UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Marcelo Digiacomo Chryssovergis

Uma Interface para Refinamento de Pesquisas de Políticas de Segurança em Ambientes de Grid Services

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Carla Merkle Westphall, Dra.
Professora Orientadora

Florianópolis, Agosto de 2005

Uma Interface para Refinamento de Pesquisas de Políticas de Segurança em Ambientes de Grid Services

Marcelo Digiacomo Chryssovergis

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração em Sistemas de Computação e aprovada em sua forma final pelo programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Raul Sidnei Wazlawick, Dr.
Coordenador do programa de pós-graduação
em Ciência da Computação.

Banca examinadora:

_____

Prof. Dra. Carla Merkle Westphall.
Orientadora

_____

Prof. Dr. Arthur Ronald de Vallauris Buchsbaum

_____

Prof. Dr. Mário Antonio Ribeiro Dantas

_____

Prof. Dr. Carlos Barros Montez

Sumário

“O progresso é feito respondendo-se
perguntas. Descobertas são feitas
questionando-se respostas.”
Bernhard Haisch

Agradecimentos

Agradeço à minha orientadora, Dra. Carla Merkle Westphall, por ter me acompanhado neste longo e difícil caminho, e aos meus amigos e colegas do LRG pela paciência, incentivo e companheirismo.

Dedicatória

Dedico este trabalho aos meus pais, Antoine e Rosemary, à minha irmã, Fabiana, e à minha noiva, Ana Cláudia, que participaram ativamente desta conquista.

Resumo

A computação em grid (ou computação em grade) consiste em uma forma de computação distribuída onde o foco principal é o compartilhamento coordenado de recursos em larga escala e resolução de problemas em organizações virtuais dinâmicas e multi-institucionais. Porém, tal compartilhamento deve ser altamente controlado, para garantir a segurança dos recursos envolvidos. Este trabalho se baseia na especificação OGSA (*Open Grid Service Architecture*) proposta pelo GGF (*Global Grid Forum*), em particular no Globus Toolkit 3 que a implementa, e apresenta uma proposta de extensão ao módulo de monitoramento e descoberta de recursos (MDS), para filtrar os resultados retornados baseado nos atributos do usuário e nas políticas do recurso.

Abstract

Grid computing consists in a distributed computing form where the main focus is the coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. However, this sharing must be highly controlled, to ensure the security of involved resources. This work has its foundation in OGSA specification (Open Grid Service Architecture) proposed by GGF (Global Grid Forum), and particularly in Globus Toolkit 3 that implements it, and presents an extension proposal to the Monitoring and Discovery Service of GT3, to filter the obtained results based on user attributes and resource policies.

Lista de Abreviaturas

Abreviatura	**Significado**
CA……..	Certificate Authority
CAS……	Community Authorization Service
ClassAd……..	Classified Advertisement
GGF……...	Global Grid Forum
GIIS……	Grid Index Information Service
GRAM……...	Globus Resource Allocation Manager
GRIS……..	Grid Resource Information Service
GSH……...	Grid Service Handle
GSI…….	Globus Security Infrastructure
GSR……	Grid Service Reference
GSS-API……	Generic Security Service Application Programming Interface
GT3……	Globus Toolkit 3
KDC……...	Key Distribution Center
LDAP…….	Lightweight Directory Access Protocol
LOID……..	Legion Object Identificator
MDS……...	Monitoring and Discovery Service
MJFS……..	Managed Job Factory Service
MJS……	Managed Job Service
MMJFS……..	Master Managed Job Factory Service
MPI……	Message Passing Interface
NAT……...	Network Address Translation
OGSA…….	Open Grid Service Architecture
PACI……...	Partnership for Advanced Computational Infrastructure
PVM……...	Paralell Virtual Machine
QoP…….	Quality of Protection
RIPS……...	Resource Information Provider Service
SDE.....	Service Data Element
SOAP…….	Simple Object Access Protocol
SSL…….	Secure Socket Layer
TCP……	Transmission Control Protocol
TLS……	Transport Layer Security
UDDI…….	Universal Description, Discovery and Integration
UHE……...	User Host Environment
URI…….	Uniform Resource Identifier
VO……..	Virtual Organization
W3C……...	World Wide Web Consortium
XML……...	Extensible Markup Language

Capítulo 1

Introdução

A computação em *grid* é considerada um grande passo da evolução dos sistemas distribuídos. Organizações Virtuais (VO) são responsáveis por reunir, em um mesmo contexto, parceiros de negócios ou de pesquisas com interesses em comum, mesmo estando geograficamente distantes. A distância física pode também ser utilizada a favor das organizações virtuais: participantes de uma VO no Brasil, com intenção de iniciar às 10:00hs da manhã uma atividade computacional que precisa de muito processamento, podem utilizar os ciclos ociosos de computadores situados no Japão, já que grande parte dos computadores japoneses estarão ociosos às 22:00hs.

A computação distribuída em larga escala exige que a infraestrutura suporte plataformas das mais variadas, tanto de hardware como de software. Os sistemas *grid* possuem esta natureza heterogênea e, portanto, um requisito fundamental é a interoperabilidade entre os integrantes da VO. Para alcançar este requisito surgiu a arquitetura OGSA (*Open Grid Service Architecture*) que se baseia em padrões XML (*Extensible Markup Language*) definidos pela W3C (*World Wide Web Consortium*). Também baseada em XML, a tecnologia de *Web Services* (ou Serviços Web) possui muito em comum com o *grid*. Algumas das características dos *web services* que também se

aplicam ao *grid* são: suporte a descoberta dinâmica de serviços, registro e descoberta de definições de serviços e criação dinâmica de *proxies* (elementos responsáveis pelo recebimento de requisições ou invocação de serviços externos). Assim surgiu o conceito de *grid services*, que além de compartilhar estas características com os *web services* suportam invocação segura, gerência de tempo de vida, notificação, gerência de política, gerência de credenciais, virtualização de recursos e criação de instâncias de serviços transientes [FOSTER 2002].

O Globus Toolkit 3 é uma plataforma que além de seguir os preceitos do OGSA, é também um *middleware* que possibilita a integração entre outras plataformas de *grid*. Segundo o *site* oficial do Globus, esta é uma tecnologia que permite que as pessoas compartilhem poder computacional, bancos de dados e outras ferramentas de forma segura, cruzando limites corporativos, institucionais e geográficos, sem comprometer a autonomia local. Por este motivo o Globus é utilizado neste trabalho para demonstrar o funcionamento da interface proposta.

1.1 Objetivo

O objetivo deste trabalho é a melhoria do serviço de descobertas de recursos em um ambiente de *grid*, através de um mecanismo de refinamento baseado nos atributos do usuário (no lado cliente) e na política de restrições aplicadas aos recursos (no lado servidor).

1.2 Justificativa

O Globus MDS (Sistema de Monitoramento e Descoberta do Globus) é uma implementação do Serviço de Informação do Grid (GIS). O MDS funciona como um repositório onde serão buscadas informações sobre serviços, baseadas em um nome distinto. Vários parâmetros podem ser submetidos em uma busca, como um sistema operacional ou poder computacional mínimo desejado. O MDS pode também funcionar como um diretório de páginas amarelas quando o objetivo é a busca de entidades

categorizadas. Combinando estas duas funcionalidades com uma extensão que possibilita pesquisas mais avançadas através de filtros *booleanos*, tem-se a mão uma poderosa ferramenta de busca.

