FOSTER 2004a].

1.3.5. Processo de Obtenção de Recursos

O processo de obtenção de recursos é descrito utilizando as entidades consideradas na seção anterior.

A Figura 6 ilustra a seqüência de ações para a obtenção de recursos:

1. a submissão das tarefas do sujeito autenticado é analisada pelo escalonador que determina se sua requisição pode ser potencialmente satisfeita com os recursos locais;
2. o escalonador solicita ao gerenciador de recursos a disponibilidade atual dos recursos;
3. o gerenciador de recursos retorna ao escalonador os recursos efetivamente disponíveis que satisfaçam os requisitos de processamento;
4. o escalonador envia o resultado da requisição para o sujeito autenticado.

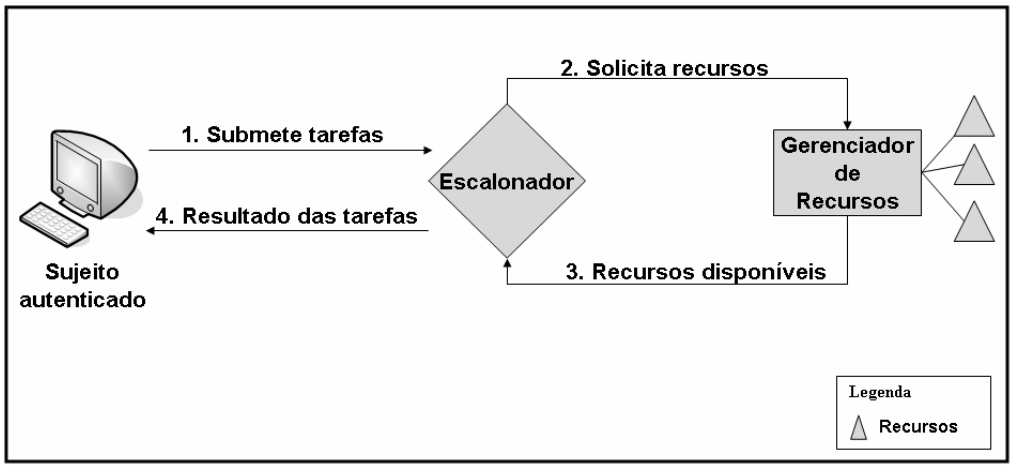

Figura 6. Obtenção de recursos em grades.

No próximo capítulo são discutidos os requisitos de segurança necessários para a utilização de forma segura da grade. Como função básica da grade computacional, o processo de obtenção de recursos necessita de mecanismos de autenticação das entidades e de controle de acesso aos recursos para evitar ataques à segurança da grade.

Capítulo 2

Segurança

Introdução

O conceito de segurança, tipos e ameaças de ataques, e de requisitos de segurança para sistemas distribuídos e grades computacionais são explorados neste capítulo.

Os principais mecanismos de segurança são detalhados, explorando a sua importância para o gerenciamento seguro de recursos para grades P2P. Os conceitos aqui apresentados são importantes para a compreensão da arquitetura de segurança proposta neste trabalho.

2.1. Segurança em Grades Computacionais

2.1.1. Conceitos

Antes de se abordar os requisitos de segurança para grades computacionais, deve-se definir o que é um ataque à segurança, mecanismos de segurança e serviço de segurança [STALLINGS 2000].

- **Ataque à segurança**

Qualquer ação que comprometa a segurança da informação. Classificam-se os ataques que modificam as informações ou dados como ataques ativos. Quando não há a modificação de dados, eles são classificados como ataques passivos [STALLINGS 2000].

- **Mecanismo de segurança**

Qualquer mecanismo utilizado para a detecção, prevenção ou recuperação de danos causados por ataques à segurança.

- **Serviço de segurança**

Qualquer serviço que garanta a segurança de sistemas de processamento de dados e informações que trafegam na rede. O objetivo de serviços de segurança é a contenção de ataques à segurança. A implantação destes serviços pode ocorrer a partir do desenvolvimento de um ou mais mecanismo de segurança.

2.1.2. Tipos de Ataques à Segurança

As grades computacionais, por se tratarem de um ambiente de processamento e comunicação de dados, estão sujeitas a alguns tipos de ataque, dentre os quais se destacam [SOARES 1995]: personificação, escuta, modificação ou alteração de dados e/ou código, repúdio, acesso não autorizado e ataques por repetição.

- **Personificação**

Uma entidade maliciosa faz-se passar por outra confiável para extrair informações que tenha como destino a entidade original. Como a entidade maliciosa, em caso de um ataque bem sucedido, tem a posse das informações enviadas, outros ataques, como os descritos a seguir, podem suceder à personificação.

- **Escuta**

No cenário clássico de escuta, a entidade maliciosa intercepta o canal de comunicação estabelecido entre duas partes. Neste tipo de ataque, a entidade maliciosa tem acesso à toda informação não cifrada, contida na computação enviada como instruções, dados e resultados. Desta forma, o atacante pode obter algoritmos proprietários ou qualquer outra informação contida na tarefa a ser processada.

- **Modificação** ou **alteração de dados e/ou código**

Uma requisição de processamento pode passar por vários dispositivos, dentre os quais alguns maliciosos. Assim, mecanismos que permitam a verificação da integridade do código e de dados armazenados devem ser providos aos dispositivos confiáveis. Soluções presentes em linguagens de programação não são suficientes, pois, o dispositivo malicioso pode estar executando uma máquina virtual modificada, por exemplo. É importante verificar a

integridade da informação, quando ela retorna à entidade de origem. Embora se tenha confiança em suas tarefas, estas podem ter sido alteradas por alguma máquina maliciosa ao longo do itinerário percorrido, com o intuito de comprometer a segurança da máquina de origem.

- **Repúdio**

Quando uma máquina participante de uma transação ou comunicação nega ter realizado as ações que foram efetuadas tem-se um repúdio. Isto pode ocorrer devida à ação de uma entidade maliciosa ou acidentalmente, por perda de uma mensagem. De qualquer forma, é importante que a infra-estrutura do sistema tenha mecanismos para poder auxiliar na resolução de tais situações.

- **Acesso não autorizado**

Ocorre quando não existe controle de acesso aos recursos. Por exemplo, um dispositivo malicioso pode alterar os dados, ou mesmo alocar recursos não-autorizados do servidor.

- **Ataques por repetição**

Neste ataque, uma terceira parte maliciosa intercepta uma mensagem, realiza uma cópia e depois clona ou a retransmite. A interceptação acontece “escutando-se” o canal de comunicação ou por meio de um servidor malicioso, quando este recebe uma mensagem.

2.1.3. Requisitos de Segurança

Segurança em ambientes de grades é uma característica complexa, requerendo que diversos recursos administrados de maneira autônoma interajam de uma forma que não impacte a utilização dos recursos, nem introduzam falhas em sistemas individuais ou no ambiente como um todo. Por isto, uma infra-estrutura de segurança é chave para o sucesso ou falha de um ambiente de grade [ROURE et al. 2003].

Qualquer sistema distribuído envolve quatro aspectos de segurança: confidencialidade, integridade, autenticação e responsabilização. Além destes quatro requisitos de segurança, tem-se ainda: disponibilidade, controle de acesso, não-repúdio, anonimato e os requisitos de segurança adicionais para as grades [TANENBAUM 2002; SEGURANÇA 2005].

A. Confidencialidade

Confidencialidade consiste em proteger a informação contra leitura e/ou cópia por alguém que não tenha sido explicitamente autorizado pelo proprietário daquela informação. Deve-se cuidar não apenas da proteção da informação como um todo, mas também de partes da informação que podem ser utilizadas para interferir sobre o todo. No caso da transmissão em rede, isto significa que os dados, enquanto em trânsito, não serão vistos, alterados, ou extraídos por pessoas não autorizadas ou capturados por dispositivos ilícitos.

A confidencialidade é obtida evitando-se a escuta, ou se isto não for possível, evitando-se a inteligibilidade dos dados durante o processo de transmissão.

Por se tratar de uma rede de meios físicos compartilhados, as grades computacionais são vulneráveis à interceptação de informações que trafegam na rede, quando não possuem mecanismos de confidencialidade. Para a captura destas informações é preciso usar um dispositivo físico ou um programa. Os programas de captura proporcionam uma interface com um dispositivo de *hardware* que é executado no modo promíscuo (*sniffer*), ou seja, copiando todos os pacotes que chegam até ele independentemente do endereço de destino contido no pacote.

Se um *sniffer* for instalado em alguma parte da rota entre dois *hosts* de uma rede, senhas e informações confidenciais podem ser capturadas, causando transtornos e prejuízos.

B. Integridade

A integridade consiste em proteger a informação contra modificação sem a permissão explícita do proprietário daquela informação. A modificação inclui ações como escrita, alteração de conteúdo, remoção e criação de informações. Deve-se considerar a proteção de informação nas suas mais variadas formas, como por exemplo, armazenada em discos ou fitas de *backup*.

C. Autenticação

O controle de autenticidade está associado com a identificação correta de um usuário ou computador. O serviço de autenticação em um sistema deve assegurar ao receptor que a mensagem é realmente procedente da origem informada em seu conteúdo. Normalmente, isso é desenvolvido a partir de um mecanismo de senhas ou de assinatura digital. A verificação de autenticidade é necessária após todo processo de identificação, seja de um usuário para um

sistema, de um sistema para o usuário ou de um sistema para outro sistema. Ela é a medida de proteção de um serviço/informação contra a personificação por intrusos.

D. Responsabilização

Cada processo, usuário ou agente em um dado nó deve ser responsabilizado pelas ações que venham a realizar na segurança. Como exemplos destas ações estão as modificações na configuração de acesso a um arquivo ou mudanças administrativas nos mecanismos de segurança de um determinado recurso. Para que haja responsabilização, cada entidade no sistema deve ser unicamente identificada e autenticada.

Sem responsabilização, mecanismos de proteção baseados em detecção a posteriori são completamente ineficazes: uma determinada máquina poderia, por exemplo, modificar uma informação e não ser punida pelos seus atos.

E. Disponibilidade

Existem ataques de negação de serviços em que o acesso a um sistema/aplicação é interrompido ou impedido, deixando de estar disponível; ou uma aplicação, cujo tempo de execução é crítico, é atrasada ou abortada.

Disponibilidade consiste na proteção dos serviços prestados pelo sistema de forma que eles não sejam degradados ou se tornem indisponíveis sem autorização, assegurando o acesso aos dados sempre que deles precisar. Isto pode ser chamado também de continuidade de serviços.

Um sistema indisponível, quando um usuário ou sistema autorizado necessita dele, pode resultar em perdas tão graves quanto as causadas pela remoção das informações daquele sistema. Atacar a disponibilidade significa realizar ações que visem a negação do acesso a um serviço ou informação.

F. Controle de Acesso

No contexto de segurança de rede, o controle de acesso é a habilidade de limitar ou controlar o acesso aos computadores hospedeiros ou aplicações, através dos enlaces de comunicação e do controle de acesso físico. Para tal, cada entidade que precisa obter acesso ao recurso, deve primeiramente ser identificada, ou autenticada de forma que os direitos e permissões de acesso sejam atribuídos ao usuário.

G. Não-repúdio

Não-repúdio é um requisito de segurança que impede que entidades neguem as ações por elas realizadas em uma transação ou comunicação. Para isto, a infra-estrutura de grades deve coletar e registrar informações que comprovem a ocorrência de transações e eventos importantes.

H. Anonimato

Em alguns casos é desejável que a identidade da pessoa responsável pela informação permaneça no anonimato. Há muitas situações, porém, em que os participantes podem relutar em negociar com uma entidade anônima. Entretanto, a privacidade deve ser cuidadosamente balanceada em relação à necessidade dos servidores de manterem os serviços responsáveis pelas ações realizadas.

2.1.4. Outros Requisitos

Em ambientes de grade, um recurso pode assumir o papel de servidor no momento que está recebendo tarefas de outros recursos, e como cliente, na hora de enviar tarefas para outros recursos. Com esta nova visão, novos requisitos de segurança são necessários para o ambiente de grade: autenticação única, mapeamento dos mecanismos de segurança locais, delegação, autorização e políticas baseadas na comunidade, relações de confiança [ROURE et al. 2003; BUTLER et al. 2000].

- **Autenticação única (single sign-on)**

O usuário deve ser capaz de se autenticar uma única vez e então atribuir para a computação o direito de representá-lo (*user proxy*), geralmente por um determinado período, evitando desta forma múltiplos processos de autenticação e pontos de vulnerabilidade.

- **Mapeamento dos mecanismos de segurança locais**

Sítios distintos podem usar diferentes soluções de segurança localmente, portanto, a infra-estrutura de grade precisa mapear estas soluções locais para que cada operação seja processada com seus devidos privilégios.

- **Delegação**

Mecanismo pelo qual o usuário ou processo dá direito de outro para atuar em seu nome. Com a delegação, pode-se evitar que o usuário da grade tenha que entrar com sua

informações confidenciais (login, senha) várias vezes, deixando suas informações vulneráveis à ataques [FOSTER et al. 1998; GASSER 1990; HOWELL 2000].

- **Autorização e políticas baseadas na comunidade**

Além da autorização baseada na identidade do usuário e do recurso que deseja acessar, tradicional em sistemas cliente-servidor, as políticas e regras de autorização nas grades computacionais necessitam serem expressas através de outros critérios, tais como informações do grupo dos quais os membros da grade fazem partes.

- **Relações de confiança**

A segurança em sistemas de grade é fortemente relacionada à noção de confiabilidade. Um sistema confiável é aquele que permite ao usuário, de maneira justificada, confiar na execução de seus serviços.

Confiar significa acreditar em alguma qualidade ou atributo de uma entidade, ou na verdade de uma premissa [JOHNSTON 2003]. Por exemplo, para um usuário utilizar recursos de múltiplos provedores juntos, o sistema de segurança não deve exigir interação ou cooperação entre administradores para configurar o ambiente. Se um usuário tem o direito para utilizar os sítios A e B, ele deve ser capaz de fazer uso destes sítios juntos, sem necessitar que os administradores de segurança dos sítios interajam, caracterizando uma relação de confiança entre os sítios A, B e o usuário. Estas relações de confiança, entre os sítios e organizações, são estabelecidas durante o processo de autenticação, permitindo a colaboração em larga-escala.

Através de mecanismos de autenticação, comunicação segura e do gerenciamento de confiança, é possível que grupos remotos de colaboração interajam entre si de forma confiável, servindo como a base de políticas de compartilhamento de recursos [JOHNSTON 2003].

É difícil estabelecer confiança em organizações virtuais grandes e heterogêneas, que envolvem membros de múltiplas instituições, devido à inexistência de um modelo de confiança compartilhada. Tomemos como exemplo os sistemas administrativamente similares (dentro de uma organização): geralmente o modelo de confiança existente é informal, podendo ser estendido para a autenticação e autorização da grade, o que não acontece com a colaboração entre organizações distintas.

As soluções de segurança para grades também devem prover suporte flexível para proteção da comunicação (controle sobre o grau de proteção, proteção de dados para protocolos não confiáveis, suporte para protocolos de transporte confiáveis diferente do TCP) e permitir o controle sobre decisões de autorização, incluindo a habilidade para restringir os direitos de delegação em várias formas [FOSTER, 2003].

2.2. Mecanismos de Segurança

A segurança em sistemas distribuídos pode ser dividida em duas partes. A primeira parte consiste na comunicação entre as entidades do sistema (processos e usuários, por exemplo), sendo o principal mecanismo de segurança a presença de um canal seguro de comunicação. A segunda parte consiste na autorização, cujo objetivo é garantir que um processo ou usuário possa realizar somente as operações permitidas pelos seus direitos de acesso [TANENBAUM 2002].

Canais seguros de comunicação e mecanismos de controle de acesso trabalham com conceitos que envolvem o gerenciamento de chaves criptográficas e a certificação de entidades. Desta forma, os mecanismos de segurança para sistemas distribuídos e grades computacionais estão baseados no uso de três tipos de técnicas: criptografia, autenticação e controle de acesso [TANENBAUM 2002]. A delegação também é um mecanismo que merece destaque no contexto de segurança de grades computacionais, sendo importante para o crescimento e utilização de uma grade computacional de forma segura.

2.2.1. Criptografia

O uso de mecanismos de criptografia em segurança de sistemas distribuídos é fundamental. A criptografia consiste em tomar uma mensagem m (texto plano[5]) que deve ser transmitida entre as entidades S e R, cifrando-a em m' (mensagem cifrada) que é uma mensagem não inteligível. Quando R recebe a mensagem, é feito o processo de decifragem da mensagem m' para m. São utilizadas chaves como parâmetros dos processos de cifragem e decifragem. A Figura 7 demonstra o método exposto.

[5] Texto plano (*plaintext)*: texto no seu formato original, que não foi cifrado e pode ser lido com facilidade.

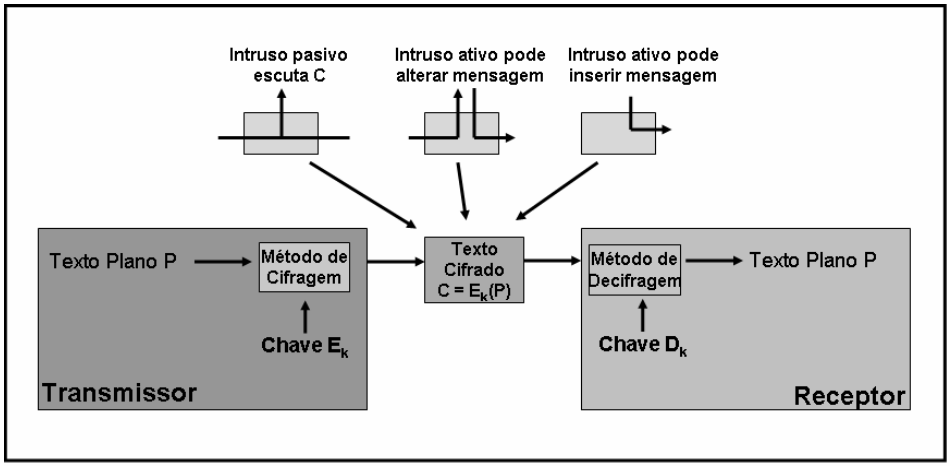

Figura 7. Exemplo de criptografia.

Os métodos de criptografia podem utilizar chaves assimétricas (chave pública e privada), chaves simétricas (chave compartilhada) e funções *hash* (ANEXO A).

A seguinte notação é utilizada nesta dissertação:

- **$C = E_k(P)$** denotando que o texto cifrado C é obtido pela cifragem do texto plano P, utilizando a chave k;
- **$P = D_k(C)$** é usado para expressar a decifragem do texto cifrado C, utilizando a chave k, resultando o texto plano P.

Com o uso da criptografia, podem-se evitar alguns tipos de ataques como interceptação, modificação e fabricação. Na interceptação, se a mensagem for interceptada, ela somente será legível se o intruso possuir a chave utilizada na cifragem ou decifragem. Para que haja a modificação a mensagem deve antes ser decifrada e depois cifrada novamente. E para que ocorra a fabricação, toda mensagem deve ser codificada pelas chaves utilizadas.

2.2.2. Autenticação

No processo de autenticação a entidade se identifica e valida suas informações antes de ter acesso ao sistema. O objetivo da autenticação é verificar se uma entidade é realmente quem afirma ser [TANENBAUM 2002].

Procedimentos de autenticação agregam características de integridade e confidencialidade, além de controlar a não-repudiação e responsabilização das entidades através do uso de assinaturas digitais.

Na comunicação cliente-servidor, a autenticação baseia-se na maioria das vezes na autenticação somente do servidor. Em ambientes P2P, esta situação é mais complexa. A distinção entre cliente e servidor desaparece, uma vez que um recurso em um dado momento (quando recebe uma requisição) pode atuar como servidor e como cliente em outro (quando emite uma requisição para outro recurso). Nesse caso, é necessária a autenticação mútua de entidades [FOSTER 2003a].

Entre os mecanismos de autenticação, destacam-se a utilização da infra-estrutura de chaves públicas (ICP), técnicas de assinaturas digitais e do protocolo SSL/TLS. Além disso, a implementação de autenticação pode se valer tanto de um acordo entre pares, quanto de uma autoridade certificadora (CA – *Certificate Authority*) ou fazer uso de centros de distribuição de chaves (KDC – *Key Distribution Center*).

A. Infra-Estrutura de Chaves Públicas (ICP)

ICP é uma infra-estrutura de segurança baseada em chaves públicas. Tem como objetivo facilitar o desenvolvimento de sistemas computacionais distribuídos, escaláveis e seguros. O objeto central é o certificado digital que é utilizado durante o processo de autenticação.