Apesar do poder desta ferramenta, uma pesquisa pode retornar uma grande quantidade de resultados, dentre os quais o usuário pode não ter permissão para acessar vários dos recursos. O módulo de extensão que este trabalho propõe restringe a busca de forma a retornarem apenas resultados válidos, ou seja, todos os recursos encontrados poderão ser acessados pelo usuário.

1.3 Organização do Trabalho

No capítulo 1 é apresentada uma introdução, objetivo e justificativa desta dissertação. No capítulo 2, são relembrados conceitos de segurança computacional, incluindo criptografia, políticas de segurança e outros conceitos relevantes. No capítulo 3 é abordado o assunto de computação em *grid*, organizações virtuais, suas vantagens, arquitetura do *grid*, entre outros conceitos. No capítulo 4, é abordada a questão da segurança em *grid*, especialmente no *globus.* O capítulo 5 apresenta os trabalhos relacionados. O capítulo 6 apresenta o modelo proposto nesta dissertação, e sua relação com os trabalhos citados no capítulo 5. Para finalizar, o capítulo 7 apresenta a conclusão desta dissertação.

Capítulo 2

Segurança Computacional

O conceito de segurança de computadores se alterou com o passar do tempo. Antigamente os computadores eram tão grandes e existiam tão poucos, que para protegê-los bastava restringir o acesso físico a eles [LANDWEHR 2001]. Hoje, é praticamente impossível encontrarmos um computador isolado de uma rede. Dados bancários, documentos confidenciais ou quaisquer outros tipos de informação que se deseje proteger, trafegam na internet todos os dias. Este capítulo aborda alguns tipos de ataques que estes dados podem sofrer, as principais propriedades e mecanismos de segurança computacional e explica com mais detalhes um destes mecanismos de segurança: a criptografia.

2.1 Conceito

De acordo com [LANDWEHR 2001], um sistema computacional está seguro se está livre de ameaças e de preocupações. Em alguns casos, as preocupações equivalem à proteção de dados contra leitura não permitida (por exemplo, na transmissão de um número de cartão de crédito pela internet), e em outros casos, deve-se apenas garantir a disponibilidade de um serviço e a integridade dos dados enquanto são enviados.

No começo da era computacional os equipamentos eram extremamente caros, mas mesmo nesta época os dados sempre receberam a maior atenção. Hoje, na era da informação, é difícil imaginar um computador isolado de uma rede e, portanto, a segurança dos dados deve receber ainda maior atenção.

Antes de abordar os assuntos deste capítulo, é importante revisar alguns conceitos relacionados à segurança computacional. Abaixo, seguem os conceitos de vulnerabilidade, ameaça e risco, segundo [LANDWEHR 2001]:

- Vulnerabilidade: uma falha em um software que é executado em modo privilegiado, uma falha no sistema operacional, uma senha mal configurada, ou uma regra mal delimitada em um *firewall* caracterizam vulnerabilidades em um sistema;

- ameaça: é a intenção de se causar danos em um sistema. Uma ameaça pode se concretizar através de um ataque, onde o autor tira proveito de uma vulnerabilidade encontrada no sistema. Em muitos casos, a principal ameaça parte dos próprios usuários, que possuem acesso legítimo ao sistema;

- risco: é a probabilidade de um sistema não ser capaz de impor sua política de segurança frente a ataques. Portanto, o risco é uma função da exposição das vulnerabilidades de um sistema no contexto da sua arquitetura de segurança e do nível de ameaça manifestada contra o sistema em um determinado momento.

2.2 Políticas de Segurança

A definição de política de segurança, proposta por [WANG 2004] é a especificação do comportamento de um sistema, no que é relevante à segurança. Por exemplo, uma política de controle de acesso define quem pode (ou não) acessar determinados recursos, e em que condições. Outros tipos de políticas podem especificar sob quais condições as credenciais de um usuário serão aceitas, ou como o *firewall* de uma organização é configurado. Portanto, podemos conceituar políticas de segurança como conjuntos de regras estabelecidas para minimizar riscos de segurança, e planos para lidar com estes riscos.

Na especificação de uma política de segurança, deve-se fazer uma análise da organização para a qual esta política será aplicada, respondendo algumas questões:

- Quais os bens a serem protegidos, e o valor deles;
- Quais tipos de ameaças que estes bens estão sofrendo; e
- Quais ameaças devem (e podem) ser eliminadas, por que meios e a que custo.

No *grid*, cada instituição dentro de uma VO pode querer definir sua própria política institucional. Porém, esta política deve ser consistente com a política da organização virtual, que por sua vez deve ser consistente com a política de cada recurso [PEARLMAN 2002].

2.3 Administração de Riscos

A administração de riscos é uma disciplina que visa analisar qualitativamente e quantitativamente os riscos de perdas e vazamentos de informações de uma instituição [BLAKLEY 2001]. As medidas de segurança tomadas para conter um ataque devem ser comensuradas com a ameaça em si [MOFFETT 1995]. Fazendo uma analogia com o mundo físico, se quisermos proteger uma residência podemos tomar providências como a instalação de um alarme, comprar uma fechadura mais robusta e um cachorro treinado. Estas seriam medidas simples a serem tomadas, porém apresenta algumas mudanças no cotidiano dos moradores:

- O alarme pode ser de difícil manuseio, exigindo que os moradores compreendam seu mecanismo para não ativar o serviço de vigilância sem motivo (aumento da complexidade);
- A fechadura apresenta mais um obstáculo que separa o morador e o interior da casa. Ao invés de abrir uma fechadura, agora se deve abrir duas, ou mais (redução do desempenho); e

- Um cachorro exige manutenção, cuidados veterinários e compra de comida (aumento do custo).

Apesar de apresentar alguns inconvenientes, o aumento da segurança é considerado necessário. Porém, se o consultor de segurança oferecesse ao dono da casa um equipamento de circuito fechado, células fotoelétricas, detector de presença e uma equipe de seguranças particulares, muito provavelmente a oferta seria negada, pois o custo da implantação e manutenção inviabilizaria o projeto, levando-se em consideração que se trata de uma residência, e não de um banco.

A análise de riscos é uma atividade que examina o valor dos bens de uma organização e os riscos que eles enfrentam, para então tomar decisões sobre a proteção que eles necessitam. Para compensar a implantação das medidas de segurança, seu custo deverá ser menor que o valor daquele conjunto de dados. Entre várias metodologias relacionadas à análise de riscos, algumas tarefas em comum são: identificação e avaliação dos bens, construção de cenários de riscos de segurança, avaliação da probabilidade da ocorrência destes cenários, e identificação e custos de medidas de segurança possíveis para cada cenário [LANDWEHR 2001].