Os certificados digitais utilizados possuem o formato X.509 versão 3. É importante destacar que são usados através da camada de transporte seguro (TLS – *Transport Layer Security*) e da camada de *sockets* seguros (SSL – *Secure Sockets Layer*) [DIERKS 2005; FREIER 2005].

O formato X.509 (padrão ISO - *International Orgazination for Standardization*) define uma estrutura que provê serviços de autenticação. É uma coleção de um conjunto padrão de campos contendo informações sobre um usuário ou dispositivo e sua chave pública correspondente, conforme Figura 8 [STALLINGS 2000a].

O padrão X.509 define qual informação está presente no certificado, e descreve o formato de codificação destas informações. Estes certificados podem ser criados pelo próprio usuário, mas geralmente são emitidos por CAs que possuem uma maior confiabilidade. Os principais campos utilizados no processo de autenticação são o identificador de nome único (**DN** - *Distinguished Name*), a chave pública do usuário e o período de validade do certificado.

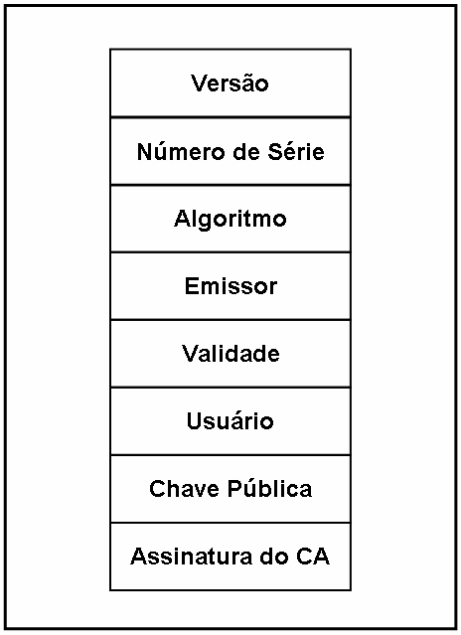

Figura 8. Exemplo de certificado.

B. Assinaturas Digitais

Assinaturas digitais correspondem ao processo pelo qual o usuário garante ao sistema ser o responsável pelas mensagens, informações, transações ou ações por ele introduzidas. Este mecanismo assegura que modificações no estado do sistema são notadas, bem como a não-repudiação de quem as efetuou.

Dentre os principais algoritmos de criptografia pública, destacam-se o RSA (Rivest, Shamir, Adleman), cuja forma tradicional de assinatura é mostrada na Figura 9.

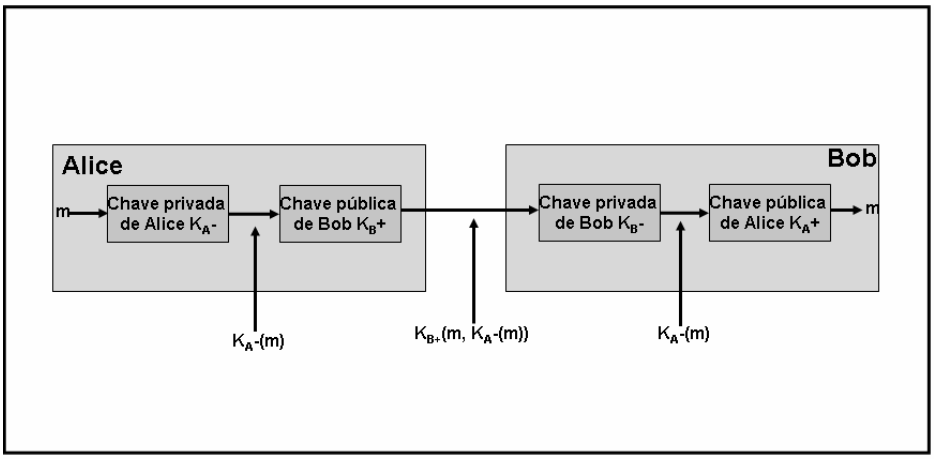

Figura 9. Criptografia de chave pública (RSA).

C. Protocolo SSL/TLS

O protocolo SSL/TLS é responsável pela confidencialidade, integridade e não-repudiação na comunicação entre duas partes, através de conexões TCP/IP. Localiza-se entre a camada de transporte TCP e o protocolo da camada de aplicação [FREIER 2005].

O protocolo SSL/TLS é flexível, característica importante para ambientes heterogêneos como as grades P2P. Através de uma das suas duas camadas (*SSL Handshake*) é possível escolher a utilização da autenticação mútua entre servidores e clientes, negociar o algoritmo de cifragem e a chave criptográfica simétrica, antes que a aplicação transmita ou receba um conjunto de dados. A outra camada do protocolo, *SSL Record*, é a camada responsável pela cifragem, decifragem, compactação e descompactação de dados.

A autenticação entre cliente e servidor no protocolo SSL é feita através das trocas de certificados X.509. Os nós só conseguem trocar informações se cada um possuir o certificado do outro. Desta forma, um nó malicioso que deseja se passar por um nó confiável não obterá sucesso na comunicação, uma vez que não possui a chave privada do nó confiável para iniciar a transação.

D. KDC e CAs

Centros de distribuição de chaves determinam as comunicações permitidas entre as entidades. Quando uma permissão é garantida para o estabelecimento de conexão, o KDC emite uma chave de sessão para aquela conexão [STALLINGS 2000b].

Autoridades certificadoras são utilizadas na administração da base de certificados, no estabelecimento de procedimentos técnicos de troca de chaves e na criação de uma estrutura para o gerenciamento de chaves. Autoridades certificadoras não apenas emitem certificados, mas também devem gerenciá-los, determinando a validade de cada certificado e as condições de renovação e possivelmente gerando e armazenando uma lista de certificados já emitidos e que não estão mais válidos, ou seja, estão revogados [TANENBAUM 2002].

KDC e CA são entidades intermediárias na comunicação entre partes, também conhecidas como terceiras partes (*third parties*). Uma terceira parte T pode ser classificada em três categorias, utilizando como critério sua localização relativa na comunicação entre as partes A e B: *in-line*, *on-line e off-line*. Quando T serve em tempo real à comunicação entre A e B, ele é classificado como *in-line*. No modo *on-line*, T é envolvido em tempo real durante cada instância do protocolo, mas A e B se comunicam diretamente, independente de T.

Finalmente, quando T não está envolvido no protocolo em tempo real, mas prepara a informação a priori, que posteriormente é disponibilizada a A, B ou a ambos, sendo a informação utilizada em tempo de execução, ele é classificado como ***off-line***.

A utilização de KDCs e CAs em sistemas distribuídos demonstram algumas dificuldades. Havendo indisponibilidade dos KDCs, comunicações seguras não podem ser estabelecidas, a menos que se use alguma técnica alternativa como o algoritmo Diffie-Hellman. E os CAs, se estiverem comprometidos, impossibilitam a verificação da validade de uma chave pública. Neste caso, a solução é a replicação dos servidores. No entanto, isto aumenta a vulnerabilidade e o custo do sistema [STALLINGS 2000].

2.2.3. Controle de Acesso

O controle de acesso aos recursos é um requisito chave nos sistemas de segurança para grades. A grande variedade e quantidade de recursos, usuários e serviços transientes em grades P2P tornam o controle de acesso um desafio considerável.

A coordenação do compartilhamento de recursos em organizações dinâmicas, virtuais e multi-institucionais é um dos requisitos funcionais da infra-estrutura de uma grade computacional. O aspecto dinâmico implica no número e tipos de participantes, de atividades, duração e escala da interação e a quantidade de recursos que está sendo compartilhada [FOSTER, 2003].

Gerenciar recursos é um desafio, principalmente quando um número variável de participantes, dos mais variados graus de confiança, desejam compartilhá-los para a execução de suas tarefas. Além do mais, o compartilhamento é algo a mais do que a simples troca de documentos, pode envolver o acesso direto a sistemas remotos com computadores, bancos de dados, sensores, e outros tipos de recursos.

Nas grades P2P, as relações de compartilhamento podem ser combinadas e cada recurso utilizado, por uma tarefa, pode pertencer a uma organização diferente, abrangendo quaisquer subconjuntos de participantes (mesmo potencial de fornecer os serviços). Estes relacionamentos não necessariamente envolvem um conjunto de indivíduos reconhecidos explicitamente por cada sítio, eles podem ser definidos implicitamente pelas políticas que orientam o acesso aos recursos. Por exemplo, uma organização pode permitir o acesso a seus recursos para qualquer um que possa demonstrar que é membro de uma entidade reconhecida. A habilidade para delegar autoridade, em meios controlados, é importante em tais situações,

como fazem os mecanismos para coordenar operações através de múltiplos recursos. Além disso, ambientes de compartilhamento devem permitir que cada proprietário de recurso que deseja disponibilizá-lo, restrinja quando, onde, e quais operações podem ser realizadas sobre eles. A implementação de tais restrições requer mecanismos para expressar políticas, para o estabelecimento da identidade de um consumidor de recurso (autenticação), e para determinar se uma operação é consistente com o relacionamento de compartilhamento aplicável (autorização).

O controle de acesso está relacionado à verificação dos direitos de acesso e a autorização está relacionada à garantia de direitos de acesso. O modelo de controle de acesso é apresentado na Figura 10. Nela, pode-se observar a presença de um monitor de referência. O monitor de referência registra qual sujeito pode fazer o quê, e decide se no objeto é permitida a execução de uma operação específica. O monitor é chamado cada vez que o objeto é invocado.

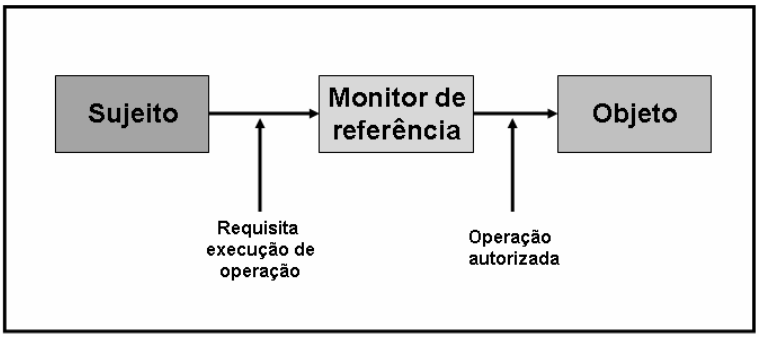

Figura 10. Monitor de referência.

Os mecanismos de controle de acesso devem obedecer às políticas de segurança do sítio ou do sistema (a grade computacional como um todo), que descrevem precisamente quais ações as entidades do sistema são autorizadas a tomar e quais delas são proibidas.

A. Focos de Controle de Acesso

O foco de controle de acesso trata da forma pela qual são asseguradas as permissões. Na Figura 11 são mostrados três casos [TANENBAUM 2002]:

- na primeira abordagem a segurança é concentrada diretamente na proteção dos dados ou objetos que são associados com a aplicação;

- a segunda abordagem concentra a segurança na especificação exata de quais operações devem ser invocadas e por quem, quando certos dados ou recursos são acessados;
- a terceira abordagem aplica a segurança diretamente nos usuários, tomando medidas pelas quais somente pessoas específicas têm acesso para as aplicações, independente das operações a serem executadas. O controle está na definição das regras que os usuários devem seguir.

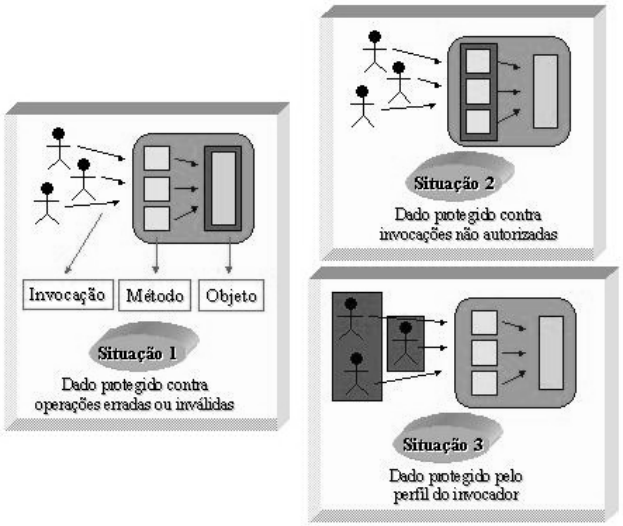

Figura 11. Situações do foco de controle.

B. Técnicas de Autorização

Têm-se como principais técnicas para modelar regras de autorização a objetos, a utilização de matrizes de controle de acesso e domínios de proteção [TANENBAUM 2002].

- **Matriz de controle de acesso**

Modelam os direitos de acesso dos sujeitos sobre os objetos (recursos). O sujeito é representado por uma linha e o objeto por uma coluna. Considerando que uma grade computacional possa conter milhares de objetos e sujeitos, muitas entradas desta matriz podem estar vazias. Desta forma, uma alternativa é o uso de Listas de Controle de Acesso (ACL – *Access Control List*) e de capacidades.

As ACLs relacionam quais permissões cada sujeito possui em um objeto particular, sendo armazenada junto com cada objeto. Já as capacidades indicam para cada sujeito as permissões que eles possuem para cada objeto.

- **Domínios de proteção**

A utilização de capacidades e ACLs podem se transformar em estruturas extensas. Desta forma, utilizam-se domínios de proteção, que consistem em conjunto de pares (objeto, direitos de acesso). Cada par indica quais operações são permitidas ao objeto. Quando o servidor é requisitado para executar uma operação de um sujeito, ele verifica o monitor que por sua vez consulta o domínio de proteção ao qual o objeto pertence. É possível a criação de vários domínios de maneira hierárquica.

C. Modelos de Controle de Acesso

Modelos de controle de acesso (ou modelos de autorização de acesso) definem características primitivas de um determinado conjunto de regras de autorização a serem utilizadas. Estas características influenciam os limites da semântica de autorização que pode ser expressa no modelo e conseqüentemente a sua implementação. Os modelos fundamentais de controle de acesso são DAC (*Discretionary Access Control*), MAC (*Mandatory Access Control*) e RBAC (*Role-Based Access Control*) [FERRAIOLO 2005].

- **DAC - *Discretionary Access Control***

Este modelo é baseado na idéia de que usuários individuais são “donos” de objetos e portanto tem controle total em quem deve ter permissões para acessar o objeto. Um usuário transforma-se em dono do objeto ao criá-lo [LAMPSON 1971].

Não existem políticas de controle de acesso em DAC, já que o modelo de operação “padrão” do modelo é suficiente para suas necessidades. Este modelo é utilizado em vários sistemas operacionais (exemplo: UNIX, série Windows NT, 2000 e XP).

O modelo DAC possui uma fraqueza inerente: o fato de que informação pode ser copiada de um objeto para outro, de modo que o acesso a uma cópia é possível, mesmo que o dono do objeto original não tenha provido acesso ao original [SANDHU 1996].

- **MAC - *Mandatory Access Control***

Este modelo prevê que usuários individuais não têm escolha em relação a que permissões de acesso eles possuem ou a que objetos podem acessar [SANDHU 1993; SANDHU et al. 1996]. Os usuários individuais não são considerados donos dos objetos, portanto não podem definir suas permissões – isso é realizado pelos administradores do sistema.

O modelo MAC é conhecido por sua utilização em políticas de acesso multinível, em que se deseja controlar o fluxo de informações em um sistema. Em geral, este modelo garante que a informação só flua em um determinado sentido, por exemplo, de níveis mais baixos de confidencialidade para níveis maiores de confidencialidade.

Políticas de acesso mandatórias são bastante utilizadas em sistemas militares [ANDERSON 2001].

- **RBAC – *Role-Based Access Control***

RBAC é projetado para gerenciar, de maneira centralizada, os privilégios ao prover uma camada de abstração, conhecida como *role* (papel), mais alinhado à estrutura da organização. A idéia central de RBAC é associar permissões a papéis, por sua vez os usuários são associados aos seus papéis corretos na organização. Isso simplifica bastante a administração e gerenciamento de permissões de acesso em grandes organizações. Papéis são criados para as várias funções de negócio da organização, e os usuários são associados a estes papéis de acordo com suas responsabilidades e qualificações. Usuários podem ser facilmente re-associados de um papel para outro. Papéis podem receber novas permissões de acesso, à medida, que novas aplicações ou funcionalidades são adicionadas ao sistema e permissões podem ser revogadas sempre que necessário [SANDHU et al. 1996; SANDHU 2000].

O conjunto de usuários e permissões associadas a um papel é transitório. O papel é mais estável porque as atividades e funções da organização, em geral, mudam menos do que o conjunto de usuários ou de permissões. Isso torna a administração de um sistema RBAC muito mais fácil e escalável.

A política de controle de acesso é incorporada em vários componentes de RBAC, como as relações papel – permissão, usuário – papel e papel – papel. Estes componentes, em conjunto, determinam se um usuário particular tem permissão de acesso aos dados fornecidos pelo sistema.

2.2.4. Delegação

A delegação ocorre quando uma entidade transfere seus direitos de acesso a uma outra. Dentre os mecanismos de delegação utilizados em grades, tem-se o uso de *proxies* [GLOBUS PROJECT 2005] e de cadeias de certificados [LORCH et al. 2003].

Um *proxy* é um *token* que permite ao seu proprietário operar com os direitos e privilégios semelhantes ou mais restritos que do seu criador. Enquanto que uma corrente de certificados é composta de vários certificados, sendo cada certificado assinado digitalmente por seu predecessor, demonstrando relações de confiança entre os membros da corrente. Por exemplo, se um determinado sítio desconhece o certificado de Alice, mas confia no certificado de Bob, e Alice se identifica com a corrente de certificados ilustrada na Figura 12, a relação de confiança entre Alice e este sítio pode ser estabelecida, uma vez que a corrente de certificados possibilita a verificação de seus membros ou "elos" e desta forma, reconhecer o certificado de Bob.

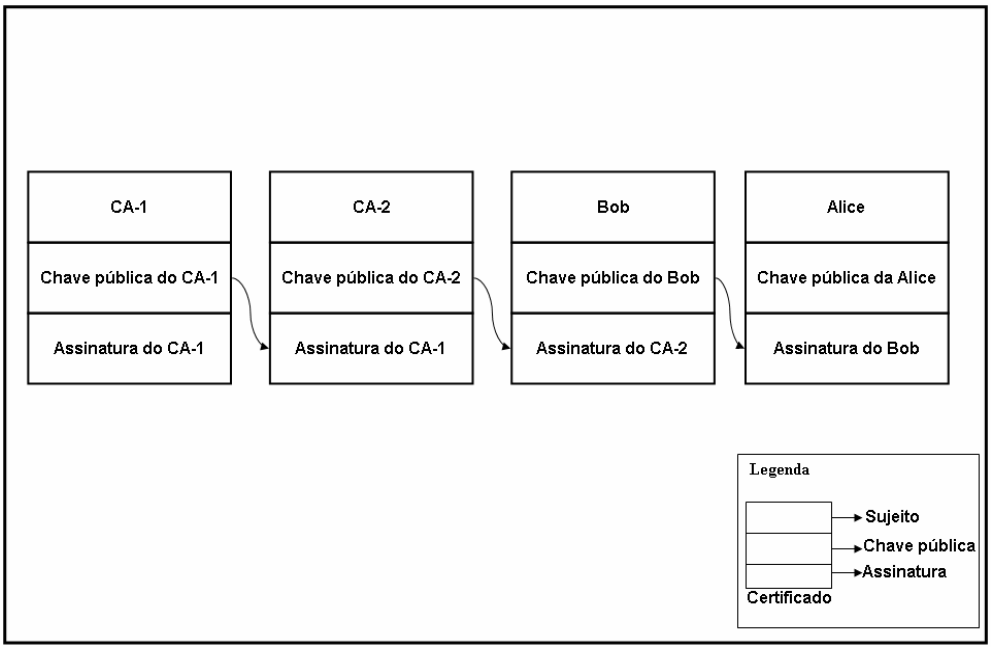

Figura 12. Corrente de certificados.

No próximo capítulo, a arquitetura MyGSI é apresentada usando as técnicas de autenticação, controle de acesso e delegação, mencionadas neste capítulo.

Capítulo 3

MyGSI – *My Grid Security Infrastructure*

Introdução

Neste capítulo, é apresentada a arquitetura e implementação de MyGSI- uma proposta de infra-estrutura para segurança em grades P2P.

Os detalhes de cada um dos três módulos (MyAuth, MyAC e MyDel) que compõem a arquitetura MyGSI são explicados na seção 3.1. A implementação do protótipo de MyGSI é apresentada na seção 3.2.

Na seção 3.3 são mostrados outros trabalhos relacionados com a segurança em grades P2P. Conclui-se o capítulo comparando MyGSI com as soluções relacionadas.