2.4 Propriedades de Segurança

Para cumprir os objetivos de uma política de segurança, algumas propriedades de segurança são necessárias [LANDWEHR 2001]:

- Confidencialidade: é a proteção contra a leitura não autorizada de informações. A confidencialidade se diferencia da privacidade, pois um conjunto de informações pode ser compartilhado por um grupo de entidades, enquanto que a privacidade é algo que apenas o dono deve ter acesso;

- Integridade: a verificação de que um conjunto de dados não foi alterado sem autorização;

- Disponibilidade: é a garantia de que as informações estarão sempre disponíveis aos usuários válidos do sistema;

- Autenticação: é um serviço de segurança essencial que verifica a identidade de uma entidade logo ao entrar no sistema. A autenticação é um pré-requisito para as três propriedades citadas acima, já que para impô-las é necessário que o sistema conheça a identidade do usuário;

- Não-repúdio: é a garantia de que uma terceira entidade, neutra, possa afirmar a ocorrência (ou não) de uma determinada transação.

2.5 Tipos de Ataques

As redes de computadores, incluindo a internet, possibilitam a troca de informações entre usuários do mundo inteiro. Porém, ao trafegar em uma rede pública, os dados podem sofrer alguns tipos de ataques. Dentre os ataques, alguns são classificados como passivos (onde ocorre apenas leitura dos dados), e outros como ativos (onde ocorre alteração ou destruição dos dados ou impedimento de utilização de um serviço). Segue abaixo um resumo sobre alguns tipos de ataques [STALLINGS 1999]:

- Interrupção: este é um ataque que suspende a disponibilidade dos dados, podendo ocorrer através da destruição física do equipamento ou da linha de transmissão, ou desabilitando o sistema de gerenciamento de arquivos (figura 2.1);

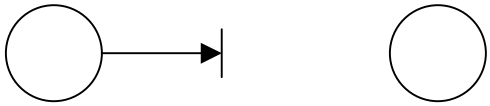

Figura 2.1 - Ataque de Interrupção

- Intercepção: consiste em um atacante (entidade não-autorizada no sistema) visualizando informações sem autorização. Este é um ataque contra a confidencialidade dos dados (figura 2.2);

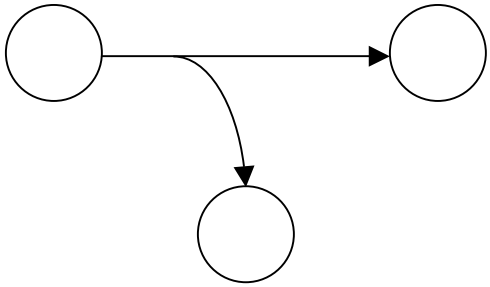

Figura 2.2 – Ataque de Intercepção

- Modificação: uma entidade não autorizada que além de interceptar informações não pertinentes a ela, também altera estas informações. Este é um ataque contra a integridade dos dados (figura 2.3);

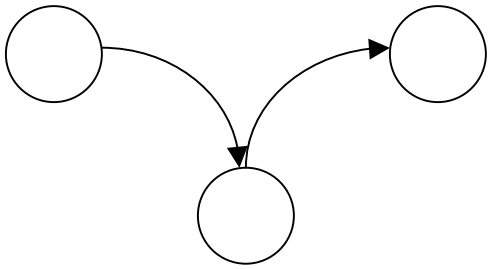

Figura 2.3 – Ataque de Modificação

- Personificação: consiste na geração de dados ilegítimos por uma entidade se passando por uma outra entidade, causando um ataque contra a autenticidade (figura 2.4)

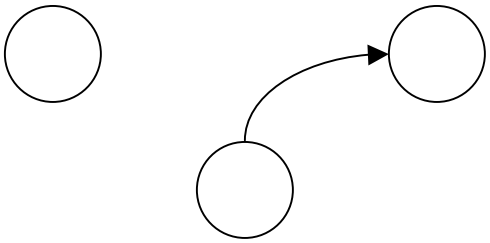

Figura 2.4 – Ataque de Personificação

2.6 Mecanismos de Segurança

Na seção anterior foram ilustrados alguns tipos de ataques que podem ocorrer em uma rede. O objetivo desta seção é explicar três dos principais mecanismos de segurança existentes, e exemplificar casos de uso.

2.6.1 Controle de acesso

Qualquer sistema de gerência de informações necessita impor um importante requisito: que todos os acessos ao sistema e aos seus recursos devem ser controlados e que apenas acessos autorizados devem ocorrer.

Definir uma política de controle de acesso não é uma tarefa trivial. O conjunto de enunciados que delimitam uma política deve ser muito bem especificado para não haver problemas de ambigüidade na sua interpretação. Três classes principais de políticas de controle de acesso são listadas abaixo: [SAMARATI 2000].

- Discricionária (*Discritionary Access Control - DAC*): esta política se baseia na identidade do solicitante e nas regras de acesso que declaram o que ele pode ou não fazer. Um sistema que utiliza esta política permite que um usuário delegue um ou vários dos seus privilégios para outro usuário;
- Mandatária (*Mandatory Access Control - MAC*): esta política se baseia em regras obrigatórias determinadas por uma autoridade central. A forma mais comum é a política de segurança multinível, que se baseia na classificação de sujeitos e objetos no sistema. Sujeitos são entidades ativas que agem em benefício dos usuários, e objetos são arquivos, diretórios ou bancos de dados; e
- Baseada em papéis (*Role-based Access Control - RBAC*): O modelo RBAC define um conjunto de elementos básicos: usuários, papéis, operações, permissões e objetos, podendo haver relacionamento de muitos para muitos entre os elementos

(um usuário pode exercer vários papéis, e um papel pode ser atribuído a vários usuários) [ANSI/INCITS 2004].

2.6.2 Autenticação

A autenticação é um processo utilizado para estabelecer confiança entre duas entidades, baseando-se na identificação de cada entidade, seguida de verificação. Uma entidade que pode ser identificada unicamente em um sistema distribuído é chamada de *principal*.

Existem três tipos principais de autenticação [WOO 1996]:

- Autenticação do conteúdo da mensagem – verifica que o conteúdo da mensagem recebida é o mesmo de quando foi enviada;
- Autenticação da origem da mensagem – verifica que o emissor da mensagem recebida é o mesmo que está gravado no campo “remetente” da mensagem; e
- Autenticação da identidade de uma entidade – verifica que o principal realmente é quem alega ser.

O objetivo desta seção é fundamentar o último tipo de autenticação citado. Existem vários mecanismos para autenticar um usuário que deseja acessar o sistema, e dentre eles serão citados alguns dos mais conhecidos.

Alguns mecanismos de autenticação se baseiam em algo que o usuário sabe (baseados em senha), outros se baseiam em algo que o usuário possui (*tokens*, *smartcard*), e outros se baseiam em características físicas do usuário (biometria).

A autenticação por senha utiliza o protocolo desafio-resposta, iniciado quando o usuário envia sua identificação para o sistema. Em seguida o sistema envia para o usuário um número aleatório, que servirá como identificador da sessão, além de uma função *hash* e de uma função desafio. O usuário encaminha sua resposta, que compreende o resultado da função envolvendo o *hash* que foi aplicado à senha enviada, e o número aleatório recebido. Se o resultado for equivalente ao resultado da função com o número aleatório e a função *hash* da senha real do usuário, a autenticação é concedida.

Na autenticação por *token*, o dispositivo pode conter o código armazenado em sua memória (depois de aplicar uma função *hash* a ele), ou gerar códigos que são utilizados uma única vez. Os códigos utilizados são protegidos por uma senha que deve ser fornecida pelo usuário para efetuar a autenticação. Esta senha é processada no interior do *token*, não se sujeitando, portanto, a riscos de captura de senha no computador.