3.1. Arquitetura MyGSI

MyGSI (*My Grid Security Infrastructure*) surge como uma proposta de arquitetura de segurança para grades P2P para autenticação, controle de acesso, e delegação de direitos de acesso. Incorporando esses mecanismos de segurança às grades P2P, requisitos de segurança como disponibilidade, integridade e confidencialidade são satisfeitos, além de tornar o ambiente menos vulnerável aos ataques de interceptação, modificação e fabricação.

A arquitetura MyGSI é dividida em três módulos: MyAuth (*My Authorization),* MyAC (*My Access Control*) e MyDel (*My Delegation).*

3.1.1. MyAuth

MyAuth é o módulo de MyGSI responsável pela autenticação das entidades da grade e criação dos canais seguros de comunicação. O mecanismo de autenticação utilizado por MyAuth baseia-se na infra-estrutura de chave pública (ver Seção 2.2.2), trabalhando com certificados X.509 e canais SSL/TLS.

3.1.2. MyAC

MyAC é o módulo de MyGSI responsável pela autorização e controle de acesso aos recursos da grade, respeitando o aspecto dinâmico, multi-institucional e a diversidade dos domínios pertencentes. Possibilitando cada sítio possuir regras de acesso distintas e independentes, de acordo com suas características administrativas.

MyAC incorpora a arquitetura de redes baseada em políticas do padrão IETF (*Internet Engineering Task Force*) /DMTF (*Desktop Management Task Force*) para os ambientes de grade P2P [VERMA 2001].

A arquitetura definido pelo IETF/DMTF consiste de quatro elementos básicos, ilustrados na Figura 13: ferramenta de gerenciamento de políticas, repositório de políticas, ponto de decisão de políticas e ponto de aplicação de políticas.

- **Ferramenta de gerenciamento de políticas**

Utilizado para a geração e inserção das políticas no repositório. As políticas de entrada são escritas em alto-nível, ficando a cargo do ponto de aplicação de políticas (PEP – *Policy Enforcement Point*) a sua execução.

- **Repositório de políticas**

É o local de armazenamento das políticas geradas pela ferramenta de gerenciamento de políticas.

- **Ponto de Decisão de Políticas (PDP)**

Responsável pela interpretação das políticas armazenadas no repositório de políticas e pelo envio da decisão de acesso para o PEP. Através do PDP são selecionadas as políticas aplicáveis para as requisições de acesso enviadas pelo PEP. O retorno da decisão de acesso (permite ou nega o acesso) na sintaxe entendida pelo PEP também é função do PDP.

- **Ponto de Aplicação de Políticas (PEP**)

Responsável pela aplicação das políticas. Realiza o controle de acesso, fazendo requisições de decisão de acesso e aplicando o resultado dessas decisões.

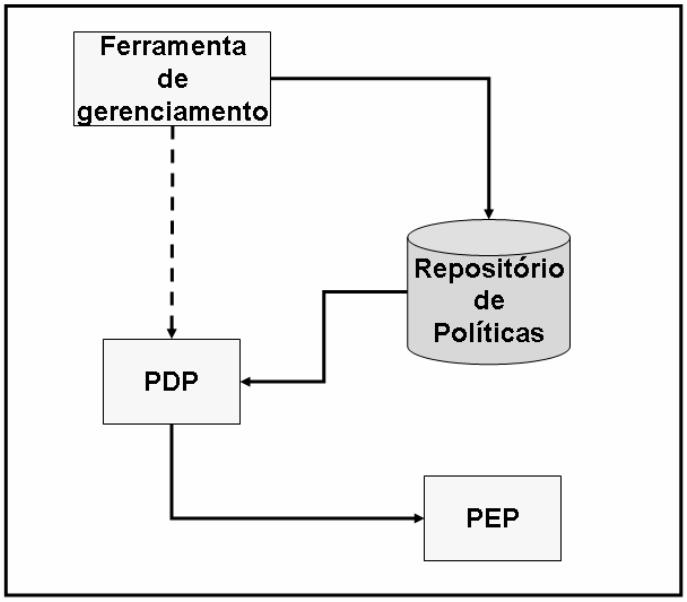

Figura 13. Arquitetura baseada em políticas (IETF/DMTF) [VERMA 2001].

A Figura 14 mostra MyAC com a disposição dos elementos da arquitetura baseada em políticas, em ambientes de grades P2P. O processo de acesso aos recursos sofre algumas modificações, realizando os seguintes passos:

1. o sujeito autenticado envia uma requisição de recursos ao escalonador e seus atributos ao PEP;
2. o escalonador solicita recursos ao gerenciador de recursos;
3. o gerenciador de recursos retorna os recursos disponíveis para o PEP, ao invés de retornar diretamente ao escalonador. Através do PEP a aplicação verifica os direitos de acesso;
4. o PEP produz uma requisição para o PDP;
5. o PDP retorna os recursos efetivamente permitidos (se existir) para o PEP;
6. o escalonador obtém os recursos autorizados (se houver algum), distribui as tarefas e envia o resultado ao sujeito (passo 7).

A arquitetura ilustrada é projetada para que haja pouca modificação no sistema ou *middleware* da grade. O PEP é o único componente incluso na infra-estrutura, deixando o PDP e o repositório de políticas sem alteração.

A administração e o controle das políticas se tornam mais fáceis com esta separação do repositório. Os administradores de domínios ou os próprios donos de recursos podem manipular suas políticas sem influenciar no funcionamento da grade.

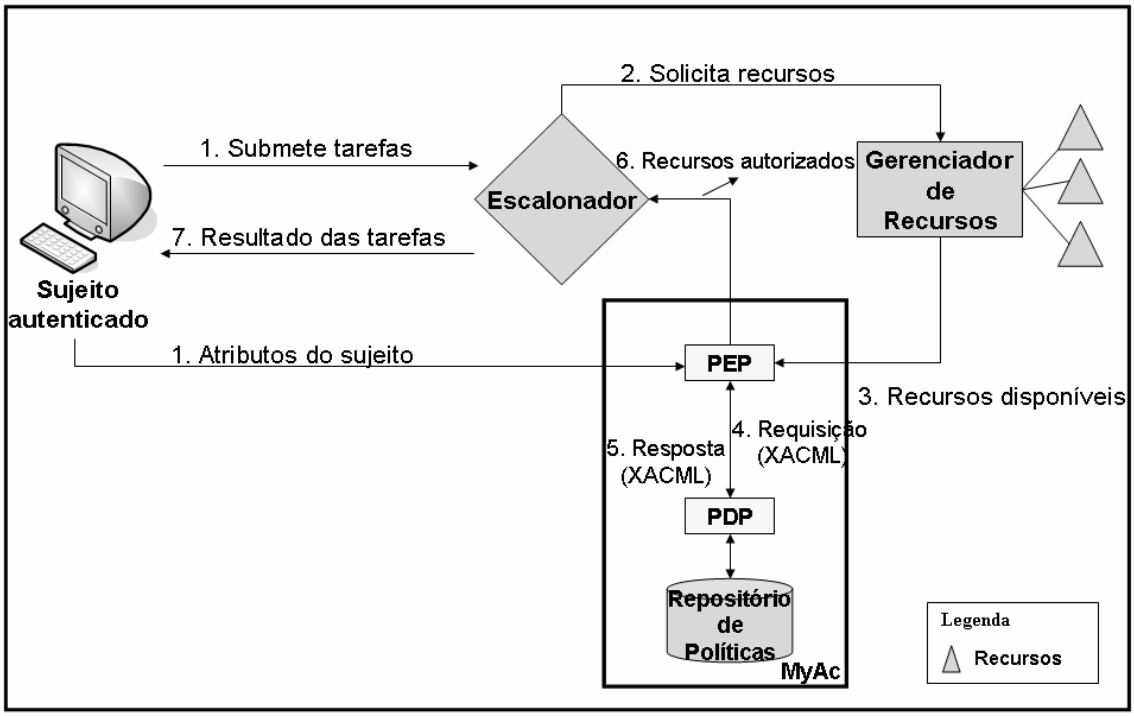

Figura 14. Arquitetura do MyAC.

O processo de autorização é realizado somente após o gerenciador de recursos retornar os recursos efetivamente disponíveis e que satisfaçam a requisição do cliente. Todo o processo é transparente para o gerenciador de recursos e para o escalonador, sem a necessidade de alteração em seu funcionamento ou algoritmo.

Uma das principais contribuições da arquitetura proposta é a glanularidade das regras de acesso que podem ser criadas. As políticas de segurança podem limitar o acesso tanto de um sujeito a um recurso particular, como também restringir o acesso baseado em informações gerais dos recursos e do sujeito. Isto é possível uma vez que a validação dos direitos de acesso é feita após o retorno dos recursos disponíveis pelo gerenciador de recursos. Desta forma, pode-se implementar políticas mandatórias (**MAC**) e baseadas em papéis (**RBAC**).

A utilização do MyAC, em cada domínio ou sítio da grade, permite o gerenciamento independente das políticas de segurança individuais. A natureza da política associada com os certificados dos usuários depende dos objetivos da comunidade da grade e das organizações virtuais associadas.

3.1.3. MyDel

No contexto de MyGSI, a delegação é utilizada para evitar a troca massiva de certificados entre os membros da grade, uma vez que a utilização de uma entidade certificadora geraria um ponto de centralização na grade.

MyDel é o módulo responsável pela delegação de direitos de acesso entre os membros da grade. Esta delegação baseia-se no conceito das correntes ou cadeias de certificados (ver Seção 2.2.4).

Com a utilização de MyDel, cada nó da rede delega seus direitos de acesso (total ou parcialmente) a um novo nó que deseja entrar na rede e que não possui relações de confiança com os outros nós existentes. Da mesma forma, esta corrente de certificados pode ser criada para nós existentes e utilizada para intermediar comunicações que antes não possuíam confiabilidade, evitando a troca de chaves entre os nós comunicantes e aumentando a escalabilidade da grade P2P.

3.2. Protótipo MyGSI

3.2.1. Considerações gerais

O protótipo MyGSI foi desenvolvido na linguagem JAVA versão 1.4.2 do J2SE (*Java 2 Plataform Standard Edition*) para suportar uma grande quantidade de sistemas de grades existente (Globus, OurGrid, entre outras). Além disso, a linguagem JAVA facilita o desenvolvimento de aplicações amigáveis e dispõe de uma vasta biblioteca que trabalha com os protocolos de comunicação de rede [JAVA 2005]. Os componentes de segurança de terceiros utilizados pelo protótipo MyGSI também são escritos nesta linguagem, facilitando a integração.

O desenvolvimento da aplicação foi dividido entre as funções de segurança suportadas: autenticação, controle de acesso e delegação.

3.2.2. Autenticação – MyAuth

MyAuth trabalha com certificados no formato X.509 versão 3 e cria canais seguros de comunicação através do protocolo SSL\TLS. Desta forma, o usuário da grade que já possui seu certificado X.509 pode utilizá-lo no MyGSI sem a necessidade da geração de um novo.

Para suportar a comunicação SSL/TLS, MyAuth especializa os construtores dos objetos *sockets* da linguagem JAVA.

Presentes no pacote *java.net.socket*, os métodos *createSocket()* e *createServerSocket()* utilizam as fábricas *SSLSocketFactory* e *SSLServerSocketFactory* pertencentes à classe *javax.net.ssl* para a geração de canais seguros no lado cliente como mostra a Figura 15 e no lado servidor ilustrado na Figura 16 respectivamente. Além disso, é necessário habilitar a autenticação do cliente, provendo autenticação mútua, através do método *setNeedClientAuth()*.

```
public Socket createSocket(String host, int port) throws IOException {

    /* the client socket factory based on SSL protocol */
    SSLSocketFactory factory;

    /* initializations */
    factory = null;
    factory = (SSLSocketFactory) SSLSocketFactory.getDefault();
    Socket socket = factory.createSocket(host, port);
    return socket;
}
```

Figura 15. Especialização da classe Socket (lado cliente).

```
public ServerSocket createServerSocket(int port) throws IOException {


    /* the server socket factory based on SSL protocol */
    SSLServerSocketFactory factory;

    /* A server socket instance */
    SSLServerSocket serverSocket;

    /* initializations */
    factory = null;

    factory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
    serverSocket = (SSLServerSocket) factory.createServerSocket(port);
    serverSocket.setNeedClientAuth(true); //mutual authentication
    return serverSocket;
}
```

Figura 16. Especialização da classe ServerSocket (lado servidor).

Vale ressaltar que a troca de chaves entre as partes comunicantes é necessária para o processo de autenticação. Dois nós podem trocar informação somente se ambos possuírem a chave pública do outro, caso contrário, a comunicação não pode ser estabelecida.

O processo de geração e troca de chaves não é definida por MyAuth, podendo desta forma ser realizada por diversos meios. Por exemplo: através de *email*, troca *off-line* de chaves (através de disquetes), entre outros.

3.2.3. Controle de Acesso - MyAC

O módulo de controle de acesso tem como fases no seu processo de desenvolvimento a escolha de uma linguagem padrão para as políticas e a implementação do PEP e PDP.

A. Formato das Políticas

XACML (*eXtensible Access Control Markup Language*) versão 1.0 é adotada como linguagem padrão para expressar as políticas e trocas de mensagens realizadas entre o PEP e o PDP [OASIS 2005].

XACML é uma linguagem padrão aberto (*open-standard*) que permite controlar o acesso a sistemas completos bem como a um recurso específico. O formato das políticas XACML permite também criar regras baseadas nos atributos do sujeito, dos recursos e do ambiente, além do formato XML possibilitar a manipulação e visualização de dados através de uma estrutura fácil de ser interpretada por humanos (ver Anexo B). Outros fatores que levam à escolha do padrão XACML, como formato padrão das políticas, são suas características de [OASIS 2005]:

- combinar regras e políticas em um único conjunto de políticas (*PolicySet*), que são aplicadas em uma solicitação de decisão;
- definir de forma flexível procedimentos pelos quais regras e políticas são combinadas;
- lidar com atributos multivalorados;
- fornecer um conjunto de operadores lógicos e matemáticos que podem operar sobre os atributos do sujeito, do recurso e do ambiente;
- manipular um conjunto distribuído de componentes de política, enquanto abstrai o método para localizar, retornar e autenticar os componentes de política;

- identificar de forma rápida as políticas que se aplicam a uma determinada ação, baseada nos valores dos atributos do sujeito, recurso e ambiente;
- fornecer uma camada de abstração que isola o criador da política dos detalhes do ambiente da aplicação.

B. Entidades – PEP e PDP

A biblioteca SunXACML é utilizada no desenvolvimento das entidades PEP e PDP.

SunXACML é uma implementação aberta do padrão XACML escrito em JAVA. Desenvolvida pela Sun Microsystems Laboratories, o projeto SunXACML dá suporte completo para todas as características obrigatórias do XACML [SUNXACML 2005].

O pacote SunXACML apresenta classes que implementam os processos de criação, de requisições e respostas XACML, bem como a avaliação de políticas. O pacote também fornece classes para a geração de políticas XACML.

PEP e PDP são implementados no MyGSI por três objetos, conforme a Figura 17: MyGsiPDP, MyGsiPEP e MyGsiFinderModule.

- **MyGsiPDP**

Classe que implementa o PDP. Possui os métodos básicos de carregar as políticas (*loadPolicies()*) e avaliar as requisições submetidas retornando uma decisão (*evaluate() e evaluateRequest()*).

- **MyGsiPEP**

Classe que implementa o PEP. Responsável pela formatação, no padrão XACML, dos atributos do sujeito, de informações do recurso a ser acessado e da ação que deseja realizar a ser submetido ao MyGsiPDP. Possui métodos que atribuem o usuário (*setUser()*), a máquina (*setMachine()*) e a ação (*setAction()*) desejada para a requisição de acesso (*makeRequest()*).

MyGsiPEP é responsável por aplicar a decisão de acesso do MyGsiPDP (*getResponse()*) no sistema da grade.

- **MyGSIFinderModule**

Módulo de localização de políticas utilizado pelo MyGsiPDP. Os localizadores de políticas são usados tanto para localizar políticas que satisfaçam uma determinada requisição,

quanto para encontrar as políticas referenciadas por um conjunto de políticas (*policy set*). Os módulos de localização devem ser implementados para evitar o retorno de mais de uma política aplicável para uma dada requisição e com ações opostas.

O código-fonte das classes de MyAC estão no ANEXO C-1.

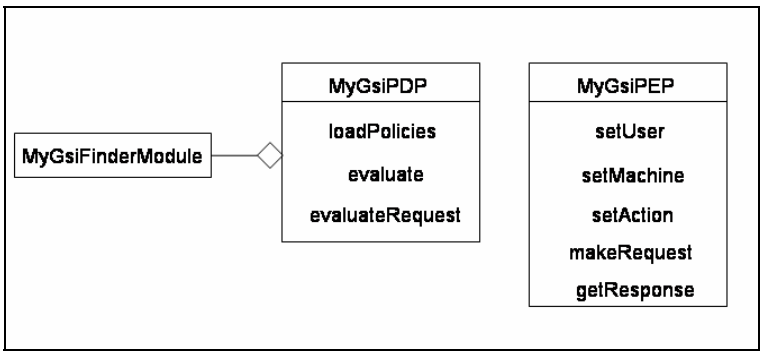

Figura 17. Diagrama de classes – MyAC.

3.2.4. Delegação - MyDel

As bibliotecas padrões do J2SE v. 1.4 não permitem a criação de certificados X.509 e também a criação de correntes de certificados. Neste caso, é necessário a utilização do pacote BouncyCastle, uma biblioteca de distribuição livre. BouncyCastle é um pacote JAVA que contém algoritmos criptográficos, objetos que permitem a criação de certificados no formato X.509 versão 3 e de correntes de certificados [BOUNCY CASTLE 2005].

MyDel utiliza este pacote e representado pelo objeto MyCertChain (ANEXO C-2), consiste de uma ferramenta de geração de corrente de certificados, tendo como método principal o *createDelegatedCert()*.

3.2.5. Dificuldades Encontradas

Durante a implementação do protótipo MyGSI, algumas dificuldades foram encontradas: o formato dos certificados X.509 e a limitação das bibliotecas de segurança do J2SE.

Os certificados X.509 possuem um padrão de difícil entendimento e implementação. Os formatos de dados não são legíveis e são expressos na notação *Abstract Syntax Notation One* (ASN.1). O ASN.1 é um padrão poderoso e complexo, o que torna a infra-estrutura de chaves públicas X.509 difícil de se implementar e usar [CLARKE et al. 2001].

MyGSI não implementa o suporte à utilização de Listas de Certificados Revogados (CRL - *Certificate Revocation Lists*), que invalidam certificados antes de suas datas de expiração. As CRLs são constantemente emitidas, elas são datadas, assinadas digitalmente e informam quando a próxima CRL será emitida. Porém, é considerável o atraso entre a notificação da CRL e a revogação de algum certificado que está sendo utilizado na grade. O SSL/TLS não verifica as listas de revogação de certificados, e atualmente as CRLs estão próximas do desuso completo [GERCK 2000].

Outra dificuldade encontrada é a limitação das bibliotecas padrões de segurança do J2SE de não permitirem a criação dos certificados, implicando a utilização de bibliotecas de terceiros. Além disso, existem poucas alternativas de pacotes que permitem a criação de certificados X.509 e de suas correntes.

3.3. Trabalhos Relacionados

Alguns mecanismos de segurança utilizados por outros sistemas de grade computacional são apresentados nesta seção.

3.3.1. I-WAY

I-WAY utiliza um ambiente de autenticação uniforme que permite a entidade se autenticar somente uma vez antes de obter acesso aos recursos geograficamente distribuídos.

O objetivo principal da segurança no I-WAY é a criação de um único mecanismo de autenticação, utilizado por todos os domínios que compõe a grade. Não existe a preocupação de contabilização, autorização, privacidade e integridade dos dados.

Quando uma entidade deseja utilizar os recursos de seu domínio, ele deve se autenticar e suas informações são armazenadas pelo escalonador e utilizadas para obtenção de recursos. O problema está na obtenção de recursos em outros domínios, a entidade deve novamente se autenticar, exigindo uma conta para cada domínio participante, prejudicando a escalabilidade do sistema [FOSTER et al. 1996].

O tipo de autenticação usado no I-WAY é difícil de ser utilizado em sistemas escaláveis, pois os usuários inevitavelmente precisam acessar recursos localizados em domínio remotos, os quais eles podem não possuir quaisquer relação de confiança.

3.3.2. Globus Toolkit (versão 3)

A autenticação, proteção da comunicação e autorização no Globus é responsabilidade do GSI (GSI - *Grid Security Infrastructure*). GSI estende os protocolos da camada de transporte seguro para atingir alguns requisitos de segurança em grades: autenticação única, delegação, integração com soluções locais, e relacionamentos de confiança baseado nos usuários [FOSTER, 2003].

As identidades utilizadas na grade estão no formato X.509 e o controle de autorização é suportada através de um conjunto de ferramentas que permitem os donos dos recursos integrarem políticas locais, através de uma interface de controle de autorização e acesso genéricos (GAA – *Generic Authorization and Access*).

O mecanismo de delegação também permite uma entidade delegar somente um subconjunto de seus privilégios totais para outra entidade.

3.3.3. Community Authorization Service

Community Authorization Service (CAS) implementa o gerenciamento de controle de acesso através da introdução de uma terceira parte denominado servidor CAS, utilizando políticas capazes de especificar quem (qual usuário ou grupo) tem direito de acessar qual recurso ou grupo de recursos. Através do tipo de direito de acesso ou permissão é possível limitar as ações do usuário [PEARLMAN et al. 2002].

A arquitetura CAS tem como componentes principais um servidor central (*CAS Server*) para o armazenamento dos direitos de acesso da comunidade ou do domínio e um servidor de recursos (*Resource Server*), que armazena as políticas aplicadas sobre as capacidades dos usuários. A utilização desses servidores cria pontos de centralização no processo de controle de acesso, além dos administradores primeiramente definirem os direitos de acesso da comunidade para posteriormente definirem as capacidades de cada usuário.

Quando um usuário deseja utilizar recursos da grade, ele primeiro obtém todas as suas capacidades no servidor CAS para depois verificar seus direitos de acesso juntamente com o servidor de recursos (Figura 18).

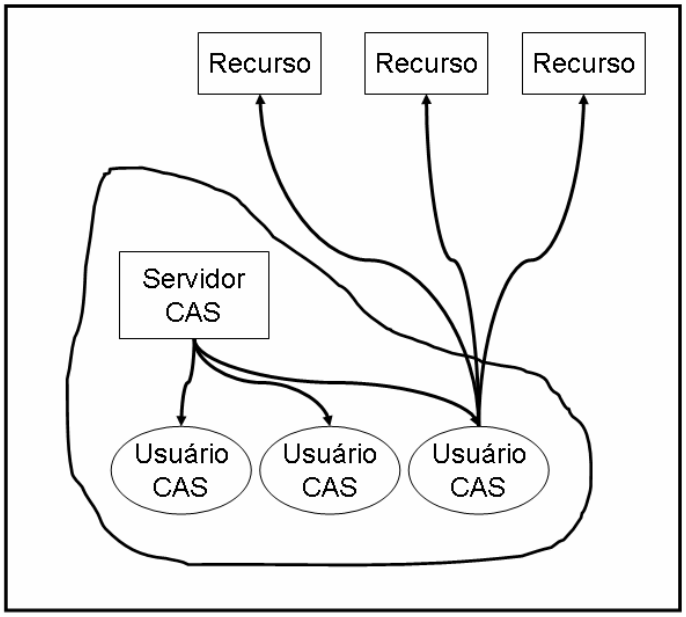

Figura 18. Visão geral do CAS [PEARLMAN et al. 2002].

3.3.4. CARDEA Authorization System

Utilizando entidades de arquitetura baseada em políticas, CARDEA Authorization System, é um sistema de autorização distribuída, desenvolvido pela NASA Information Power Grid, que através da utilização de certificados *proxy* X.509 realiza o controle de acesso. Seu foco principal da autorização baseia-se nas identidades globais dos usuários delegadas aos certificados X.509 [LEPRO 2003].

3.3.5. PeGAC

PeGAC (*Peer-to-peer Grids Access Control*) implementa um controle de acesso para grades P2P baseado no modelo RBAC [SILVA 2005]. PeGAC não é um sistema independente e funciona como serviço da arquitetura *Peer-to-Peer Security Layer Framework* (P2PSLF) [DETSCH 2004].

PeGAC usa as informações do usuário e dos requisitos do recurso para validar o controle de acesso. Estas informações baseiam-se em um conjunto de requisitos (por exemplo, memória acima de 256MB, sistema operacional Linux).

3.3.6. Outros Sistemas

PRIMA (*Privilege Management and Authorization*) é um sistema de controle de acesso para grades computacionais que gerencia privilégios de granularidade fina, na

autorização. PRIMA utiliza certificados X.509 e insere os componentes PEP e PDP no processo de controle de acesso [LORCH et al. 2003].

Além dos sistemas mencionados, pode-se citar também o SlashGrid Project [SLASHGRID PROJECT 2005] que utiliza listas de controle de acesso escritas em XML para controlar o acesso aos sistemas de arquivo da grade.

3.4. Considerações Sobre os Trabalhos Relacionados

A arquitetura MyGSI possui alguns diferenciais frente as soluções estudadas na literatura e mencionadas nas seções anteriores.

I-WAY representa uma primeira tentativa de implementação de segurança para grades, mesmo não possuindo mecanismos para controlar o acesso aos recursos após a autenticação das entidades.

O Globus não permitia o controle de acesso baseada em comunidades, até incorporar o CAS em sua arquitetura de segurança. Além disso, a configuração de seu ambiente seguro não é uma tarefa simples, devida a grande quantidade de componentes presentes em seu modelo.

No MyGSI, as políticas que definem as regras de direito de acesso dos usuários aos recursos são armazenadas em um repositório de políticas, externo à aplicação da grade e consultadas através das entidades responsáveis pelas decisões de acesso (PDP). Não possui dois níveis de armazenamento de políticas ou de direitos de acesso como no CAS. Além disso, o usuário não precisa transportar todas as suas capacidades sobre os recursos para requisitar acesso, os pontos de aplicação de políticas (PEP) presente no MyAC extraem somente as informações necessárias para a aquisição dos direitos de acesso.

MyAC utiliza as políticas de controle baseadas nas identidades dos domínios locais dos usuários. Já o CARDEA utiliza identidades globais.

PeGAC não permite a criação de regras que controlem um recurso em particular, mas recursos que satisfazem um conjunto de requisitos definidos na grade (por exemplo, bloquear o acesso aos recursos que possuem o sistema operacional Linux). Enquanto que MyGSI possibilita definições das regras e políticas de acesso com uma granularidade mais fina.

Apesar da similaridade de algumas características de MyGSI e PRIMA, como a utilização do PEP e PDP, o uso de políticas em formato XACML e a identificação baseada em certificados X.509, MyGSI é totalmente gerenciado no nível da aplicação, enquanto PRIMA emprega alguns mecanismos de controle de acesso diretamente no sistema operacional. Por exemplo, PRIMA utiliza listas de controle de acesso POSIX, que estão disponíveis em sistemas operacionais UNIX [POSIX 2005].

O processo de integração de MyGSI em uma grade real e a verificação de seus resultados, são abordados no próximo capítulo.

Capítulo 4

Estudo de Caso – MyGSI e OurGrid

Introdução

Este capítulo descreve o funcionamento de MyGSI integrado ao OurGrid / MyGrid [ANDRADE et al. 2003; CIRNE et al. 2003].

MyGrid é definido como uma solução de grade para a execução de aplicações Bag-of-Tasks (aplicações cujas tarefas são independentes) utilizando os recurso disponíveis ao usuário. Já OurGrid estabelece uma rede P2P dos sítios que compartilham recursos com o objetivo de formar uma grade, funcionando como um escalonador de recursos [COSTA et al. 2004].

4.1. MyGrid

Como é descrito acima, MyGrid é uma solução de grade computacional para a execução de aplicações do tipo BoT (*bag of tasks*), desenvolvida por pesquisadores da Universidade Federal de Campina Grande (UFCG) na Paraíba, utilizando os recursos acessíveis ao usuário.

Uma característica importante de MyGrid é a utilização de abstrações relativamente simples para o gerenciamento de recursos, submissão de serviços (*jobs)*, escalonamento de atividades, acompanhamento e coleta de informações processadas na grade. Outro aspecto importante a ser mencionado é que MyGrid utiliza protocolos tradicionais como o SSH (*Secure Shell*) [SANDES 2005].

4.1.1. Características

As características básicas de MyGrid são as seguintes: simplicidade, completude, distribuição.

- **Simplicidade**

Permite a execução de aplicações de forma relativamente simples. As implicações disto envolvem a aplicabilidade restrita apenas às aplicações BoT;

- **Completude**

O objetivo é permitir uma visão completa do ciclo de produção, do desenvolvimento à execução, bem como manipulação de E/S (entradas e saídas). Para isto, abstrações como o ambiente de trabalho (*working environment)* são utilizadas para permitir uma manipulação mais fácil dos arquivos. O ambiente de trabalho de MyGrid consiste na utilização das abstrações de arquitetura que permitem a visão da grade como um único elemento computacional [CIRNE et al. 2003].

- **Distribuição**

Permite ao usuário utilizar todos os recursos disponíveis no sistema.

4.1.2. Arquitetura

A arquitetura de MyGrid pode ser resumida nos seguintes componentes [COSTA et al. 2004]: máquina base ou *home machine*, e máquina da grade ou *grid machine*.

- **Máquina base ou *home machine***

Consiste na máquina responsável por coordenar a execução de aplicações BoT através de MyGrid. Através dela, o usuário submete tarefas que compõem a aplicação a ser executada na grade.

- **Máquina da grade ou *grid machine (GuM)***

São máquinas que efetivamente compõem a grade computacional e nas quais são executados os processos previamente determinados pelo usuário. Cada *grid machine* possui um *user agent*. O *user agent* implementa as funcionalidades necessárias de comunicação com a *home machine*, sendo responsáveis pela execução da tarefa.

Juntamente com os ambientes de *scripts*, variáveis de configuração do ambiente como *playpens* (relacionado com o tamanho de espaço para armazenamento temporário) e *requirements* (os requisitos pelos quais determinadas máquinas da grade satisfazem as condições para processamento de determinados *jobs*) devem ser previamente definidas pelo usuário, a fim de permitir a execução das aplicações no MyGrid. Tais informações servem de entrada para o funcionamento do escalonador [COSTA et al. 2004].

4.2. OurGrid

Enquanto que o MyGrid escalona as aplicações do usuário utilizando todos os recursos acessíveis por esse usuário no seu domínio administrativo, OurGrid funciona como um escalonador de recursos [ANDRADE et al. 2003].

OurGrid implementa uma grade P2P para execução de aplicações BoT através de MyGrid [ANDRADE et al 2005]. É através dele que os recursos são disponibilizados para a comunidade, além disso, o OurGrid é responsável pela obtenção de recursos em outros domínios, quando a demanda de processamento local esgota as capacidades dos recursos. OurGrid pode ser também referenciado como GuMP (G*rid Machine Provider* – provedor de máquinas da grade) [ANDRADE et al 2005].

A obtenção e o compartilhamento de recursos são baseados na rede-de-favores [ANDRADE et al. 2004]. Cada nó OurGrid armazena informações sobre o empréstimo de recursos de seu domínio (débito) ou obtenção temporária de recursos (crédito) de outros nós. Quando uma requisição de recursos é feita por um nó externo chega ao OurGrid, ele verifica a sua dívida com este nó e define a prioridade proporcional ao crédito, incentivando a colaboração de domínios da grade.

4.2.1. Integração com MyGrid

A integração entre o OurGrid e o MyGrid ocorre através do arquivo GDF *(grid description file)*. O GDF, mostrado na Figura 19, é um arquivo texto que identifica quais os nós OurGrids devem ser utilizados.

```
# public.gdf file
peer:
name: public.deti.ufc.br
port: 3083
```

Figura 19. Exemplo de arquivo GDF.

A configuração dos recursos gerenciados pelo OurGrid também utiliza arquivos semelhantes ao GDF, denominados SDF (s*ite description files*) e ilustrado na Figura 20.

```
gumdefaults:
    site : deti.ufc.br
    type : ualinux
    os : linux
    processorfamily : IA32
    mem : 640
    remExec : ssh -x $machine $command
    copyFrom : scp $machine:$remotefile $localfile
    copyTo : scp $localfile $machine:$remotefile
    port : 1234
gum:
    name: gum1.deti.ufc.br
    mem : 256
gum:
    name: gum2.deti.ufc.br
gum:
    name:gum3.deti.ufc.br
    type: uawindows
    os: windows
    port: 2351
```

Figura 20. Exemplo de arquivo SDF.

A submissão de tarefas se dá através da criação de arquivos de descrição de tarefas (JDF – *job description file*). Um arquivo JDF, exemplificado na Figura 21, possui todas as informações que o MyGrid necessita saber sobre sua aplicação, como: o número de tarefas, os requisitos físicos e lógicos das tarefas, etc.

```
job :
   label : myjob1
   requirements : ( os = linux and mem > 100 )
task :
    init : put input-1.dat input-1.dat
          put mytask mytask
    remote : mytask < input-1.dat > output-1.res
    final : get output-1.res output-1.res
task:
    init : put input-2.dat input-2.dat
          put mytask mytask
    remote : mytask < input-2.dat > output-2.res
    final : get output-2.res output-2.res
```

Figura 21. Exemplo de arquivo JDF.

4.2.2. Obtenção de Recursos

OurGrid, conforme Figura 22, obedece o seguinte fluxo na execução das tarefas:

1. o sujeito submete o serviço a ser processado para a máquina base, através do JDF;
2. o serviço é enviado para a GuMP, sendo então dividido em tarefas;
3. GuMP localiza os recursos disponíveis (GuMs) e distribui as tarefas;
4. as tarefas são processadas nas GuMs e seus resultados retornados para a GuMP;
5. GuMP retorna o resultado das tarefas para a máquina base.

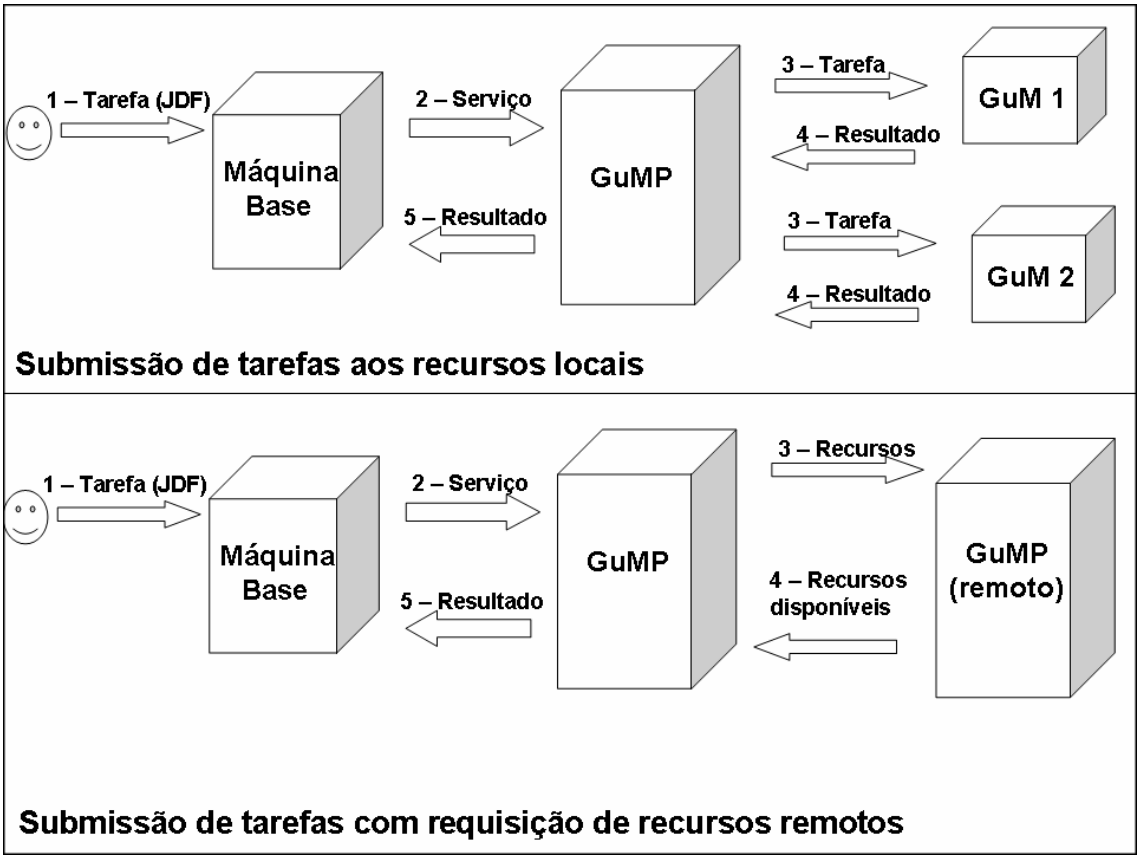

Figura 22. Submissão de tarefas no OurGrid.

Para o usuário é transparente a localização e alocação dos recursos a serem utilizados para o seu processamento, fica a cargo do OurGrid verificar a disponibilidade dos recursos locais ou a necessidade de obtenção dos recursos externos.

Três entidades participam do processo de obtenção de recursos executado pelo OurGrid: o escalonador, o provedor de recursos e as máquinas da grade. Semelhante ao processo mencionado na Seção 1.3.5, quando uma requisição de processamento de tarefas é recebida pelo escalonador, ele verifica quantas máquinas são necessárias para executar a tarefa e pede ao provedor de máquinas os recursos disponíveis. Se o número de máquinas disponíveis não satisfazer a requisição, outros provedores de recursos remotos serão consultados.

4.2.3. Segurança

Em relação à segurança, OurGrid e MyGrid utilizam áreas especiais para a execução de tarefas dos usuários nas máquinas da grade, denominadas “playpens” [COSTA et al. 2004]. A utilização do protocolo SSH na comunicação entre as entidades da grade e a transferência de arquivos é uma tentativa de implementação de canais de comunicação seguros, mas este

protocolo se restringe como serviço proposto pelo sistema operacional das máquinas (Unix ou Linux).

O controle de acesso no OurGrid é bastante limitado, sendo possível somente duas opções de decisão: negar todos os recursos ou permitir o acesso completo. Isto prejudica bastante o crescimento da grade, inibindo o compartilhamento de recursos, por exemplo, com o ingresso de novos nós (desconhecidos) para alguns usuários da grade, é natural o não compartilhamento de recursos entre nós que não possuem relação de confiança.

4.3. Estudo de Caso

MyGSI foi integrado com sucesso na versão 3.0 do OurGrid[6]. O módulo de autenticação foi desenvolvido no projeto Kanindé, em parceria com o Departamento de Engenharia de Teleinformática (DETI) e de Computação (DC) da Universidade Federal do Ceará (UFC), Laboratório de Sistemas Distribuídos (LSD) da UFCG e o Instituto Atlântico, como parte de um projeto maior denominado Pauá.

O Projeto Pauá ou Grade Pauá é um dos esforços de construção de uma grade computacional a nível nacional. Financiado pela HP-Brasil, a Grade Pauá é a implementação de uma grade geograficamente distribuída, utilizando de forma transparente e não-dedicada, o tempo de máquinas instaladas nas instituições que compõem o projeto, utilizando-se das tecnologias OurGrid e MyGrid.

O projeto Kanindé foi responsável pelo estudo e implementação de mecanismos, algoritmos e políticas de segurança aplicáveis ao OurGrid como reforço da comunicação entre as instituições [KANINDE 2005].

A Figura 23 mostra o ambiente de teste utilizado pelo MyGSI, composto de onze máquinas, sendo duas configuradas como GuMPs, duas como máquinas base e seis como máquinas da grade, simulando dois domínios administrativos distintos (*atlantico1 e atlantico2*). Para a reprodução fiel do ambiente utilizado pela Grade Pauá, introduziu-se um CorePeer. O CorePeer é um super-nó responsável pela indexação dos GuMPs disponíveis na grade. Ele também armazena as informações de cada domínio da grade, coletando

[6] Nesta seção o sistema OurGrid mencionado também contempla a utilização do MyGrid.

informações da quantidade de recursos e de créditos e débitos da rede de favores da grade [OURGRID 2005].