A autenticação biométrica se baseia em padrões encontrados em impressões digitais, íris, retina, face, mão, ou fala, que identificam com precisão um indivíduo. Um dispositivo biométrico captura dados de um usuário para obter um padrão. Este padrão é combinado com um número aleatório e, depois de criptografado, é utilizado para verificação de compatibilidade com o padrão que foi previamente armazenado no servidor.

2.6.3 Criptografia

A palavra “criptografia” é formada a partir de duas raízes gregas: *kryptos*, que significa “segredo”, e *graphos*, que significa “escrita”. Porém, os objetivos da criptografia atual vão além da escrita secreta. A criptografia é um meio utilizado para alcançar as propriedades de segurança citadas na seção 2.4: confidencialidade, integridade, não-repúdio, autenticação e disponibilidade.

Existem dois tipos de algoritmos criptográficos: simétrico (ou de chave secreta), e assimétrico (ou de chave pública), que serão descritos com mais detalhes abaixo:

2.6.3.1 Criptografia Simétrica

Os algoritmos simétricos utilizam a mesma chave tanto para cifrar como para decifrar os dados. Este método envolve um algoritmo, um texto plano e uma chave secreta. O resultado é um texto ininteligível que só poderá ser lido novamente através da aplicação do algoritmo para decifrá-lo, o que ocorre com a mesma chave secreta. A segurança deste método se baseia na qualidade e no tamanho da chave secreta utilizada, e não no conhecimento do algoritmo, já que muitas vezes é público.

O inconveniente em relação à criptografia simétrica, é que como esta utiliza a mesma chave para cifrar e decifrar os dados, é necessário que o fornecimento da chave secreta para o outro participante da comunicação seja efetuado pessoalmente, ou através de uma outra comunicação segura. Em muitos casos as organizações realmente trocam a chave desta forma, através de encontros ou através de um KDC (centro de distribuição de chaves). Porém, a criptografia de chave pública oferece um meio prático e seguro de se efetuar a troca de chaves secretas, como é descrito na próxima seção. Abaixo, na figura 2.5, é ilustrado o processo de aplicação desta técnica.

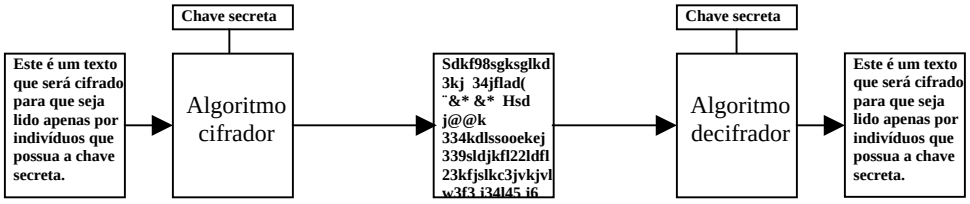

Figura 2.5 – Criptografia Simétrica

2.6.3.2 Criptografia Assimétrica

O surgimento da criptografia assimétrica foi um grande avanço para a segurança computacional. Ao contrário da criptografia convencional (com compartilhamento da mesma chave secreta entre todas as partes envolvidas), a criptografia assimétrica possui duas chaves – a chave pública e a chave privada.

Esta técnica utiliza funções matemáticas que criam duas chaves relacionadas, porém é impraticável que se deduza uma a partir da outra. Um dos motivos que impulsionou o surgimento desta técnica foi o problema da distribuição de chaves. Na criptografia assimétrica a chave pública de um usuário está ao alcance de todos, enquanto a chave privada deve ser conhecida apenas pelo seu dono. A seguir será explicado como prover alguns dos serviços de segurança através deste mecanismo:

- Confidencialidade: para enviar dados de forma segura da entidade “A” para “B”, “A” deve cifrar estes dados com a chave pública de “B”. Quando “B” recebe a mensagem, utiliza sua chave privada (que ninguém mais conhece) para decifrá-la;

- Autenticidade: quando uma entidade “A” envia dados para “B”, “A” pode assinar a mensagem (ou o resultado de uma função *hash* aplicada a ela) cifrando-a com sua chave privada. Em seguida, “B” decifra a mensagem utilizando a chave pública de “A”;

- Troca de chaves: como a criptografia de chave pública envolve grande *overhead* computacional, muitas vezes é utilizada apenas para estabelecer uma chave secreta entre duas partes, para cifrar e decifrar mensagens utilizando criptografia simétrica;

- Integridade: freqüentemente a criptografia de chaves públicas é utilizada em conjunto com uma função de hash que gera um resumo da mensagem enviada. Tal resumo é assinado com a chave privada do emissor, garantido tanto a autenticidade como a integridade da mensagem.

Na criptografia assimétrica, a chave pública de um usuário “A” deve estar ao alcance de todos, para que outros usuários verifiquem a assinatura digital de “A”, ou quando se deseja cifrar mensagens direcionadas a “A”.

A autoridade certificadora (CA) é uma entidade confiável que armazena certificados digitais dos usuários. Um certificado contém informações pessoais do dono e principalmente sua chave pública, e é assinado pela CA para garantir sua autenticidade e integridade.

2.7 Considerações Finais do Capítulo

A segurança no envio de mensagens é desejada desde muito antes do surgimento dos computadores. O imperador romano Júlio César enviava ordens secretas aos seus generais utilizando um algoritmo simples (Cifra de César) que consiste na substituição de uma letra pela terceira letra seguinte. Hoje, com a utilização da criptografia, a segurança da informação evoluiu muito.

A segurança computacional não consiste apenas em ocultar informações. A criptografia possibilita a verificação da integridade da mensagem (a mensagem que foi

recebida é idêntica à que foi enviada), a autenticidade (garante a identidade do remetente) e o não-repúdio (impede que se negue o envio ou recebimento de uma mensagem). Usufruir destas propriedades é, além de um direito, muitas vezes questão de segurança nacional.

Capítulo 3

Computação em Grid

A necessidade de grande poder computacional acompanha todo o histórico da informática. Sempre existem problemas que os computadores convencionais não são capazes de solucionar, sejam eles biológicos, médicos, meteorológicos ou de engenharia. Para tanto, surgiram os supercomputadores, máquinas de alto custo, específicas para aplicações científicas que requerem altíssimo desempenho.

A partir de 1980 começa a ser utilizada uma técnica que envolve agrupamento de vários computadores, resultando na emulação de um computador multiprocessado [FOSTER 2000]. Os chamados clusters são utilizados como uma alternativa eficiente e muito mais acessível do que os supercomputadores, sendo utilizados em muitos centros de supercomputação até hoje.

A computação distribuída tem por objetivo o compartilhamento de recursos computacionais. Alguns exemplos de infraestruturas de computação distribuída são: CORBA (*Common Object Request Broker Architecture*), RPC (*Remote Procedure Call*), RMI (*Remote Method Invocation*), DCOM (*Distributed Component Object Model*), agentes móveis, *peer to peer* e GRID [FOSTER 2001].