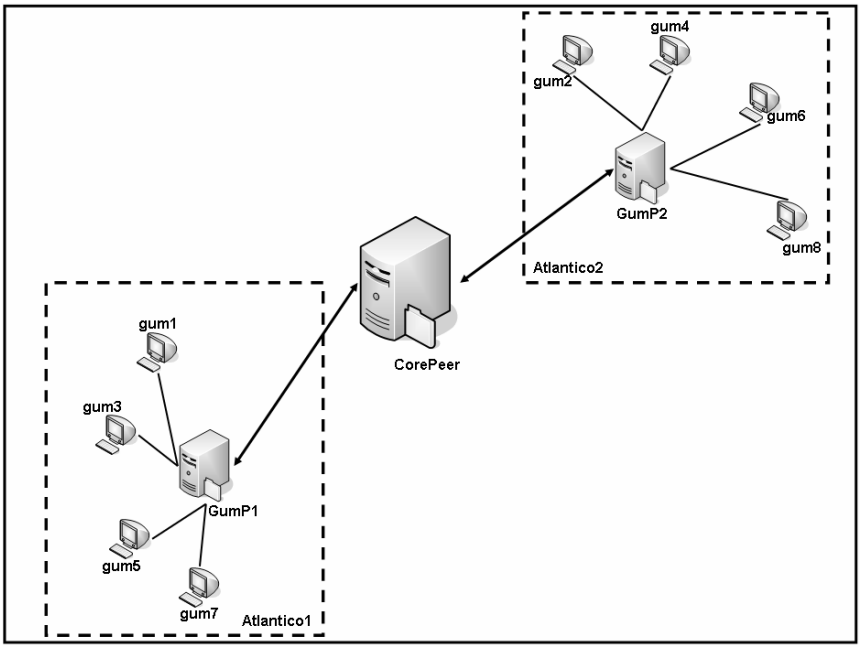

Figura 23. Ambiente de teste do MyGsi.

4.3.1. MyAuth

MyAuth é incorporado no OurGrid modificando a classe de criação de sockets pelas classes de sockets seguro do MyGSI, passando a utilizar o uso do protocolo SSL/TLS para o transporte seguro das informações e limitando a utilização dos recursos na prevenção de operações maliciosas (conforme Seção 3.2.2) [OURGRID 2005].

Os canais de comunicação seguros são utilizados nas submissões de tarefas, entre a máquina base e o provedor de recursos, na distribuição de tarefas, entre o provedor de recursos e nas máquinas da grade no mesmo domínio, como também nas solicitações de recursos externos feita entre os provedores de recursos da grade.

A ferramenta Keytool é utilizada para a geração e troca automática dos mesmos, sendo um utilitário de gerenciamento de certificados e chaves que já vem instalada no J2SE, ele possibilita ao usuário administrar seus pares de chaves públicas e privadas [KEYTOOL 2005].

4.3.2. MyAC

Para integrar MyAC ao OurGrid é necessário primeiramente definir o que deve ser controlado: quais recursos e atributos, as ações que podem ser executadas sobre os recursos e os atributos dos usuários e do sistema. Desta forma, MyAC usa como recursos as máquinas da grade (GuMs), como atributos do usuário as informações presentes nos certificados X.509 e como atributos do sistema a data e o horário local do domínio.

A Figura 24 ilustra a arquitetura MyAC adaptada com as entidades do OurGrid. Nesta Figura, setas largas indicam a troca de mensagem padrão do OurGrid, independente da utilização ou não do MyAC, e as setas mais finas mostram as mensagens trocadas entre as entidades de MyAC.

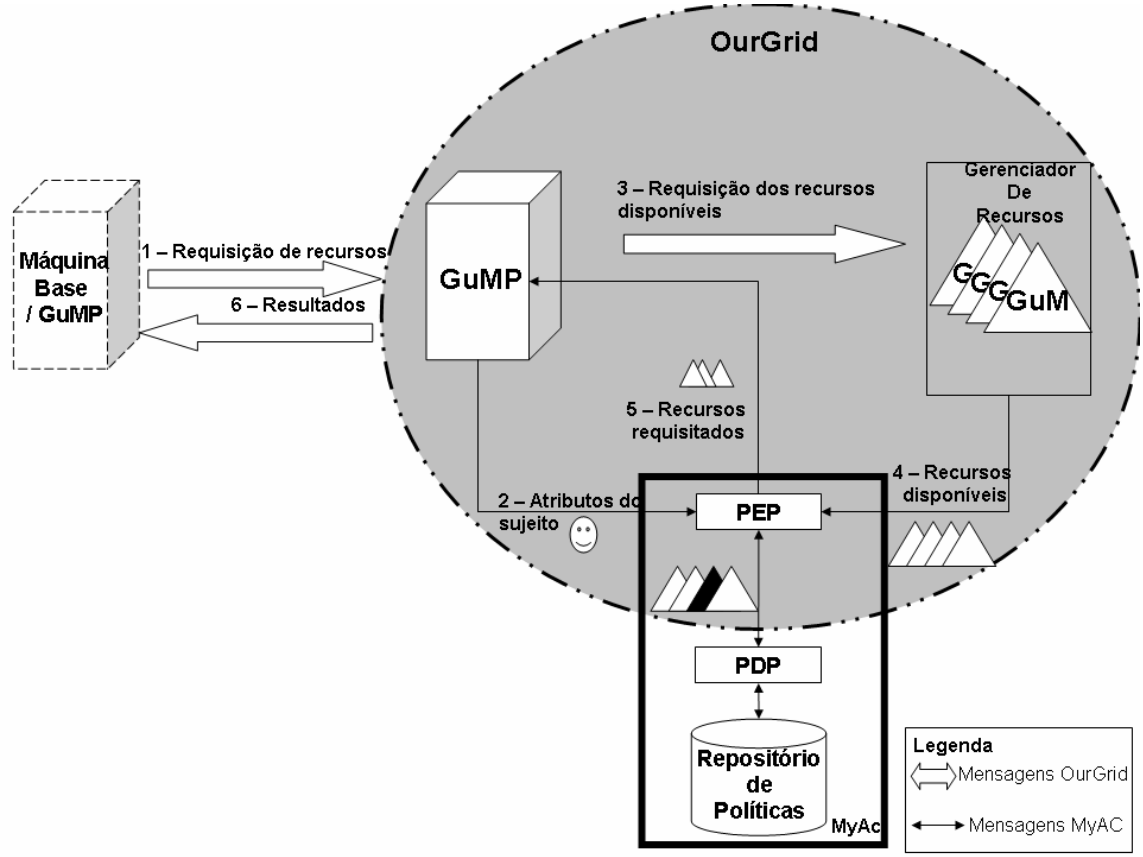

Figura 24. MyAC integrado com OurGrid.

O GuMP analisa as requisições internas e externas ao seu domínio, e baseado na identificação do usuário (passo 2 da Figura 24) e nos recursos efetivamente disponíveis que satisfazem a requisição (passo 4) submete ao MyGsiPDP através do MyGsiPEP, no formato XACML, o pedido de acesso. Os recursos requisitados e autorizados são retornados ao GuMP

como disponibilidade ou não do recurso (passo 5), e depois os resultados do processamento enviados para a máquina solicitante (passo 6).

Note na Figura 24 que o PEP é o único componente do MyAC a ser introduzido no código do OurGrid. É através do PEP que se obtém os atributos mencionados do código do OurGrid. Funcionando como um tradutor do formato de informações utilizadas pelo ambiente da grade pra requisições na linguagem XACML.

A introdução do PEP modifica somente duas classes do OurGrid para a obtenção das informações necessárias da requisição do controle de acesso (*PeerHeuristic.deliverGuM()*, *OurGumProviderImpl.wannaGuMs()*), facilitando desta forma a integração com MyAC.

O PDP é implementado como um serviço remoto RMI (*Remote Method Invocation*) (ANEXO C1) [JAVA RMI 2005]. A opção em possuir um PDP como um serviço remoto permite que o objeto seja instanciado em uma máquina diferente daquela que está usando o OurGrid, servindo como opção de isolamento das máquinas da grade. O repositório de políticas utilizado é o próprio sistema de arquivos da máquina, reservando um diretório para o armazenamento das políticas. Logo, quando uma política é definida e escrita ela é armazenada na máquina do PDP em um diretório particular.

Foram criadas vários cenários de utilização das políticas de acesso aos recursos para validação do comportamento previsto do MyAC. Destacam-se dois cenários que mostram o controle de acesso refinado de MyAC e a utilização dos atributos do sistema.

A. Cenário Um

No cenário um, exemplificado na Figura 25, é criada uma política XACML que nega o acesso de utilização das máquinas *gum2* e *gum4* ao usuário David. Este tipo de política serve para limitar o acesso à máquinas que possuem conteúdos sensíveis do domínio.

```
<?xml version="1.0" encoding="UTF-8"?>
 <Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         PolicyId="Cenario1"
         RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
 <Description>
  This policy to block the access to gum2 and gum4 machines to David user.
 </Description>
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">David</AttributeValue>
     <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
    </SubjectMatch>
   </Subject>
  </Subjects>
  <Resources>
   <AnyResource/>
  </Resources>
  <Actions>
   <AnyAction/>
  </Actions>
 </Target>
 <Rule RuleId="AccessRule" Effect="Deny">
  <Target>
   <Resources>
    <Resource>
     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gum2 </AttributeValue>
      <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
     </ResourceMatch>
    </Resource>
    <Resource>
     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gum4 </AttributeValue>
      <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
     </ResourceMatch>
    </Resource>
   </Resources>
   <Actions>
    <AnyAction/>
   </Actions>
  </Target>
 </Rule>
</Policy>
```

Figura 25. Política do cenário um.

B. Cenário Dois

No cenário dois, é criada uma política, mostrada na Figura 26, que restringe o acesso de qualquer usuário do domínio atlantico2 às máquinas do domínio atlantico1 durante o período das oito horas (08:00:00) às dezessete horas (17:00:00).

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      PolicyId="Cenario2"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">

 <Description>
  This policy denies access to resources for users of atlantico2 domain between 08:00 and 17:00 hours.
 </Description>

 <Target>
    <Subjects>
     <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">atlantico2 </AttributeValue>
       <SubjectAttributeDesignator DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
                          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
      </SubjectMatch>
     </Subject>
    </Subjects>
  <Resources>
   <AnyResource/>
  </Resources>
  <Actions>
   <AnyAction/>
  </Actions>
 </Target>
 <Rule RuleId="BloqueiaAcesso" Effect="Deny">
  <Condition FunctionId="http://research.sun.com/projects/xacml/names/function#time-in-range">
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#time"
                          AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
   </Apply>
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">08:00:00</AttributeValue>
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
  </Condition>
 </Rule>
</Policy>
```

Figura 26. Política do cenário dois.

No primeiro cenário a política utiliza atributos do usuário e das máquinas, enquanto que no segundo são utilizados atributos do sistema. A política do cenário um está armazenada no repositório de políticas do domínio atlantico2 e a segunda política está armazenada no repositório de políticas do domínio atlantico1. Cabe observar que, ambos os domínios, gerenciam o acesso aos seus recursos de maneira independente, respeitando os domínios administrativos diferentes.

Quando uma tarefa é submetida ao GumP2, configurado como mostrado na Figura 27, o PEP obtém os atributos do usuário que submete a tarefa e consulta o PDP para cada recurso disponível retornado pelo provedor de recursos.

```
david@gump2:~$ peer status
Security is enabled!
OurGrid Peer 3.0 is Up and Running
Local GuMs:
        grid2
        grid4
        grid6
        grid8
Community GuMs: none
Network of Favors Accounting:
        Peer Id                         Peer Balance
        7019723740799493120             0.00
```

Figura 27. Configuração do domínio Atlantico2.

No caso da tarefa ter sido submetida pelo usuário David, a troca de mensagens XACML entre PEP e PDP indicam a negação de acesso na tentativa de utilização da máquina grid2, mesmo estando disponível. A Figura 28 é uma cópia da tela do computador mostrando a negação de acesso à máquina *grid2* ao usuário David.

```
Fazendo requisicao user: david + machine: grid2
<Request>
<Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>david</AttributeValue></Attribute>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
<AttributeValue>david@atlantico1</AttributeValue></Attribute>
</Subject>
<Resource>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>grid2</AttributeValue></Attribute>
<Attribute AttributeId="IP"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>grid2</AttributeValue></Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>process</AttributeValue></Attribute>
</Action>
</Request>

Resposta do PDP : <Response>
<Result ResourceID="grid2">
<Decision>Deny</Decision>
<Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
</Response>

Maquina grid2 NEGADA para user: david
```

Figura 28. Acesso negado.

Caso a máquina seja uma das permitidas (grid6 ou grid8), o PDP retorna a permissão de acesso. Neste caso, a Figura 29 é uma cópia da tela do computador mostrando a permissão de acesso à máquina grid6 ao usuário David.

```
Fazendo requisicao user: david + machine: grid6
<Request>
<Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>david</AttributeValue></Attribute>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
<AttributeValue>david@atlantico1</AttributeValue></Attribute>
</Subject>
<Resource>
<Attribute AttributeId="IP"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>grid6</AttributeValue></Attribute>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>grid6</AttributeValue></Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>process</AttributeValue></Attribute>
</Action>
</Request>

Resposta do PDP : <Response>
<Result ResourceID="grid6">
<Decision>Permit</Decision>
<Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
</Response>

Maquina grid6 PERMITIDA para user: david
```

Figura 29. Permissão de acesso.

Verificando a execução desta tarefa pelo escalonador, através do comando "*mygrid status*" do OurGrid, se comprova o funcionamento correto das políticas de controle de acesso. A Figura 30 mostra a tela do computador listando o *status* do *MyGrid*, somente as máquinas *grid6* e *grid8* processaram as tarefas submetidas pelo usuário David.