O *grid* fornece uma visão transparente do sistema, como se o conjunto de recursos (*hardware* e *software)* que o formam fosse um único e poderoso computador [IBM]. Além

disso, a computação em *grid* possibilita a integração de redes de diferentes instituições de forma dinâmica, para compartilhamento de ciclos de processador, espaço em disco, acesso a banco de dados, *software*, arquivos, ou qualquer outro tipo de recurso, como microscópios eletrônicos ou telescópios. O que motiva os pesquisadores é o potencial de tornar disponível um excelente poder computacional para os participantes do *grid*.

3.1 Definição

A estrutura das redes de distribuição de energia foi uma das tecnologias que fundamentaram a computação em *grid*. Apesar da complexidade da estrutura, para o usuário basta conectar um equipamento eletrônico em uma tomada para receber energia. Esta corrente elétrica possivelmente atravessou o estado, ou o país inteiro, proveniente de uma usina que utiliza água, vento, ou energia nuclear. O consumidor, entretanto, não tem interesse em saber como a eletricidade foi gerada, deseja apenas utilizá-la.

Fazendo uma analogia com a internet, toda a complexidade por trás da web fica oculta para os usuários. Quando se recorre a um *website* qualquer, não é necessário que se conheça a localização do servidor, ou o procedimento completo que seu navegador executou para encontrá-lo [IBM]. Esta é exatamente a proposta da computação em *grid* – oferecer recursos computacionais aos usuários do *grid*, sem que seja necessário conhecer detalhes como a localização do recurso ou o protocolo utilizado.

A computação em *grid* consiste no compartilhamento coordenado de recursos, e resolução de problemas em organizações virtuais multi-institucionais e dinâmicas [FOSTER 2001].

3.1.1 Vantagens

Existem várias aplicações práticas onde os *grids* podem oferecer grandes benefícios. De acordo com [SKILLICORN 2002], existem quatro tipos de *grid*: *grids* computacionais,

grids de acesso, *grids* de dados e *grids* datacêntricos. Cada um dos tipos oferece algumas vantagens, como é descrito a seguir.

- *Grids* computacionais: através de uma única interface, usuários utilizam o *grid* como um grande computador paralelo, executando as requisições dos usuários com grande aumento de desempenho;
- *Grids* de acesso: os *grids* de acesso oferecem um ambiente no qual usuários de diferentes organizações interajam como se estivessem na mesma plataforma de hardware. Este tipo não possui o desempenho como objetivo primário;
- *Grids* de dados: permitem que grandes conjuntos de dados sejam armazenados em repositórios, e manipulados com a mesma facilidade com que se manipula pequenos aquivos atualmente. O objetivo principal deste tipo de *grid* é a disponibilidade de dados;
- *Grids* datacêntricos: este tipo difere do *grid* de dados, no sentido que ao invés de mover os dados para a computação move a computação para os dados, em casos onde os dados devem permanecer imóveis. Uma aplicação dos *grids* datacêntricos é a mineração de dados distribuídos (*distributed data mining*).

Um levantamento da IBM sobre os benefícios do *grid* (*http://www-1.ibm.com/grid/about_grid/benefits.shtml*) aponta as seguintes vantagens:

- Otimização da infraestrutura – gerenciamento de carga de trabalho consolidada e capacidade para computação de alta demanda;
- Aumento na colaboração e no acesso a dados – distribuição global dos dados, grande colaboração multidisciplinar, inclusive entre organizações;
- Infraestrutura flexível e altamente disponível – possibilita balanceamento de carga e recuperação em caso de falha.

3.1.2 Arquitetura de Protocolos

A computação em *grid*, para ser utilizada globalmente, deve assumir que qualquer pessoa ou organização seja um participante em potencial. Para tanto, é necessário estabelecer uma padronização de protocolos, e assim facilitar a interoperabilidade entre domínios diferentes e, portanto, a criação de organizações virtuais extensíveis.

Para alcançar esta interoperabilidade, o *grid* utiliza os princípios do "modelo de relógio de areia", que consiste no mapeamento de serviços de alto nível para tecnologias específicas em um nível mais baixo, através de um pequeno número de protocolos. A figura 3.1 mostra um paralelo entre o modelo do relógio de areia e o modelo do *grid*. A figura 3.2 [FOSTER 2001] ilustra a arquitetura de protocolos do *grid*, que segue o modelo de relógio de areia.

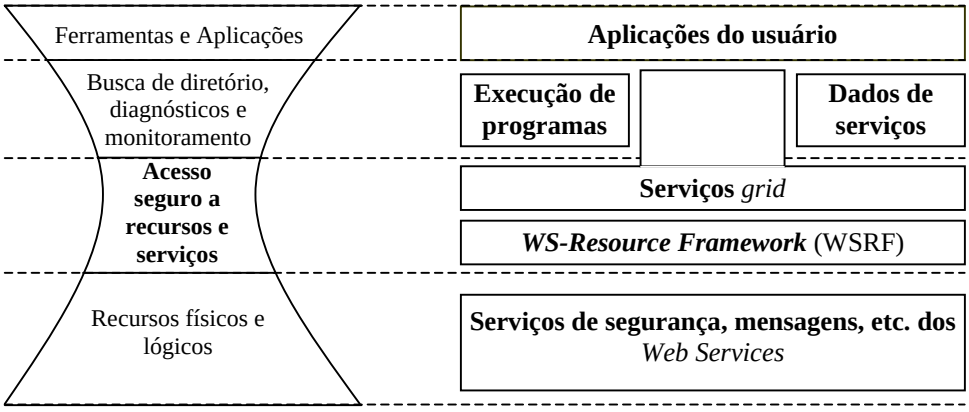

Figura 3.1 – Modelo de relógio de areia [FOSTER 2001]

Da mesma forma que na arquitetura de camadas da internet, os níveis superiores utilizam serviços oferecidos pelos níveis inferiores, sem precisar conhecer seus detalhes de implementação [TANENBAUM 1997].

A figura 3.2 ilustra a arquitetura de protocolos do *grid*. Os componentes da camada inferior, chamada "Camada de Construção" (*Fabric Layer*), implementam operações

específicas de cada recurso disponível, seja ele físico ou lógico. O mínimo que esta camada deve implementar é um mecanismo de busca para que os recursos permitam que terceiros conheçam sua estrutura, estado ou funcionalidades. É desejável também que se tenha algum controle sobre a qualidade de serviço, e para isto ser alcançado, deve ser implementado nesta camada um mecanismo de gerenciamento de recursos.

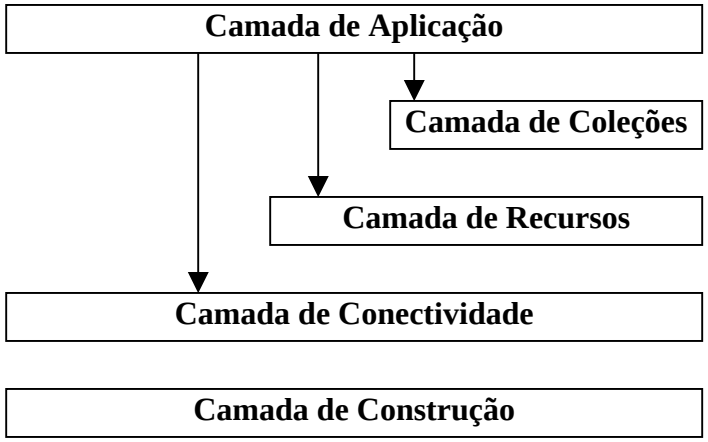

Figura 3.2 – Arquitetura de Protocolos do *Grid.* [FOSTER 2001]

A Camada de Conectividade, ou *Connectivity Layer*, possibilita a troca de dados entre recursos da Camada de Construção, através de protocolos de comunicação e autenticação. Sendo a flexibilidade um dos principais requisitos da computação em *grid*, é importante que se tenha controle também sobre o nível de proteção dos dados, além de suportar outros protocolos de transporte, além do TCP [FOSTER 2001].