```
mgk-2212@home2:~$ mygrid status
Security is enabled!
MyGrid 3.0 is Up and Running
MyGuMP:
Remote Grid Machine Providers:
 => rmi://gump2:2010/gumprovider
Scheduler Grid Machines: none
Jobs:
        Job 1: SimpleJob [Finished]
                Task 1: [Finished]
                        Replica 1: [Finished] - assigned to grid6
                Task 2: [Finished]
                        Replica 1: [Finished] - assigned to grid8
                Task 3: [Finished]
                        Replica 1: [Finished] - assigned to grid6
                Task 4: [Finished]
                        Replica 1: [Finished] - assigned to grid8
                Task 5: [Finished]
                        Replica 1: [Finished] - assigned to grid6
                Task 6: [Finished]
                        Replica 1: [Finished] - assigned to grid8
                Task 7: [Finished]
                        Replica 1: [Finished] - assigned to grid6
                Task 8: [Finished]
                        Replica 1: [Finished] - assigned to grid8
                Task 9: [Finished]
                        Replica 1: [Finished] - assigned to grid6
                Task 10: [Finished]
                        Replica 1: [Finished] - assigned to grid8
```

Figura 30. Validação das políticas.

4.3.3. MyDel

Devida à necessidade da autenticação mútua, as máquinas devem realizar previamente a troca de chaves. Para o ambiente utilizado são realizadas cinqüenta e quatro trocas de chaves, enumeradas na Tabela 1.

No. De Trocas	Máquina	Contém as chaves públicas de
2	CorePeer	GumP1, GumP2
10	GumP1	CorePeer, GumP2, gum1, gum2, gum3, gum4, gum5, gum6, gum7, gum8
10	GumP2	CorePeer, GumP1, gum1, gum2, gum3, gum4, gum5, gum6, gum7, gum8
4	gum1	GumP1, gum3, gum5, gum7
4	gum2	GumP1, gum4, gum6, gum8
4	gum3	GumP1, gum1, gum5, gum7
4	gum4	GumP1, gum2, gum6, gum8
4	gum5	GumP1, gum1, gum3, gum7
4	gum6	GumP1, gum2, gum4, gum8
4	gum7	GumP1, gum1, gum3, gum5
4	gum8	GumP1, gum2, gum4, gum6

Tabela 6. Trocas de chaves.

A troca massiva de chaves entre as entidades da grade afeta a escalabilidade da rede e o melhor aproveitamento do potencial da grade, uma vez que com o crescimento da grade seja provável a resistência da troca de certificados entre administradores e domínios desconhecidos.

A adição de novos recursos na grade e o ingresso de novos domínios participantes são utilizados como exemplos para mostrar o benefício de se utilizar MyDel.

Para cada novo recurso inserido no domínio *atlantico1*, deve ser criado um novo certificado para este recurso e depois ser feita a troca de certificados com cada máquina do seu domínio. Em relação ao ingresso de um novo domínio, por exemplo, o domínio *atlantico3*, para que ele aproveite a potencialidade da grade, deve ser feita a troca de certificados entre o provedor de recursos *gump3* com o *gump1* e com o *gump2*. Se não houver

relação de confiança entre o domínio *atlantico2* e este novo domínio, não haverá troca de certificados e a grade será subutilizada.

Uma possível solução para este problema é a adoção de entidades certificadoras na grade, mas isso cria pontos de centralização, o que não é uma boa abordagem em ambientes distribuídos e descentralizados como são os sistemas P2P.

A ferramenta MyDel contorna este problema mencionado da seguinte forma: como ferramenta de criação de correntes de certificados, MyDel é utilizado na confecção de cadeias de certificados de dois níveis para resolver o problema da troca de chaves no ambiente OurGrid com CorePeer. Dois níveis é o tamanho mínimo para que qualquer *peer* obtenha o potencial de acesso a todos os recursos da grade Pauá contendo o CorePeer.

Para solucionar o primeiro exemplo, o novo recurso (por exemplo, *gum7*) deve enviar seu certificado para o administrador da rede gerar a corrente de certificados sobre a tutela de algum recurso confiável do domínio. Pode ser uma corrente de certificados do *gum7*, utilizando a confiança do certificado do *gum5*, que por sua vez é um recurso confiável por todo o domínio.

Para a geração da corrente de certificados:

1. o certificado do *gum7* é enviado para o *gum5*;
2. *Gum5* executa o comando *MyCertChain.createDelegatedCert(<certificado gum5>,<certificado gum7>)* e gera a corrente;
3. a corrente é enviada para o *gum7*, e por sua vez é utilizado como certificado deste nó.

A Figura 31 mostra a corrente de certificados gerada para o gum7 com a responsabilidade do gum5.

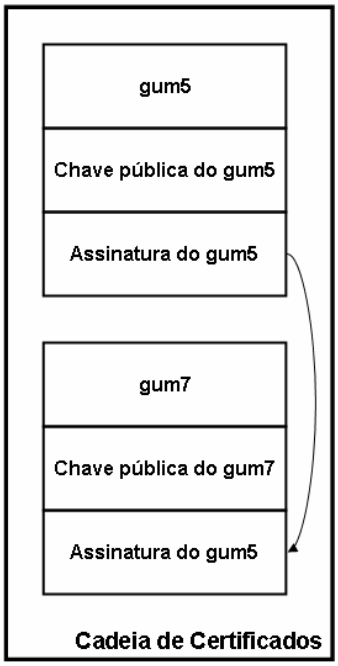

Figura 31. Corrente de certificados do gum7.

O segundo exemplo é um pouco diferente, para que a *gump3* ingresse na grade, ela deve ser confiável por algum GumP da grade. Supondo que o *gump1* confie no *gump3*, o bastante para se responsabilizar pelas ações deste novo domínio, desta forma deve ser seguido os mesmos passos de geração da corrente de certificados, mencionados anteriormente no primeiro exemplo.

Como o CorePeer é a entidade que conhece todos os *peers* da grade Pauá, este novo recurso ou novo domínio sendo tutelado por um membro já confiável da grade e do CorePeer, consegue acessar todos os outros *peers*.

O próximo capítulo reúne as vantagens obtidas da utilização de MyGSI. A contribuição de MyGSI para a comunidade científica e trabalhos futuros também se fazem presentes neste próximo capítulo.

Capítulo 5

Conclusões e Trabalhos Futuros

MyGSI é uma proposta de arquitetura de segurança para grades P2P. As três partes principais de MyGSI são: autenticação (MyAuth), controle de acesso (MyAC) e delegação de direitos de acesso (MyDel).

MyAuth tem como características a utilização de infra-estrutura de chaves públicas, para fornecer a autenticação de entidades da grade e a geração canais de comunicação segura utilizando o protocolo SSL.

MyAC, por sua vez, utiliza as entidades PEP e PDP para descentralizar o controle de acesso da grade, possibilitando a coexistência de políticas distintas para cada domínio que compõe a grade. Além disso, MyAC permite a criação de políticas com regras de acesso com granularidade fina, relacionando usuários remotos ou locais a um determinado recurso, como também políticas baseadas nas informações das comunidades dos usuários, por exemplo, o nome de seu domínio.

MyDel, como o nome indica, trata do problema da delegação de direitos de acesso através da criação de correntes de certificados. Desta forma, uma entidade da grade pode transmitir seus direitos de acesso, ou parte deles, para outra entidade.

MyGSI foi implementado na versão 3.0 do OurGrid como estudo de caso, e políticas que limitam o acesso de determinados usuários a determinados recursos como também políticas baseadas em características do domínio são utilizadas.

O uso da linguagem XACML nas políticas de segurança e trocas de mensagem entre o PEP e PDP, fornece uma sintaxe simples de definição dos atributos dos recursos, dos usuários e do sistema que seriam controlados.

Somente o PEP é introduzido no código do OurGrid, funcionando como “tradutor” do retorno dos recursos disponíveis pelo gerenciador de recurso ao escalonador para o formato XACML.

A utilização de um PDP e de um repositório de políticas de políticas externos ao sistema, permite um gerenciamento flexível do controle de acesso da grade: a manipulação de políticas não necessita parar a execução da grade, tornando possível a edição e inserção de políticas com a grade em execução.

Embora MyGSI não seja uma ferramenta totalmente acabada, os testes até agora realizados produziram resultados satisfatórios. Por exemplo, a Grade Pauá está adotando os mecanismos de autenticação de MyAuth.

Como resultados do estudo e desenvolvimento de MyGSI, destaca-se o trabalho de [SANTOS et al. 2005].

Os mecanismos de controle de acesso e delegação também são divulgados na comunidade científica através de dois artigos, referenciados em [VALE et al. 2005] e [VALE et al. 2004].

MyGSI não desenvolve um módulo de gerenciamento de políticas, responsável pela geração e inserção das políticas no repositório. As políticas utilizadas são geradas através de editores de texto, de forma manual. O desenvolvimento deste módulo é um trabalho a ser realizado no futuro. Além disso, o mecanismo de delegação (MyDel) de MyGSI pode ser aprimorado para permitir a criação de correntes de certificados de vários níveis. MyDel implementa somente correntes de dois níveis, suficientes para a utilização com o OurGrid e o CorePeer.

O protótipo de MyGSI baseia-se fortemente na arquitetura utilizada pelo OurGrid, sendo desenvolvido como objetos padrões JAVA. O estudo da viabilização de implementar MyGSI em uma arquitetura orientada à serviços (SOA – *Service Oriented Architecture*) também é uma possibilidade de extensão deste trabalho.

I. Referências Bibliográficas

[ABRAMSON et al. 1995].ABRAMSON, David et al. Nimrod: a tool for performing parametrised simulations using distributed workstations. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 4., 1995. Proceedings... Los Alamitos: IEEE Computer Society, 1995. p. 112.

[ABRAMSON et al. 2001].ABRAMSON, David et al. ActiveSheets: SuperComputing with Spreadsheets. In: ADVANCED SIMULATION TECHNOLOGIES CONFERENCE, 10., 2001, Seattle. High Performance Computing Symposium. Seattle: Society For Computer Simulation International, 2001. p. 110 - 115.

[ALLEN et al. 2001].ALLEN, Gabrielle et al. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In: SUPERCOMPUTING, 2001, Denver. Proceedings... New York: ACM Press, 2001. p. 52 - 52.

[ALMOND 1999].ALMOND, Jim; SNELLING, Dave. UNICORE: uniform access to supercomputing as an element of electronic commerce. Future Generation Computer Systems, v. 15, n. 5-6, p.539-548, out. 1999.

[ANDERSON 2001] .ANDERSON, Ross J.. Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley Computer Publishing, 2001.

[ANDRADE et al. 2003].ANDRADE, Nazareno et al. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In: WORKSHOP ON JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, 9., 2003, Seattle. Proceedings...Springer, 2003. p. 61 - 86.

[ANDRADE et al. 2004].ANDRADE, Nazareno et al. Discouraging Free Riding in a Peer-To-Peer CPU-Sharing Grid. In: INTERNATIONAL SYMPOSIUM ON HIGH-PERFORMANCE DISTRIBUTED COMPUTING, 13., 2004, Honolulu. : The Leading Technical Conference on Grids & Distributed Computing. Honolulu: IEEE Computer Society, 2004. p. 129 - 137.

[ANDRADE et al 2005].ANDRADE, Nazareno et al. Peer-to-peer Grid computing with the OurGrid Community. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 23., 2005, Fortaleza. IV Salão de Ferramentas. Fortaleza: UFC, 2005.

[AOL 2005].AOL. AIM. Disponível em: <http://americaonline.com.br/>. Acesso em: 04 set. 2005.

[BARAK 1999].BARAK, Amnon; WHEELER, Richard. MOSIX: an integrated multiprocessor UNIX. In: MILOJICIC, Dejan; DOUGLIS, Frederick; WHEELER, Richard. Mobility: processes, computers, and agents. New York: ACM Press/Addison-Wesley Publishing Co., 1999. p. 41-53.

[BERMAN 2003].BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. The Grid: past, present, future. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 9-50.

[BERMAN et al 1996].BERMAN, Fran et al. Application-Level Scheduling on Distributed Heterogeneous Networks. In: THE INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS, 1996, Pittsburgh. Proceedings... Pittsburgh: ACM Press, 1996. CD-ROM.

[BOUNCY CASTLE 2005] The Legion of the Bouncy Castle. Disponível em: <http://www.bouncycastle.org/latest_releases.html>. Acesso em: 20 jan. 2005.

[BUTLER et al. 2000].BUTLER, Randy et al. Design and Deployment of a National-Scale Authentication Infrastructure. IEEE Computer Society, v. 33, n. 12, p.60-66, dez. 2000.

[BUYYA 2000].BUYYA, Rajkumar; ABRAMSON, David; GIDDY, Jonathan. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING IN ASIA-PACIFIC REGION, 0., 2000, Beijing. Proceedings...IEEE Computer Society Press, 2000. p. 698 - 714.

[CALLADO et al. 2004].CALLADO, Arthur et al. Peer-to-Peer: Computação Colaborativa na Internet. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 22., 2004, Gramado. Minicursos. Gramado: UFRGS, 2004. p. 3 - 46.

[CHIEN 2003].CHIEN, Andrew A. Architecture of a commercial enterprise desktop Grid: the Entropia system. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid

Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 337-350.

[CIRNE et al. 2003].CIRNE, Walfredo et al. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 32., 2003, Kaohsiung. Proceedings... IEEE Computer Society, 2003. p. 407.

[CLARKE et al. 2001]. .CLARKE, D. et al. Certificate Chain Discovery in SPKI/SDSI. Journal of Computer Security, p. 285-322. 2001.

[CLARKE et al. 2005].CLARKE, Ian et al. The Freenet Project. Disponível em: <http://freenet.sourceforge.net/>. Acesso em: 04 set. 2005.

[CONDOR PROJECT 2005].THE Condor Project. Disponível em: <http://www.cs.wisc.edu/condor/>. Acesso em: 04 set. 2005.

[COSTA et al. 2004].COSTA, Lauro et al. MyGrid: A complete solution for Running Bag-of-Tasks Applications. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 22., 2004, Gramado. Salão de Ferramentas. Gramado: UFRGS, 2004.

[CZAJKOWSKI 2004].CZAJKOWSKI, Karl; FOSTER, Ian; KESSELMAN, Carl. Resource and Service Management. In: FOSTER, Ian; KESSELMAN, Carl. The Grid 2 - Blueprint for a New Computing Infrastructure. 2. ed. San Francisco: Morgan Kauffman, 2004. p. 260-283.

[CZAJKOWSKI et al. 1998].CZAJKOWSKI, Karl et al. A resource management architecture for metacomputing systems. In: WORKSHOP ON JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, 4., 1998. Proceedings... Heidelberg: Springer-Verlag, 1998. p. 62 - 82.

[CZAJKOWSKI et al. 2001].CZAJKOWSKI, Karl et al. Grid information services for distributed resource sharing. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 10., 2001, San Francisco. Proceedings... Los Alamitos: IEEE Computer Society Press, 2001. p. 181 - 184.

[DATA GRID PROJECT 2005].DATA Grid Project. Disponível em: <http://eudatagrid.web.cern.ch/eu-datagrid>. Acesso em: 04 set. 2005.

[DETSCH 2004]. DETSCH, André; GASPARY, Luciano. Towards a Flexible Security Framework for Peer-to-Peer based Grid. In: INTERNATIONAL WORKSHOP ON MIDDLEWARE FOR GRID COMPUTING, 2., 2004, Toronto. Proceedings... Toronto: 2004.

[DIERKS 2005].DIERKS, T.; ALLEN, C.. The TLS Protocol - Version 1.0. Disponível em: <http://www.ietf.org/rfc/rfc2246.txt>. Acesso em: 05 set. 2005.

[FAFNER 2005].FAFNER. Disponível em: <http://www.npac.syr.edu/factoring.html>. Acesso em: 04 set. 2005.

[FERRAIOLO 2005] FERRAIOLO, David; KUHN, Rick. An Introduction to role-based access control. Disponível em: <http://csrc.nist.gov/rbac/NIST-ITL-RBAC-bulletin.html>. Acesso em: 20 nov. 2005

[FOSTER 1997].FOSTER, Ian; KESSELMAN, Carl. Globus: A Metacomputing Infrastructure Toolkit. Internacional Journal of Supercomputing Applications, p. 115-128. 1997.

[FOSTER, 2003].FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. The Anatomy of the Grid. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 171-198.

[FOSTER 2003a].FOSTER, Ian. The Grid: A new infrastructure for 21st century science. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 51-64.

[FOSTER 2004].FOSTER, Ian; KESSELMAN, Carl, TUECKE, Steven. The Open Grid Services Architecture. In: FOSTER, Ian; KESSELMAN, Carl. The Grid 2 - Blueprint for a New Computing Infrastructure. 2. ed. San Francisco: Morgan Kauffman, 2004. p. 215-257.

[FOSTER 2004a].FOSTER, Ian; KESSELMAN, Carl. Concepts and Architecture. In: FOSTER, Ian; KESSELMAN, Carl. The Grid 2 - Blueprint for a New Computing Infrastructure. 2. ed. San Francisco: Morgan Kauffman, 2004. p. 37-63.

[FOSTER et al. 1996].FOSTER, Ian et al. Software infrastructure for the I-WAY high-performance distributed computing experiment. In: HIGH PERFORMANCE DISTRIBUTED COMPUTING, 5., 1996, Syracuse. Proceedings... Washington: IEEE Computer Society, 1996. p. 562 - 571.

[FOSTER et al. 1998].FOSTER, Ian et al. A security architecture for computational grids. In: CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 5., 1998, San Francisco. Proceedings... New York: ACM Press, 1998. p. 83 - 92.

[FOSTER et al. 2003].FOSTER, Ian et al. The Physiology of the Grid. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 217-249.

[FOX et al. 2003].FOX, Geoffrey et al. Peer-to-peer Grids. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 471-490.

[FREIER 2005].FREIER, Alan O.; KARLTON, Philip; KOCHER, Paul C.. SSL Procol V. 3.0. Disponível em: <http://wp.netscape.com/eng/ssl3/ssl-toc.html>. Acesso em: 05 set. 2005.

[FREY et al. 2001].FREY, James et al. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 10., 2001. Proceedings... Los Alamitos: IEEE Computer Society, 2001. p. 55 - 66.

[GAO 2004].GAO, Raymond. An Overview of Peer-To-Peer Technology, Platforms, and Key Concerns. In: IADIS INTERNATIONAL CONFERENCE, 2004, Lisboa.

[GASSER 1990].GASSER, Morrie; MCDERMOTT, Ellen. An architecture for practical delegation in a distributed system. In: SYMPOSIUM ON SECURITY AND PRIVACY, 1990, Oakland. Proceedings... Oakland: IEEE, 1990. p. 20 - 30.

[GEHRING 1996].GEHRING, Jörn; REINEFELD, Alexander. MARS—a framework for minimizing the job execution time in a metacomputing environment. Future Generation Computer Systems, Amsterdam, v. 12, n. 1, p.87-99, 1996.

[GERCK 2000]. GERCK, E.. Overview of Certification Systems: X.509, CA, PGP and SKIP. Disponível em: <http://www.mcg.org.br/cert.htm>. Acesso em: 05 set. 2005.

[GLOBUS PROJECT 2005].THE Globus Project Web Site. Disponível em: <http://www.globus.org>. Acesso em: 04 set. 2005.

[GNUTELLA 2005].GNUTELLA. Disponível em: <http://www.gnutella.com/>. Acesso em: 04 set. 2005.

[GRIMSHAW 1997].GRIMSHAW, Andrew. The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM, New York, v. 40, n. 1, p.39-45, jan. 1997.

[GROPP et al. 2003].GROPP, William et al. The Sourcebook of Parallel Computing. San Francisco: Morgan Kaufmann, 2003.

[HASTINGS 2000]. HASTINGS, Nelson E.; POLK, W. Timothy. Bridge Certifiction Authorities: Connecting B2B Public Key Infrastructures. NIST, 2000.

[HOSCHEK 2000].HOSCHEK, Wolfgang et al. Data Management in an International Data Grid Project. In: IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING, 1., 2000, Bangalore. Proceedings... London,: Springer-Verlag, 2000. p. 77 - 90.

[HOWELL 2000].HOWELL, Jon; KOTZ, David. End-to-end authorization. In: SYMPOSIUM ON OPERATING SYSTEMS DESIGN, 2000, San Diego. Proceedings... San Diego: USENIX Association, 2000. p. 151 - 164.

[IBM AUTONOMIC COMPUTING 2005].IBM Autonomic Computing. Disponível em: <http://www.research.ibm.com/autonomic/>. Acesso em: 04 set. 2005.

[JAVA 2005]. JAVA. Disponível em: <http://java.sun.com/j2se/1.4/>. Acesso em: 04 set. 2005.

[JAVA RMI 2005]. JAVA RMI.Disponível em: <http://java.sun.com/products/jdk/rmi/>. Acesso em: 09 set. 2005.

[JOHNSTON 2003].JOHNSTON, William et al. Implementing production Grids. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 117-167.

[KANINDE 2005].KANINDE Project. Disponível em: <www.atlantico.com.br/~kaninde>. Acesso em: 09 maio 2005.

[KAZAA 2005].KAZAA. Disponível em: <http://www.kazaa.com/>. Acesso em: 04 set. 2005.

[KEYTOOL 2005]. KEYTOOL. Disponível em: <http://java.sun.com/j2se/1.3/docs/tooldocs/>. Acesso em: 04 set. 2005.

[KRAUTER 2002].KRAUTER, Klaus; BUYYA, Rajkumar; MAHESWARAN, Muthucumaru. A taxonomy and survey of grid resource management systems for

distributed computing. Software—Practice & Experience, New York, v. 32, n. 2, p.135-164, fev. 2002.

[KUNSZT 2003].KUNSZT, Peter Z., GUY, Leanne P. The Open Grid Services Architecture, and data Grids. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 385-407.

[LAMPSON 1971].LAMPSON, B.W.. Protection. In: PRINCETON SYMPOSIUMON INFORMATION SCIENCE AND SYSTEMS, 5., 1971. Proceedings... ACM, 1971. p. 437 - 443.

[LEPRO 2003]. LEPRO, R.. Cardea: Dynamic Access Control in Distributed Systems. Disponível em: <http://www.nas.nasa.gov/News/Techre-ports/2003/PDF/nas-03-020.pdf>. Acesso em: 20 fev. 2005.

[LITZKOW 1988].LITZKOW, Michael; LIVNY, Miron; MUTKA, Matt. Condor - A Hunter of Idle Workstations. In: INTERNATIONAL CONFERENCE OF DISTRIBUTED COMPUTING SYSTEMS, 8., 1988, San Jose. Proceedings... San Jose: IEEE-CS Press, 1988. p. 104 - 111.

[LORCH et al. 2003]. LORCH, Markus et al. The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments. In: INTERNATIONAL WORKSHOP ON GRID COMPUTING, 4., 2003, Phoenix. Proceedings... Phoenix: 2003.

[LV et al. 2002].LV, Qin et al. Search and replication in unstructured peer-to-peer networks. In: INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, 16., 2002, New York. Proceedings... New York: ACM Press, 2002. p. 84 - 95.

[MILOJICIC et al. 2005].MILOJICIC, Dejan S. et al. HP Laboratories. Peer-to-Peer Computing. Technical Report. Disponível em: <http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.html>. Acesso em: 04 jul. 2005.

[MSN MESSENGER 2005].MICROSOFT. Msn Messenger. Disponível em: <http://messenger.msn.com/>. Acesso em: 04 set. 2005.

[NAHRSTEDT 1999]. NAHRSTEDT, Klara; CHU, Hao-hua; NARAYAN, Srinivas. QoS-aware resource management for distributed multimedia applications. Journal of High Speed Networks, Amsterdam, p. 229-257. jan. 1999.

[NAPSTER 2005].NAPSTER. Disponível em: <http://www.napster.com/>. Acesso em: 04 set. 2005.

[NETSOLVE 2005].NETSOLVE RPC Based Network Computing. Disponível em: <http://icl.cs.utk.edu/netsolve/>. Acesso em: 04 set. 2005.

[NINF 2005].NINF Global Computing Infrastructure. Disponível em: <http://ninf.apgrid.org/>. Acesso em: 04 set. 2005.

[OASIS 2005]. OASIS Standard. XACML 1.0 Specification, http://www.oasis-open.org, acessado em 20 de Janeiro de 2005.

[OURGRID 2005].OURGRID. Disponível em: <http://www.ourgrid.org/>. Acesso em: 09 maio 2005.

[P2P ARCHITECT PROJECT 2003].P2P Architect Project: Ensuring dependability of P2P applications at architectural level. Disponível em: <http://www.atc.gr/p2p_architect/results/0101F05_P2P%20Survey.pdf>. Acesso em: 04 set. 2005.

[PEARLMAN et al. 2002]. PEARLMAN, L. et al. A Community Authorization Service for Group Collaboration. In: INTERNATIONAL WORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 3., 2002, Monterey. Proceedings... Monterey: IEEE, 2003.

[POSIX 2005]. POSIX. Disponível em <http://wt.xpilot.org/publications/posix.1e>. Acesso em: 09 maio 2005.

[RAMAN 1998].RAMAN, Rajesh; LIVNY, Miron; SOLOMON, Marvin. Matchmaking: distributed resource management for high throughput computing. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 7., 1998. Proceedings... Los Alamitos: IEEE Computer Society Press, 1998. p. 140.

[ROURE et al. 2003].ROURE, David De et al. The evolution of the grid. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 65-100.

[ROWSTRON 2001].ROWSTRON, Antony; DRUSCHEL, Peter. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM

INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS, 18., 2001, Heidelberg. Proceedings... Heidelberg: ACM, 2001. p. 329 - 350.

[SANDES 2005].SANDES, Edans. SSH - Secure Shell. Disponível em: <http://www.cic.unb.br/docentes/pedro/trabs/SSH.htm>. Acesso em: 05 set. 2005.

[SANDHU 1993]. SANDHU, Ravi. Lattice-Based Access Control Models. Computer, v. 26, n. 11, p.9-19, nov. 1993.

[SANDHU 1996]. SANDHU, Ravi. Access Control: The neglected Frontier",. In: AUSTRALIAN CONFERENCE ON INFORMATION SECURITY AND PRIVACY, 1., 1996, Wollongong. Proceedings... Wollongong: 1996.

[SANDHU 2000]. SANDHU, Ravi; FERRAIOLO, David; KUHN, Richard. The NIST model for role-based access control: Towards a unified standard. In: ACM WORKSHOP ON ROLE-BASED ACCESS CONTROL, 5., 2000, Berlin. Proceedings... Berlin: ACM Press, 2000. p. 47 - 63.

[SANDHU et al. 1996]. SANDHU, Ravi et al. RoleBased Access Control Models. Computer, v. 29, n. 2, p.38-47, jan. 1996.

[SANTOS et al. 2005].SANTOS, Rodrigo et al. Anti-Doping: An Approach for Grid Integrity Verification. In: ADVANCED INDUSTRIAL CONFERENCE ON TELECOMMUNICATIONS, 0., 2005, Lisboa. Proceedings... Lisboa: IEEE, 2005. p. 2 - 7.

[SCHOLLMEIER et al. 2001].SCHOLLMEIER, Rüdiger et al. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In: INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING, 1., 2001, Linkopings University. Proceedings... Suécia: IEEE, 2001. p. 101 - 102.

[SEGURANÇA 2005]. SEGURANÇA de Redes. Disponível em: <http://www.redes.unb.br/security/>. Acesso em: 04 set. 2005.

[SILVA 2005]. SILVA, Juliano; GASPARY, Luciano. PeGAC: Uma Arquitetura de Controle de Acesso para Grades Computacionais Peer-to-Peer. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 23., 2005, Fortaleza. I Workshop de Peer-To-Peer. Fortaleza: UFC, 2005.

[SLASHGRID PROJECT 2005]. Slashgrid Project. Disponível em: <http://www.gridpp.ac.uk/authz/slashgrid/>. Acesso em: 01 fev. 2005.

[SOARES 1995].SOARES, Luís Fernando Gomes; LEMOS, Guido; COLCHER, Sérgio.Segurança em Redes de Computadores. In: SOARES, Luís Fernando Gomes; LEMOS, Guido; COLCHER, Sérgio. Redes de Computadores - das LANs, MANs e WANs às Redes ATM. 6. ed. Rio De Janeiro: Editora Campus, 1995. p. 447-488.

[STALLINGS 2000].STALLINGS, Willian. Introduction. In: STALLINGS, Willian. Network Security Essentials: Applications and Standards. Upper Saddle River: Prentice-Hall, 2000. p. 4-16.

[STALLINGS 2000a].STALLINGS, Willian. Authentication Applications. In: STALLINGS, Willian. Network Security Essentials: Applications and Standards. Upper Saddle River: Prentice-Hall, 2000. p. 83-116.

[STALLINGS 2000b].STALLINGS, Willian. Key Distribution. In: STALLINGS, Willian. Network Security Essentials: Applications and Standards. Upper Saddle River: Prentice-Hall, 2000. p. 19-46.

[STOICA et al. 2003].STOICA, Ion et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON). Piscataway: IEEE Press, 2003. v. 11, p. 17 - 32.

[SUNXACML 2005]. SUNXACML. Disponível em: <http://sunxacml.sourceforge.net/>. Acesso em: 04 set. 2005.

[TANENBAUM 2002].TANENBAUM, Andrew S.; STEEN, Maarten Van. Security. In: TANENBAUM, Andrew S.; STEEN, Maarten Van. Distributed Systems: Principles and Paradigms. Upper Saddler River: Prentice-Hall, 2002. p. 413-488.

[THAIN 2003].THAIN, Douglas; TANNENBAUM, Todd; LIVNY, Miron. Condor and the Grid. In: BERMAN, Fran; FOX, Geoffrey; HEY, Anthony J.G.. Grid Computing: Making The Global Infrastructure a Reality. Hoboken: John Wiley & Sons, 2003. p. 299-335.

[TOWSLEY 2003].TOWSLEY, Donald. Peer-to-Peer Systems. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 21., 2003, Natal. Tutoriais. Natal: UFRN, 2003.

[VALE et al. 2005]. VALE, João et al. Uma Proposta de Controle de Acesso e Delegação para Grades P2P. In: III FÓRUM OURGRID, 2005, Campina Grande.

[VALE et al. 2004].VALE, João et al. Gerenciamento de Chaves nas Ferramentas PAUÁ. In: II FÓRUM OURGRID, 2004, Campina Grande.

[VERMA 2001]. VERMA, Dinesh. Policy-Enabled Networking Architecture. In: VERMA, Dinesh. Policy-Based Networking: Architecture and Algorithms. Indianápolis: New Riders Publishing, 2001. p. 5-26.

[WEISSMAN 1999]. WEISSMAN, Jon. Prophet: automated scheduling of SPMD programs in workstation networks. Concurrency - Practice and Experience, Amsterdam, v. 11, n. 6, p.301-321, maio 1999.

[XU 2004].XU, Ming et al. Service Virtualization: Infrastructure and Applications. In: FOSTER, Ian; KESSELMAN, Carl. The Grid 2 - Blueprint for a New Computing Infrastructure. 2. ed. San Francisco: Morgan Kauffman, 2004. p. 179-189.

[YAHOO 2005].YAHOO. Yahoo! Messenger. Disponível em: <http://messenger.yahoo.com/>. Acesso em: 04 set. 2005.

[ZHAO et al. 2004].ZHAO, Ben Y. et al. Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications. 2004. v. 22, p. 41 - 53.

Anexo A - Criptografia

A1 - CRIPTOGRAFIA SIMÉTRICA

Dentre os sistemas de criptografia, os que utilizam a mesma chave para os processos de cifragem e decifragem são chamados de simétricos ou privados (também conhecidos como sistemas de chave secreta ou compartilhada). Note que $M = D_k(E_k(P))$. Este tipo de chave é representada por $K_{a,b}$, a chave está compartilhada com os usuários A e B.

Exemplo: DES

O DES (*Data Encryption Standard*) é utilizado em sistemas de criptografia simétrica. O algoritmo trabalha com 64 bits de dados a cada vez. Cada bloco de 64 bits de dados sofre de 1 a 16 iterações (16 é o padrão DES), nas quais para cada iteração um pedaço de 48 bits da chave de 56 bits entra no bloco de cifragem. A decifragem é o processo inverso. A Figura 32 mostra o algoritmo que será detalhado logo adiante.

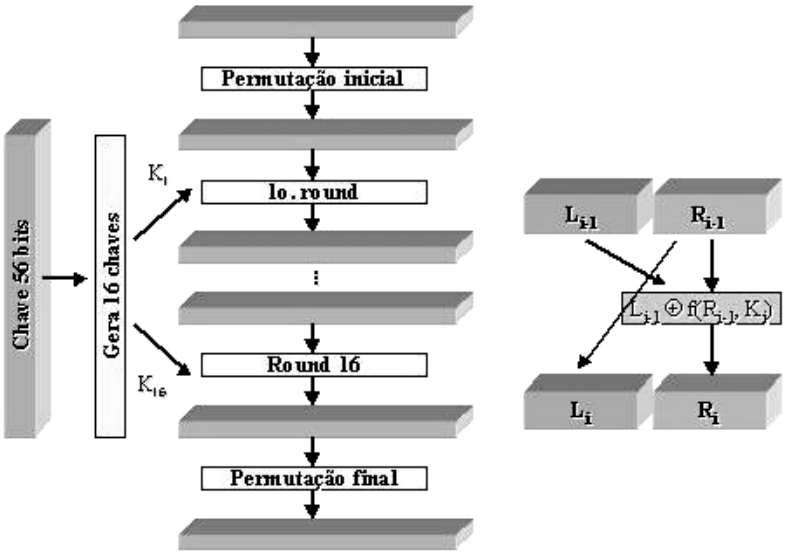

Figura 32. Princípio do DES.

Internamente, a decifragem é o processo inverso. Um exemplo esquemático deste funcionamento está indicado na Figura 33.

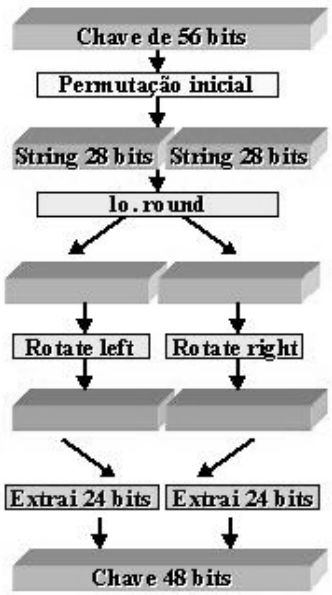

Figura 33. Detalhe na geração do DES.

O algoritmo é difícil de ser quebrado usando métodos analíticos, necessitando de força bruta para esse objetivo. Uma melhoria do DES consiste no *triple*-DES (o algoritmo que tem chave três vezes maior que o DES) e no AES (mais recente e mais seguro). Uma importante característica é que a criptografia simétrica é da ordem de milhares de vezes mais rápida para a cifragem / decifragem de chaves do que a criptografia assimétrica [STALLINGS 2000].

A2 - Criptografia Assimétrica

Quando o sistema utiliza chaves distintas nos processos de cifragem e decifragem, temos os chamados sistemas assimétricos ou de chave pública.

Exemplo: RSA

O algoritmo RSA (conhecido assim devido os seus autores: Rivest, Shamir e Adleman) é utilizado em sistemas de chave pública. Baseado no conceito de primaridade, sua segurança reside em encontrar grandes números primos, tornando este problema computacionalmente impraticável. Os passos para a geração das chaves públicas e privadas são sumariados a seguir:

1. Escolha dois números primos p e q.

2. Seja $n = p * q$ e $z = (p - 1) * (q - 1)$.

3. Escolha um número *e* que seja primo relativo a *z*.

4. Compute *d* tal que $d = e^{-1} \ mod \ z$.

Desta forma, podemos utilizar o número *"d"* para a decifragem e o número *"e"* para a encriptação. Após dividir uma mensagem *M* em blocos, cada bloco a ser enviado é cifrado da forma $C = M^e (mod\ n)$ e decifrado por $M = C^d (mod\ n)$. Veja que para a cifragem, precisamos dos valores de *"e"* e *"n"* e para a decifragem dos valores de *"d"* e *"n"*. O RSA é mais complexo do que o DES, logo ele também é mais seguro e mais lento que o DES [STALLINGS 2000].

A3 - Funções Hash

Uma das aplicações de criptografia em sistemas distribuídos é o uso de uma função *hash* H, que transforma uma mensagem *m* de tamanho variável em uma cadeia de bits *h* de tamanho fixo H(m) = h. Podemos citar como propriedades das funções *hash*:

- **Função unidirecional**, tornando computacionalmente impraticável obter *m*, sabendo-se *h*;
- **Resistência à colisão frágil**, tornando computacionalmente inviável encontrar *m'* (*m'* diferente de *m*), sendo *H(m) = H(m')*;
- **Resistência à colisão forte**, implicando que dado H, é computacionalmente impraticável encontrar *m* e *m*', tais que *H(m) = H(m')*.

Algumas das propriedades expostas são aplicadas às funções de criptografias *E*: é computacionalmente inviável encontrar uma chave *k'* relacionada a uma função *E*, tendo o texto plano *P* e o seu texto cifrado $C = E_k(P)$. Da mesma forma, dado um texto plano *P* e uma chave *k*, impossível encontrar outra chave *k'* tal que $E_k(P) = E_{k'}(P)$.

Anexo B – Linguagem XACML

B1 – Linguagem XACML

A linguagem XACML é definida por dois esquemas[7] XML: o "*xacml context*" e o *"xacml policy"*. O *"xacml context"* define como representar as mensagens de requisição e resposta das políticas entre o PEP e PDP. O *"xacml policy"* define como representar as políticas de controle de acesso. O diagrama UML do *"xacml policy"* está ilustrado na Figura 34.

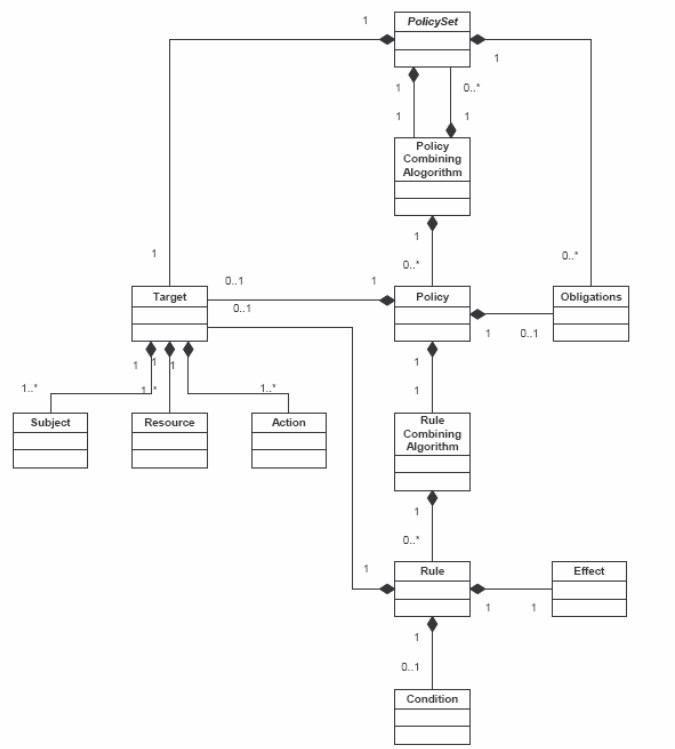

Figura 34. Diagrama UML do padrão XACML.

Uma política XACML é descrita em termos de um conjunto de permissões ou negações de acesso por estruturas denominadas alvos (*targets).* A sintaxe dos alvos expressa:

[7] O esquema XML é um arquivo XML usado para definir a estrutura e o tipo de dados que um documento XML pode conter. O esquema também especifica os elementos, atributos e tipos de dados que podem ser usados em um documento XML, juntamente com a estrutura que deve ser seguida para que o documento seja válido nesse esquema XML em particular.

um usuário (*subject*) pode (ou não pode) executar ações (*actions*) sobre os recursos (*resources*).

Os alvos podem ser associados a uma política (*policy*), a um conjunto de políticas (*policy set*), ou a uma regra (*rule*). Quando associados à política ou a um conjunto de políticas, os alvos funcionam como filtros da política aplicável, isto é, o PEP requisita uma decisão baseada nos alvos, que por sua vez é avaliada com base nas políticas que possuem aquele alvo. Quando associados à regra, os alvos permitem expressar condições de permissão ou negação de acesso. As regras definem que se a condição (*condition*) é satisfeita então o efeito (*effect*) é aplicado sobre o alvo.

Temos somente dois valores para o efeito: negar ou permitir. O efeito define o sentido real de um alvo como a permissão ou negação.

Quando o PEP envia uma requisição para o PDP, ele fornece os atributos permitindo identificar os elementos de um alvo (sujeito, recurso, ação). O PDP avalia as regras das políticas, e então retorna para o PEP o efeito correspondente: permitir ou negar. Se ele falha ao tentar encontrar um alvo nas políticas que satisfazem os atributos fornecidos, retornará a resposta “não aplicável” (n*ot applicable*).

As obrigações (*obligations*), quando definidas, são retornadas pelo PEP em conjunto com a decisão. As obrigações informam um conjunto de ações que devem ser realizadas pelo PEP, complementando a decisão.

Os algoritmos de combinação de políticas ou de combinação de regras são utilizados para selecionar uma política ou regra de um conjunto de políticas ou regras aplicáveis a um alvo.

XACML possui alguns algoritmos de combinação padrões:

- ***Deny-overrides***

Se uma política ou regra é encontrada que avalia a requisição como negação de acesso, então, a resposta será de negação sem verificar o efeito das outras políticas ou regras aplicáveis.

- ***Permit-overrides***

Se uma política ou regra selecionada que avalia a requisição de acesso possui o efeito de permissão de acesso, então, a resposta será de permissão sem verificar o efeito das outras políticas ou regras aplicáveis.

- ***First applicable***

  Retorna o efeito da primeira regra ou política aplicável encontrada.

- ***Only-one applicable***

  Utilizado somente para políticas, esse algoritmo retorna o valor indeterminado (*indeterminate*) se existir mais do que uma política ou conjunto de políticas aplicáveis à requisição de acesso. Se não existe política ou conjunto de políticas aplicáveis, ele retornará o valor “não aplicável”. Havendo somente uma política ou conjunto de políticas aplicável ele retornará o efeito da avaliação dessa única política ou conjunto de políticas.

Anexo C – Código-Fonte de MyGSI

C1 – Código-Fonte de MyAC

Este anexo apresenta o código-fonte do módulo MyAC, representado pelos objetos MyGsiPDP (Figura 35), MyGsiPEP (Figura 36) e MyGsiFinderModule (Figura 37).

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.HashSet;
import java.util.List;
import java.util.ArrayList;
import java.util.Set;


import com.sun.xacml.PDP;
import com.sun.xacml.PDPConfig;
import com.sun.xacml.ParsingException;
import com.sun.xacml.cond.FunctionFactory;
import com.sun.xacml.cond.FunctionFactoryProxy;
import com.sun.xacml.cond.StandardFunctionFactory;
import com.sun.xacml.ctx.RequestCtx;
import com.sun.xacml.ctx.ResponseCtx;
import com.sun.xacml.finder.AttributeFinder;
import com.sun.xacml.finder.PolicyFinder;
import com.sun.xacml.finder.impl.CurrentEnvModule;
import com.sun.xacml.finder.impl.FilePolicyModule;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.ByteArrayInputStream;
import java.io.IOException;

import java.rmi.Naming;

/**
 * @author Joao Carlos S. do Vale
 * *
 * Description: Implementation of PDP.
 *
 * @version 1.0 Created on 24/03/2005
 */
public class MyGsiPDP extends UnicastRemoteObject implements MyAC{
   PDP pdp;

   public MyGsiPDP()throws RemoteException{
         super();
   }

   public void loadPolicies(String directory){
         MyGsiFinderModule policyModule = new MyGsiFinderModule();

         File f = new File(directory);
         String[] files = f.list();
         for (int cont = 0; cont< files.length; cont++)
```

```
            if (files[cont].endsWith(".xml")){
                    policyModule.addPolicy(directory+files[cont]);
                    System.out.println(files[cont]);
            }

            PolicyFinder policyFinder = new PolicyFinder();
            Set policyModules = new HashSet();
            policyModules.add(policyModule);
            policyFinder.setModules(policyModules);

            CurrentEnvModule envModule = new CurrentEnvModule();

            AttributeFinder attrFinder =  new AttributeFinder();
            List attrModules = new ArrayList();
            attrModules.add(envModule);
            attrFinder.setModules(attrModules);

            // Função abaixo utilizada pra carregar o controle de tempo.
        FunctionFactoryProxy proxy =
           StandardFunctionFactory.getNewFactoryProxy();
        FunctionFactory factory = proxy.getConditionFactory();
        factory.addFunction(new TimeInRangeFunction());
        FunctionFactory.setDefaultFactory(proxy);

      pdp = new PDP(new PDPConfig(attrFinder, policyFinder, null));
    }

    public ResponseCtx evaluate(String requestFile) throws IOException, ParsingException{
            RequestCtx request = RequestCtx.getInstance(new FileInputStream(requestFile));
            return pdp.evaluate(request);
    }

    public ResponseCtx evaluate(RequestCtx request){
            return pdp.evaluate(request);
    }

    public String evaluateRequest(String req) throws RemoteException{
            RequestCtx reqCTX;
            ByteArrayOutputStream bt = new ByteArrayOutputStream();
                    try {
                            reqCTX = RequestCtx.getInstance(new ByteArrayInputStream(req.getBytes()));
                            pdp.evaluate(reqCTX).encode(bt);
                            return bt.toString();
                    } catch (ParsingException e) {
                            e.printStackTrace();
                            return null;
                    }
    }

}
```

Figura 35. Código-fonte MyGsiPDP.

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.rmi.Naming;
import java.util.HashSet;
import java.util.Iterator;

import com.sun.xacml.ParsingException;
import com.sun.xacml.attr.RFC822NameAttribute;
import com.sun.xacml.attr.StringAttribute;
import com.sun.xacml.ctx.Attribute;
import com.sun.xacml.ctx.RequestCtx;
```

```
import com.sun.xacml.ctx.ResponseCtx;
import com.sun.xacml.ctx.Subject;
import com.sun.xacml.ctx.Result;

/**
 * @author Joao Carlos S. do Vale - joaocarlos@atlantico.com.br
 *  *
 * Description: Implementation of PEP.
 *
 * @version 1.0 Created on 24/03/2005
 */
public class MyGsiPEP {

        /** HashSet of subjects - the access requester */
        private HashSet subjects;

        /** HashSet of requested resources */
        private HashSet resources;

        /** HashSet of actions to be performed */
        private HashSet actions;

        private boolean deny, permit, notApplicable, indeterminate;

        /**
         * Constructor used to create subjects, resources and actions instances.
         *
         */
        public MyGsiPEP(){
                subjects  = new HashSet();
                resources = new HashSet();
                actions   = new HashSet();
                clearDecisionResult();
        }

        public void newRequest(){
                subjects.clear();
                resources.clear();
                actions.clear();
                clearDecisionResult();
        }

        private void clearDecisionResult(){
                deny = false;
                indeterminate = false;
                notApplicable = false;
                permit = false;
        }

        /**
         * Public method to create a formatted XACML subject with a username and groupname
         * attributes.
         */
        public void setUser(String userName, String groupName) throws URISyntaxException {
                HashSet attributes = new HashSet();
                URI subjectId = new URI("urn:oasis:names:tc:xacml:1.0:subject:subject-id");
                StringAttribute UserNameValue = new StringAttribute(userName);
                attributes.add(new Attribute(subjectId,null,null,UserNameValue));
    subjectId =
       new URI("urn:oasis:names:tc:xacml:1.0:subject:subject-id");
    RFC822NameAttribute value =
       new RFC822NameAttribute(userName + '@' + groupName);
    attributes.add(new Attribute(subjectId, null, null, value));

                subjects.add(new Subject(attributes));
        }
```

```
        /**
         * Public method to create a formatted XACML resource with a IP and machineName
         * attributes.
         */
        public void setMachine(String IP, String machineName)throws URISyntaxException {
                URI resourceId =
    new URI("urn:oasis:names:tc:xacml:1.0:resource:resource-id");
                StringAttribute value = new StringAttribute(machineName);
                resources.add(new Attribute(resourceId,null,null,value));

                value = new StringAttribute(IP);
                resourceId =
    new URI("IP");
                resources.add(new Attribute(resourceId,null,null,value));
        }

        /**
         * Public method to create a formatted XACML action without attributes.
         *
         */
        public void setAction() throws URISyntaxException {
   URI actionId =
    new URI("urn:oasis:names:tc:xacml:1.0:action:action-id");

   actions.add(new Attribute(actionId, null, null,
                 new StringAttribute("process")));
}

        /**
         * Public method to create a XACML Context Request.
         *
         */
        public RequestCtx makeRequest() throws Exception{
                return new RequestCtx(subjects, resources,actions, new HashSet());
        }

        public String requestToString(){
                ByteArrayOutputStream bt = new ByteArrayOutputStream();
                try {
                        makeRequest().encode(bt);
                        return bt.toString();
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }
        }


        public boolean isPermit(){
          return permit;
        }

        public boolean isDeny(){
                return deny;
        }

        public boolean isNotApplicable(){
                return notApplicable;
        }

        public boolean isIndeterminate(){
                return indeterminate;
        }

        /**
```

```
	 * Public static method to create a XACML Context Response using a XACML response
	 * string.
	 */
	public static ResponseCtx getResponse(String resp){
		try {
			return ResponseCtx.getInstance(new ByteArrayInputStream(resp.getBytes()));
		} catch (ParsingException e) {
			e.printStackTrace();
			return null;
		}
	}

	/**
	 * Public method to set the result: isDeny, isPermit, isNotApplicable,
	 * isIndeterminate
	 */
	public void getResult(ResponseCtx response){
		if (!response.getResults().isEmpty()){
			Iterator iter = response.getResults().iterator();
			Result element = (Result) iter.next();
			switch (element.getDecision()) {
				case Result.DECISION_DENY:
					deny = true;
					break;
				case Result.DECISION_INDETERMINATE:
					indeterminate = true;
					break;
				case Result.DECISION_NOT_APPLICABLE:
					notApplicable = true;
					break;
				case Result.DECISION_PERMIT:
					permit = true;
					break;
			}
		}
	}

	public void printResult(){
		if (isPermit()) System.out.println("Permit");
		if (isDeny()) System.out.println("Deny");
		if (isIndeterminate()) System.out.println("Indeterminate");
		if (isNotApplicable()) System.out.println("NotApplicable");
	}
}
```

Figura 36. Código-fonte MyGsiPEP.