A próxima camada da arquitetura é a Camada de Recursos (*Resource Layer)*, que define protocolos para negociação segura, iniciação, monitoramento, controle, contabilidade e pagamento por operações compartilhadas em recursos individuais (uma requisição pode acarretar na alocação de vários recursos). Os protocolos da Camada de Recursos invocam funções da Camada de Construção para acessar e controlar recursos locais. Duas classes de protocolos são aqui definidas, os protocolos de informação, utilizados para obtenção da estrutura e estado de um recurso, e os protocolos de gerência,

que funcionam como intermediários para garantir a consistência das operações solicitadas [FOSTER 2001].

A Camada de Coleções (*Collective layer*) é assim chamada pelo fato de ser responsável pela interação entre conjuntos (ou coleções) de recursos através dos protocolos e serviços nela definida. Os componentes desta camada podem implementar uma grande variedade de comportamentos compartilhados sem precisar de novos requisitos no recurso que está sendo compartilhados. Alguns exemplos são: serviços de diretório, serviços de monitoramento e diagnóstico, serviços de replicação de dados e serviço de descoberta de *software* [FOSTER 2001].

A Camada de Aplicação (*Application Layer*) equivale às aplicações, *frameworks* ou bibliotecas, que são construídas com base nos serviços oferecidos pelas camadas inferiores, como ilustra a figura 3.2 [FOSTER 2001].

3.2 Organizações Virtuais

Organizações virtuais são conjuntos dinâmicos de indivíduos, organizações e recursos, cujo objetivo é criar uma política de colaboração entre duas ou mais instituições. Porém, para isto ocorrer, é necessário estabelecer os limites de utilização de recursos entre fornecedor e consumidor. Estes limites são definidos através de políticas de segurança e mecanismos que variam de uma instituição para outra.

Apesar de existirem várias tecnologias disponíveis de computação distribuída, a maioria suporta apenas compartilhamento de recursos dentro dos limites de uma organização. No momento em que se deseja transpor esta barreira, surgem algumas dificuldades, como a falta de flexibilidade e de controle. A tecnologia de *grids* tem por objetivo resolver estes problemas, garantindo a possibilidade de formação de organizações virtuais extensíveis.

Integrantes da mesma VO (Virtual Organization) possuem a vantagem de ter acesso direto a softwares, dados, computadores e outros recursos compartilhados. Uma

organização pode afiliar-se a mais de uma VO, compartilhando recursos diferentes [FOSTER 2001]. Além disso, as VOs possuem a flexibilidade de aumentar e diminuir o número de participantes durante seu tempo de vida.

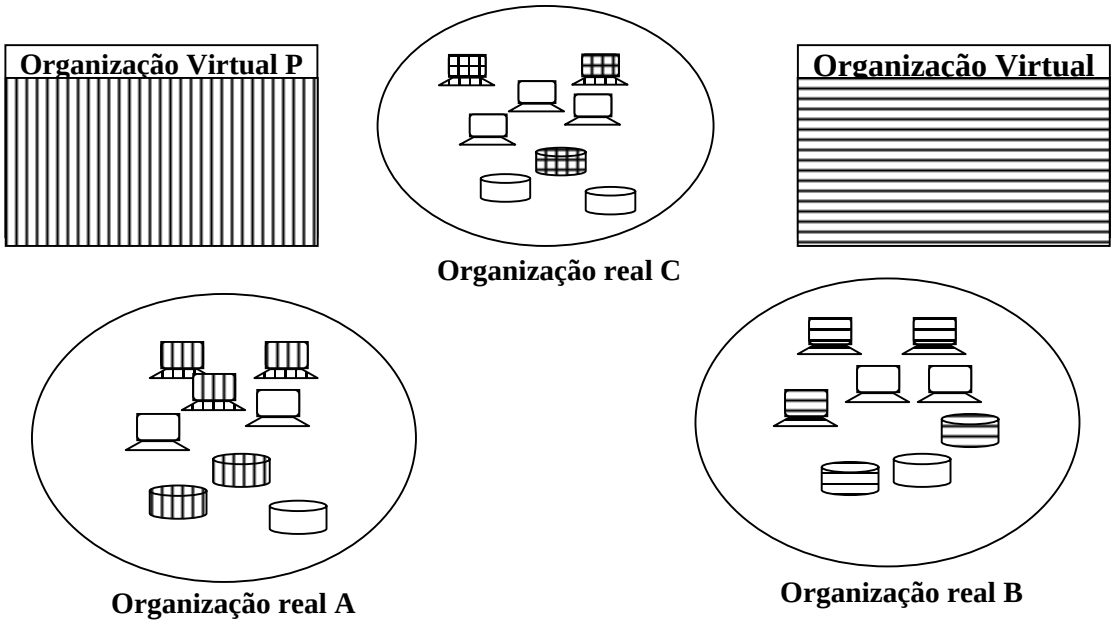

Figura 3.3 – Exemplo ilustrando que uma organização real pode participar de uma ou mais organizações virtuais simultaneamente (P e Q).

Com a natureza dinâmica das organizações virtuais, surge a necessidade de um mecanismo de descoberta de quais recursos estão disponíveis em um determinado momento, suas qualidades de serviço e as políticas de acesso. As tecnologias de *Web Services* e de *grid* compartilham algumas funcionalidades, o que motivou a GGF (*http://www.ggf.org*) a utilizar os *Web Services* como base do *grid*. Desta combinação de tecnologias surge o *Open Grid Service Architecture*.

3.3 Open Grid Service Architecture (OGSA)

Um dos principais requisitos da computação em *grid*, como foi dito anteriormente, é o compartilhamento e uso coordenado de recursos entre os participantes das organizações

virtuais. O OGSA é uma extensão da tecnologia de *grids*, que visa, entre outras coisas, agregar os preceitos dos *Web Services* para modelar tais recursos em termos de serviços [FOSTER 2001].

Em uma perspectiva orientada a serviço, um requisito essencial é a questão da interoperabilidade, que pode ser dividida em duas partes: (a) definição de interfaces de serviços e (b) identificação dos protocolos que podem ser utilizados para invocação de uma interface em particular [FOSTER 2002].

A visão orientada a serviço permite que se tenha transparência quanto à localização de um recurso (se local ou remoto) e permite a adaptação dos serviços do sistema operacional local. A visão orientada a serviço simplifica a virtualização de recursos, cujos benefícios são:

- Permitir o acesso a recursos de forma consistente através de múltiplas plataformas heterogêneas, com transparência de localização do recurso;
- Habilitar mapeamento de múltiplas instâncias de recursos lógicos para o mesmo recurso físico;
- Permitir a composição de serviços para formar um serviço mais sofisticado, sem se preocupar em como os serviços foram implementados;
- Possibilitar o mapeamento dos comportamentos semânticos de um serviço para facilidades da plataforma nativa.