```
import java.io.File;
import java.io.FileInputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.ErrorHandler;
```

```
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import com.sun.xacml.AbstractPolicy;
import com.sun.xacml.EvaluationCtx;
import com.sun.xacml.MatchResult;
import com.sun.xacml.Policy;
import com.sun.xacml.PolicySet;
import com.sun.xacml.Target;
import com.sun.xacml.combine.DenyOverridesPolicyAlg;
import com.sun.xacml.finder.PolicyFinder;
import com.sun.xacml.finder.PolicyFinderModule;
import com.sun.xacml.finder.PolicyFinderResult;
import com.sun.xacml.finder.impl.FilePolicyModule;

/**
 * @author Joao Carlos S. do Vale
*/
public class MyGsiFinderModule extends PolicyFinderModule implements ErrorHandler {

        private PolicyFinder finder;
   private Set fileNames;
   private List policies;
   private static final Logger logger =
      Logger.getLogger(FilePolicyModule.class.getName());

   public static final String JAXP_SCHEMA_LANGUAGE =
      "http://java.sun.com/xml/jaxp/properties/schemaLanguage";

   public static final String POLICY_SCHEMA_PROPERTY =
      "com.sun.xacml.PolicySchema";

   public static final String W3C_XML_SCHEMA =
      "http://www.w3.org/2001/XMLSchema";

   public static final String JAXP_SCHEMA_SOURCE =
      "http://java.sun.com/xml/jaxp/properties/schemaSource";

   private File schemaFile;

        public MyGsiFinderModule() {
      fileNames = new HashSet();
      policies = new ArrayList();

      String schemaName = System.getProperty(POLICY_SCHEMA_PROPERTY);

      if (schemaName == null)
         schemaFile = null;
      else
         schemaFile = new File(schemaName);
          }

   public boolean addPolicy(String filename) {
      return fileNames.add(filename);
   }

   public boolean isRequestSupported() {
      return true;
   }

   public boolean isIdReferenceSupported() {
      return true;
   }

   public static AbstractPolicy loadPolicy(String filename,
       PolicyFinder finder,
```

```
        File schemaFile,
        ErrorHandler handler) {
           try {
                        // create the factory
                        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                        factory.setIgnoringComments(true);
                        DocumentBuilder db = null;

                        // as of 1.2, we always are namespace aware
                        factory.setNamespaceAware(true);

                        // set the factory to work the way the system requires
                        if (schemaFile == null) {
                                // we're not doing any validation
                                factory.setValidating(false);
                                db = factory.newDocumentBuilder();
                        } else {
                                // we're using a validating parser
                                factory.setValidating(true);
                                factory.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
                                factory.setAttribute(JAXP_SCHEMA_SOURCE, schemaFile);
                                db = factory.newDocumentBuilder();
                                db.setErrorHandler(handler);
                        }

                        // try to load the policy file
                        Document doc = db.parse(new FileInputStream(filename));

                        // handle the policy, if it's a known type
                        Element root = doc.getDocumentElement();
                        String name = root.getTagName();

                        if (name.equals("Policy")) {
                                return Policy.getInstance(root);
                        } else if (name.equals("PolicySet")) {
                                return PolicySet.getInstance(root, finder);
                        } else {
                                // this isn't a root type that we know how to handle
                                throw new Exception("Unknown root document type: " + name);
                        }
           } catch (Exception e) {
                        if (logger.isLoggable(Level.WARNING))
                                logger.log(Level.WARNING, "Error reading policy from file " +
                                                filename, e);
           }
           // a default fall-through in the case of an error
           return null;
}


public static AbstractPolicy loadPolicy(String filename,
        PolicyFinder finder) {
           return loadPolicy(filename, finder, null, null);
}

public void init(PolicyFinder finder) {
   this.finder = finder;

   Iterator it = fileNames.iterator();
   while (it.hasNext()) {
      String fname = (String)(it.next());
      AbstractPolicy policy = loadPolicy(fname, finder,
                        schemaFile, this);
      if (policy != null)
         policies.add(policy);
   }
```

```
        }

    public PolicyFinderResult findPolicy(EvaluationCtx context){
        PolicySet policySet;
        try {
                        policySet = new PolicySet(new URI("MyAC"),new DenyOverridesPolicyAlg(),new
Target(null,null,null),policies);
            MatchResult match = policySet.match(context);
            int result = match.getResult();
      if (result == MatchResult.INDETERMINATE)
        return new PolicyFinderResult(match.getStatus());
                        else
                                return new PolicyFinderResult(policySet);
                } catch (URISyntaxException e) {
                        e.printStackTrace();
                        return null;
                }
    }

    public void error(SAXParseException exception) throws SAXException {
      if (logger.isLoggable(Level.WARNING))
        logger.warning("Error on line " + exception.getLineNumber() +
                ": " + exception.getMessage() + " ... " +
                "Policy will not be available");

      throw new SAXException("error parsing policy");
    }

    public void fatalError(SAXParseException exception) throws SAXException {
      if (logger.isLoggable(Level.WARNING))
        logger.warning("Fatal error on line " + exception.getLineNumber() +
                ": " + exception.getMessage() + " ... " +
                "Policy will not be available");

      throw new SAXException("fatal error parsing policy");
    }

    public void warning(SAXParseException exception) throws SAXException {
      if (logger.isLoggable(Level.WARNING))
        logger.warning("Warning on line " + exception.getLineNumber() +
                ": " + exception.getMessage());
    }

}
```

Figura 37. Código-fonte MyGsiFinderModule.

C2 – Código-Fonte de MyDel

Este anexo apresenta o código-fonte do módulo MyDel, representado pelo objeto MyCertChain (Figura 38).

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.Security;
import java.security.SignatureException;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateExpiredException;
import java.security.cert.CertificateFactory;
import java.security.cert.CertificateNotYetValidException;
import java.security.cert.CertificateParsingException;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateCrtKey;
import java.util.Date;
import java.util.Enumeration;

import org.bouncycastle.asn1.x509.BasicConstraints;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.jce.PrincipalUtil;
import org.bouncycastle.jce.X509Principal;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.x509.X509V1CertificateGenerator;
import org.bouncycastle.x509.X509V3CertificateGenerator;
import org.bouncycastle.x509.extension.AuthorityKeyIdentifierStructure;
import org.bouncycastle.x509.extension.SubjectKeyIdentifierStructure;

/**
 * @author joaocarlos@atlantico
*/
public class MyCertChain {

	private static String jksPathMaster, aliasMaster, aliasSlave, jksPasswordMaster, privateCertPassword;
	private static String passKeysPai ,	passCertPai ,	passCertFilho, yesOrNo;
	public static PrivateKey getPrivateKey(String jksFileName, String jksAlias, String jksPassword, String
privatePassword)
	{
		InputStream jksInputStream = null;
		try {
			jksInputStream = new FileInputStream(jksFileName);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			System.exit(1);
		}
		KeyStore jksKeyStore = null;
		try {
			jksKeyStore = KeyStore.getInstance("JKS", "SUN");
			try {
				jksKeyStore.load(jksInputStream, jksPassword.toCharArray());
			} catch (NoSuchAlgorithmException e1) {
				e1.printStackTrace();
			} catch (CertificateException e1) {
				e1.printStackTrace();
			} catch (IOException e1) {
				e1.printStackTrace();
			}
		} catch (KeyStoreException e) {
			e.printStackTrace();
			System.exit(1);
```

```
		} catch (NoSuchProviderException e) {
			e.printStackTrace();
			System.exit(1);
		}
	RSAPrivateCrtKey jksPrivateCrtKey = null;
	try {
		jksPrivateCrtKey = (RSAPrivateCrtKey) jksKeyStore.getKey(jksAlias, privatePassword.toCharArray());
		return jksPrivateCrtKey;
	} catch (KeyStoreException e) {
		e.printStackTrace();
		System.exit(1);
	} catch (NoSuchAlgorithmException e) {
		e.printStackTrace();
		System.exit(1);
	} catch (UnrecoverableKeyException e) {
		e.printStackTrace();
		System.exit(1);
	}
	return null;
	}

	public static void addToKeyStore(String keystoreFileName, char[] keystorePassword,
			String alias, Certificate cert) {
		try {
			// Create an empty keystore object
			File keystoreFile = new File(keystoreFileName);
			KeyStore keystore = KeyStore.getInstance(KeyStore.getDefaultType());

			// Load the keystore contents
			FileInputStream in = new FileInputStream(keystoreFile);
			keystore.load(in, keystorePassword);
			in.close();

			// Add the certificate
			keystore.setCertificateEntry(alias, cert);

			// Save the new keystore contents
			FileOutputStream out = new FileOutputStream(keystoreFile);
			keystore.store(out, keystorePassword);
			out.close();
		} catch (java.security.cert.CertificateException e) {
			e.printStackTrace();
			System.exit(1);
		} catch (NoSuchAlgorithmException e) {
			e.printStackTrace();
			System.exit(1);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			System.exit(1);
		} catch (KeyStoreException e) {
			e.printStackTrace();
			System.exit(1);
		} catch (IOException e) {
			e.printStackTrace();
			System.exit(1);
		}
	}
	public static void jksTojks(String jksOriginFileName, String jksDestinyFilename,
					String jksPasswordOrigin, String jksPasswordDestiny)
	{
		try {
			File file = new File(jksOriginFileName);
			FileInputStream isOrigin = new FileInputStream(file);
			KeyStore keystore = KeyStore.getInstance(KeyStore.getDefaultType());
			keystore.load(isOrigin, jksPasswordOrigin.toCharArray());
			Enumeration enum = keystore.aliases();
```

```
                        for (; enum.hasMoreElements(); ) {
                                String alias = (String)enum.nextElement();
                                System.out.print("Do you want add <" + alias + "> certificate to new keystore
(y/n)?[yes] ");
                                String answer = yesOrNo;
                                if (!(answer.compareTo("N") == 0) && !(answer.compareTo("n") == 0)) {
                                        Certificate cert = keystore.getCertificate(alias);

        addToKeyStore(jksDestinyFilename,jksPasswordDestiny.toCharArray(),alias,cert);
                                }
                        }
                        isOrigin.close();
                } catch (java.security.cert.CertificateException e) {
                        e.printStackTrace();
                        System.exit(1);
                } catch (NoSuchAlgorithmException e) {
                        e.printStackTrace();
                        System.exit(1);
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                        System.exit(1);
                } catch (KeyStoreException e) {
                        e.printStackTrace();
                        System.exit(1);
                } catch (IOException e) {
                        e.printStackTrace();
                        System.exit(1);
                }
        }

        public static Certificate createMasterCert(X509Certificate x509certificate)
        {
                v1CertGen.setSerialNumber(x509certificate.getSerialNumber());
    try {
                        v1CertGen.setIssuerDN(PrincipalUtil.getSubjectX509Principal(x509certificate));
                } catch (CertificateEncodingException e) {
                        e.printStackTrace();
                }
    v1CertGen.setNotBefore(new Date(System.currentTimeMillis() - 0x9a7ec800L));
    v1CertGen.setNotAfter(new Date(System.currentTimeMillis() + 0x9a7ec800L));
    v1CertGen.setSubjectDN(new X509Principal(x509certificate.getSubjectDN().toString()));
    v1CertGen.setPublicKey(x509certificate.getPublicKey());
    v1CertGen.setSignatureAlgorithm(x509certificate.getSigAlgName());
    X509Certificate x509certificate1 = null;
                try {
                        x509certificate1 =
v1CertGen.generateX509Certificate(getPrivateKey(jksPathMaster,aliasMaster,jksPasswordMaster, privateCertPassword ));
                        System.out.println("Private key captured...");
                } catch (InvalidKeyException e1) {
                        e1.printStackTrace();
                } catch (SecurityException e1) {
                        e1.printStackTrace();
                } catch (SignatureException e1) {
                        e1.printStackTrace();
                }
                try {
                        x509certificate1.checkValidity(new Date());
                } catch (CertificateExpiredException e2) {
                        e2.printStackTrace();
                } catch (CertificateNotYetValidException e2) {
                        e2.printStackTrace();
                }
    try {
                        x509certificate1.verify(x509certificate.getPublicKey());
                } catch (InvalidKeyException e3) {
                        e3.printStackTrace();
```

```
                } catch (CertificateException e3) {
                        e3.printStackTrace();
                } catch (NoSuchAlgorithmException e3) {
                        e3.printStackTrace();
                } catch (NoSuchProviderException e3) {
                        e3.printStackTrace();
                } catch (SignatureException e3) {
                        e3.printStackTrace();
                }
        return x509certificate;
        }

        public static Certificate createDelegatedCert(X509Certificate x509certificateSuper,
                        X509Certificate aCertificate)
        {
                v3CertGen.reset();
                v3CertGen.setSerialNumber(aCertificate.getSerialNumber());
                try {
                        v3CertGen.setIssuerDN(PrincipalUtil.getSubjectX509Principal(x509certificateSuper));
                } catch (CertificateEncodingException e) {
                        e.printStackTrace();
                }
                v3CertGen.setNotBefore(new Date(System.currentTimeMillis() - 0x9a7ec800L));
                v3CertGen.setNotAfter(new Date(System.currentTimeMillis() + 0x9a7ec800L));
                v3CertGen.setSubjectDN(new X509Principal(aCertificate.getSubjectDN().toString()));
                v3CertGen.setPublicKey(kpair.getPublic());
                v3CertGen.setSignatureAlgorithm(aCertificate.getSigAlgName());
                try {
                        v3CertGen.addExtension(X509Extensions.SubjectKeyIdentifier, false, new
SubjectKeyIdentifierStructure(aCertificate.getPublicKey()));
                } catch (CertificateParsingException e1) {
                        e1.printStackTrace();
                }
                try {
                        v3CertGen.addExtension(X509Extensions.AuthorityKeyIdentifier, false, new
AuthorityKeyIdentifierStructure(x509certificateSuper));
                } catch (CertificateParsingException e2) {
                        e2.printStackTrace();
                }
                v3CertGen.addExtension(X509Extensions.BasicConstraints, true, new BasicConstraints(0));
                X509Certificate x509certificate1 = null;
                try {
                        x509certificate1 =
v3CertGen.generateX509Certificate(getPrivateKey(jksPathMaster,aliasMaster,jksPasswordMaster, privateCertPassword ));
                } catch (InvalidKeyException e3) {
                        e3.printStackTrace();
                } catch (SecurityException e3) {
                        e3.printStackTrace();
                } catch (SignatureException e3) {
                        e3.printStackTrace();
                }
                try {
                        x509certificate1.checkValidity(new Date());
                } catch (CertificateExpiredException e4) {
                        e4.printStackTrace();
                } catch (CertificateNotYetValidException e4) {
                        e4.printStackTrace();
                }
                try {
                        x509certificate1.verify(x509certificateSuper.getPublicKey());
                } catch (InvalidKeyException e5) {
                        e5.printStackTrace();
                } catch (CertificateException e5) {
                        e5.printStackTrace();
                } catch (NoSuchAlgorithmException e5) {
                        e5.printStackTrace();
```

```
		} catch (NoSuchProviderException e5) {
			e5.printStackTrace();
		} catch (SignatureException e5) {
			e5.printStackTrace();
		}
		return x509certificate1;
	}


	/*
	 * Par?metros:
	 * args[0]: keystore
	 * args[1]: alias certificado
	 * args[2]: path certificado filho
	 * args[3]: alias filho
	 */
	public static void main(String args[]) throws Exception
	{
		jksPathMaster = args[0];
		aliasMaster = args[1];
		aliasSlave = args[3];
		passKeysPai = args[4];
		passCertPai = args[5];
		passCertFilho = args[6];
		yesOrNo		= args[7];


		Security.addProvider(new BouncyCastleProvider());
		if (Security.getProvider("BC") == null) {
			System.out.println("Can't find BC - adding");
			Security.addProvider(new BouncyCastleProvider());
		}
		else{
			System.out.println("BC is present");
		}

		KeyFactory keyfactory = KeyFactory.getInstance("RSA", "BC");
		Certificate acertificate[] = new Certificate[2];

		InputStream jksInputStream = null;
		try {
			jksInputStream = new FileInputStream(jksPathMaster);
			System.out.println("Establish JKS InputStream to " + jksPathMaster);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			System.exit(1);
		}

		KeyStore jksKeyStore = null;
		try {
			System.out.print(args[0] + " password: ");
			//jksPasswordMaster = Util.readStr();
			jksPasswordMaster = passKeysPai;
			jksKeyStore = KeyStore.getInstance("JKS", "SUN");
			jksKeyStore.load(jksInputStream, jksPasswordMaster.toCharArray());
			System.out.println("JKS KeyStore Object Loaded.");
			System.out.print(aliasMaster + "password = ");
			//privateCertPassword = Util.readStr();
			privateCertPassword = passCertPai;
		} catch (KeyStoreException e) {
			e.printStackTrace();
			System.exit(1);
		} catch (NoSuchProviderException e) {
			e.printStackTrace();
			System.exit(1);
		}
```

```
                acertificate[1] = createMasterCert((X509Certificate)jksKeyStore.getCertificate(aliasMaster));

                InputStream jksInputStream1 = null;
                CertificateFactory cf = CertificateFactory.getInstance("X.509");
                try {
                        jksInputStream1 = new FileInputStream(args[2]);
                        System.out.println("Getting " + args[2]);
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                        System.exit(1);
                }
                Certificate cert = cf.generateCertificate(jksInputStream1);

                KeyPairGenerator kpGen = KeyPairGenerator.getInstance("RSA");
                kpGen.initialize(1024);
                kpair = kpGen.generateKeyPair();

                acertificate[0] = createDelegatedCert((X509Certificate)jksKeyStore.getCertificate(aliasMaster),
                                (X509Certificate)cert);

                System.out.print("Temporary password: ");
                String passwdTmp = passCertFilho;

                KeyStore keystore = KeyStore.getInstance("JKS", "SUN");
                keystore.load(null, null);
                keystore.setKeyEntry(aliasSlave, kpair.getPrivate(), passwdTmp.toCharArray(), acertificate);
                FileOutputStream fileoutputstream = new FileOutputStream(aliasSlave+".jks");


                keystore.store(fileoutputstream, passwdTmp.toCharArray());
                System.out.println("Certificate chain created: " + aliasSlave + ".jks");

                jksTojks(jksPathMaster,aliasSlave+".jks",jksPasswordMaster,passwdTmp);
                System.out.println("Certificates generated!");
        }

        static X509V1CertificateGenerator v1CertGen = new X509V1CertificateGenerator();
        static X509V3CertificateGenerator v3CertGen = new X509V3CertificateGenerator();
        static KeyPair kpair = null;
}
```

Figura 38. Código-fonte MyCertChain.