Para facilitar a virtualização, surgiu a necessidade de uma linguagem capaz de expressar as funções dos serviços em um formato padrão de forma que qualquer implementação de um serviço pode ser invocada da mesma maneira. O OGSA adotou a linguagem WSDL (*Web Services Description Language*) utilizada nos *Web Services*.

3.3.1 Apresentação da Arquitetura

O que caracteriza uma arquitetura de rede é a organização de protocolos em camadas, que fornecem serviços para camadas superiores. Os detalhes de implementação dos serviços são omitidos, simplificando assim a maneira como são acessados - através de interfaces existentes entre cada camada [TANENBAUM 1997]. O OGSA é considerado uma arquitetura pelo fato de possuir um conjunto de interfaces bem definidas, a partir das quais, sistemas podem ser construídos [MYER 2003].

Quanto mais simples os serviços oferecidos por uma camada, maior a flexibilidade para criação de serviços mais complexos na camada superior. O OGSI (*Open Grid Service Infrastructure*) [TUECKE 2003] segue esta abordagem, sendo o principal benefício, a existência de um comportamento comum para todos os *grid services* [SANDHOLM 2003].

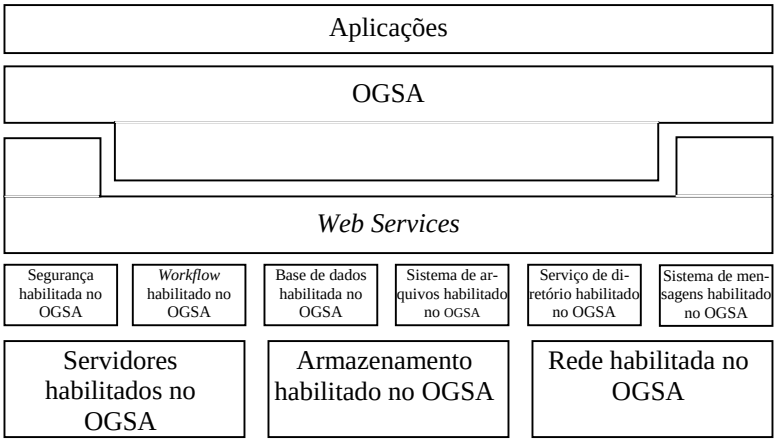

Figura 3.4 – Estrutura do *Open Grid Service Architecture* [MYER 2003]

A figura 3.4 ilustra como o OGSA complementa a tecnologia de *Web Services* para adequar-se à estrutura do *grid.* Ao invés de focar nos recursos em si, esta estrutura direciona sua atenção aos serviços oferecidos por estes recursos. As funcionalidades dos recursos (físicos ou lógicos) são abstraídas em serviços do *middleware*. Portanto as

aplicações dos clientes são construídas de modo muito simplificado, sem a necessidade de se conhecer detalhes de implementação do serviço.

3.3.2 Web Services

A tecnologia de *Web Services* é, por si só, uma abordagem de computação distribuída bastante interessante, que consiste na abstração de aplicações, escritas nas mais diversas linguagens, direcionadas para qualquer plataforma. Os usuários que farão uso das funcionalidades da aplicação direcionam suas requisições para o *web service* desejado, que é uma interface entre o usuário e o código objeto da aplicação.

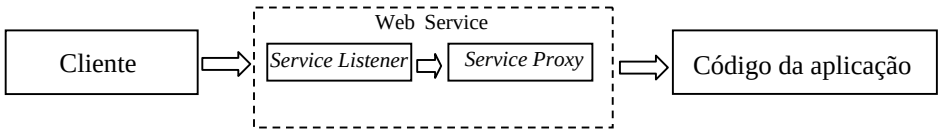

Figura 3.5 – Um *web service* na prática

Como ilustra a figura 3.5, os *web services* funcionam da seguinte forma: o servidor recebe uma requisição através de mensagens encapsuladas em protocolos de "peso leve" (*lightweight*) padrões, como o "http". Esta requisição é recebida pelo *Service Listener*, que conhece o protocolo em questão, e traduz o conteúdo da mensagem para o formato que é compreendido pelo *Service Proxy*. Este, por sua vez, é responsável pela chamada ao procedimento especificado com os parâmetros incluídos na mensagem. Caso a função solicitada aguarde um retorno, o *Service Proxy* codifica este retorno para repassá-lo ao *Service Listener*, e este para o cliente.

Uma das vantagens de se utilizar *Web Services* é que as requisições são normalmente direcionadas através da porta 80, sendo desnecessária a liberação de novas portas do roteador (o que dificulta o gerenciamento da segurança). Isto facilita a implantação de *Web Services* nas organizações, permitindo assim a interoperabilidade entre elas.

3.3.3 Grid Services

Grid Services são *Web services* que seguem algumas convenções que definem a maneira como são acessados. Pelo fato da tecnologia de *Web services* não satisfazer a todas as necessidades do OGSA, algumas extensões foram incluídas. Algumas destas extensões são apresentadas na seqüência.

Cada *grid service* é unicamente identificado através do seu *Grid Service Handle* (GSH), que deve ser uma URI (*Uniform Resource Identifier*) válida. Ao invés do GSH referenciar diretamente uma instância de um serviço *grid*, um serviço chamado *HandleResolver* faz o mapeamento para um *Grid Service Reference* (GSR). O GSR age como um apontador da rede que faz a ligação para o serviço. A linguagem WSDL é utilizada para descrever dados específicos do serviço, como o local do serviço na rede, o protocolo e outras características. As vantagens obtidas com este sistema de chamada em dois níveis são:

- Relações de 1:n (um para vários) entre GSH e GSR, possibilitando um sistema de balanceamento de carga (*load balance*) [GRIMSHAW 2004];

- Com o passar do tempo, o descritor do serviço (GSR) pode ser alterado, já que uma instância de serviço pode mudar de local por motivos de sobrecarga, antecipação de falha ou aumento do desempenho.

Outra importante extensão nos *grid services* é a utilização de dados de serviço – uma coleção de dados estruturados que possuem um ou mais elementos de dados de serviço (SDE – *Service Data Elements*). Os dados de serviço são utilizados para expor o estado interno dos *grid services*, podendo ser expostos também para atualização do estado de um serviço ou como critério de buscas de serviços. Os SDEs são utilizados nos *grid services* no mesmo sentido que os atributos são utilizados nas linguagens de programação orientadas a objeto (como Java, C++, Smalltalk, etc) [TUECKE 2002].

A especificação dos *grid services* [TUECKE 2002] define também um serviço para criação de novos *grid service*, e outro para definição do seu tempo de vida. As *factories* são serviços responsáveis pela criação de instâncias de um determinado tipo de serviço. Quando se especifica um serviço do tipo *factory*, é definido um intervalo dos valores mínimo e máximo do tempo de término do serviço, e no momento que um cliente solicita a criação de um *grid service*, a *factory* seleciona o tempo inicial do intervalo.

3.4 GLOBUS

O objetivo do *Globus* é fornecer uma infraestrutura para suportar, facilitar e melhorar a qualidade de implementações de vários tipos de serviços e softwares de metacomputação. O *Globus Toolkit* é um conjunto de serviços e bibliotecas de software baseado em comunidade e de código aberto que suporta aplicações *grid* [LEWIS 1996].

3.4.1 Sistema de Monitoramento e Descoberta - MDS

O serviço de monitoramento e descoberta (MDS) tem como requisitos fundamentais:

- Acesso a informações estáticas e dinâmicas relacionadas a componentes do sistema
- Uma base para configuração e adaptação em ambientes heterogêneos e dinâmicos
- Acesso flexível e uniforme às informações
- Acesso eficiente a dados dinâmicos
- Acesso a várias fontes de informações
- Manutenção descentralizada

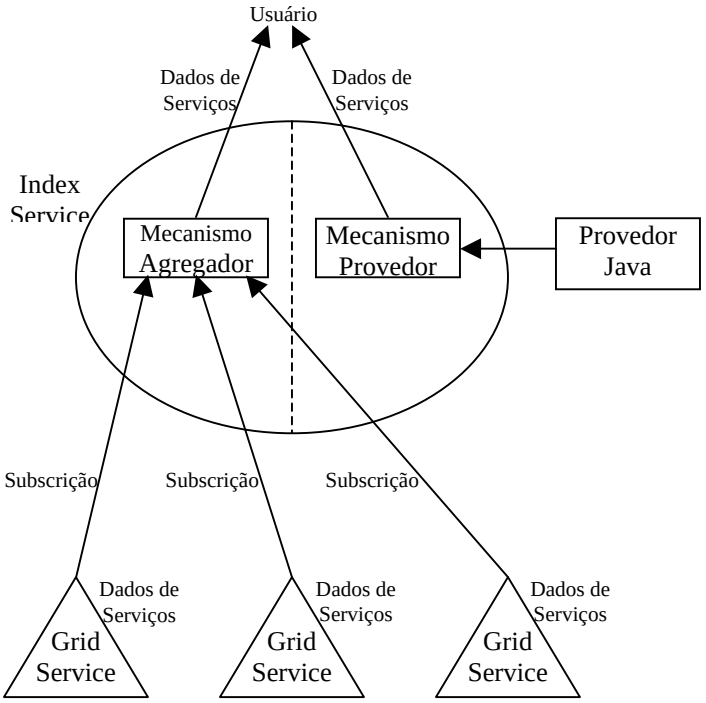

Figura 3.6 – Funcionamento do Index Service

Informações como características de hardware, de software e de rede são publicadas neste serviço como dados de serviços. Cada *grid service* possui elementos de dados de serviços atribuídos a ele, sendo nada mais que uma coleção de informações estruturadas (representadas através da linguagem XML), que podem ser facilmente indexadas e encontradas. Isto influencia na decisão de qual equipamento o usuário escolherá para executar um determinado serviço. Esta decisão, no entanto, é responsabilidade do serviço de índice (*Index Service*), que baseado em alguns critérios fornecidos pelo usuário encontram o serviço mais adequado a cada busca.

Na versão anterior do Globus Toolkit, o MDS utilizava o LDAP (*Lightweight Directory Access Protocol*) para lidar com tais informações. Dois componentes integrantes do MDS2.2 são o GRIS (*Grid Resource Information Service*) e o GIIS (*Grid Index Information Service*). O GRIS é um sistema de busca de informações sobre os serviços, equivalentes aos dados de serviços descritos anteriormente. Este componente permanece na

versão 3 do Globus, embora esteja modificado de forma a estar em conformidade com os mecanismos baseados em XML. Já o GIIS possui a função de buscar serviços baseados na sua natureza, por exemplo, serviços de armazenamento de dados ou um serviço de renderização distribuída de imagens. No GT3, o serviço equivalente a este é o serviço de índice (*Index Service*), o qual será descrito com mais detalhe em uma subseção, devido à sua importância.

3.4.1.1 Index Service

Na arquitetura OGSA, todos os recursos são modelados em termos de serviços chamados *grid services*, que nada mais são que *web services* com algumas funcionalidades a mais. Os *grid services* fornecem um conjunto de interfaces, que seguem algumas convenções específicas. Estas interfaces são utilizadas para descoberta de serviços, criação dinâmica de serviços, gerenciamento de tempo de vida de um serviço, notificações e administração [FOSTER 2002]. Dentre os motivos pelos quais tais funcionalidades foram atribuídas aos *grid services* estão o gerenciamento de instâncias de serviços transientes, e a existência de estado nos *grid services*. Um serviço construtor (*Factory Service*) persistente é responsável pela construção de serviços transientes. Cada vez que um novo serviço é criado, um novo identificador (GSH) é atribuído a ele. Este identificador é uma URL concatenada a um *hash* calculado em cada criação de um novo serviço, o que os torna únicos.

Como os *grid services* possuem um estado interno, se dois usuários utilizarem a mesma instância de um serviço transiente, ou utilizarem um serviço persistente, o resultado pode ser invalidado. Desta forma, cabe ao usuário utilizar um serviço construtor para criar uma nova instância, e ao término da computação, destruí-la. Ao mesmo tempo existem situações onde o estado interno de um serviço é uma informação importante, e que os vários usuários que o acessam devem ter acesso a ele.

O *Index Service* funciona da seguinte maneira: a partir de uma requisição feita para a interface *factory*, uma nova instância de um determinado *grid service* é criada e um identificador único (GSH) é atribuído a ele. O GSH, entretanto, deve ser convertido em um GSR (*Grid Service Reference*), que contém todas as informações para efetivamente acessar

o serviço. Um serviço de repositório (*Registry*) contém informações sobre os *grid services*, e recebe pedidos de consultas (e outras operações) dos usuários. Um serviço de notificação habilita o usuário a se subscrever a ele, para posteriormente receber notificações a respeito de eventos relacionados a um serviço, como mostra a figura 6.2. (*GT3 Index Service User´s Guide - http://www.globus.org*).

3.4.2 Gerenciamento e Alocação de Recursos GRID - GRAM

No projeto Globus, existe um conjunto de serviços que coletivamente são responsáveis pela alocação e gerenciamento de recursos remotos. A sigla GRAM, em algumas publicações tem como significado *Grid Resource Allocation and Management*, em outras publicações pode significar *Globus Resouce Allocation Manager* ou ainda *Grid Resource Acquisition and Management*.

O GRAM fornece uma única interface para solicitação e alocação de recursos em sistemas remotos para a execução de tarefas. Fornece também um mecanismo de autorização que se baseia nas identidades GSI, e uma tabela de mapeamento que relaciona identidades GSI a contas de usuários locais.

Um dos papéis do GRAM é reduzir a complexidade no gerenciamento de recursos remotos. Enquanto os sistemas locais podem utilizar vários mecanismos de gerenciamento (organizadores, sistemas de fila, sistemas de reserva, interfaces de controle), usuários e desenvolvedores de aplicações precisam conhecer apenas o GRAM para solicitar e utilizar os recursos do sistema. Esta funcionalidade entra em conformidade com o modelo de relógio de areia, como foi ilustrado no capítulo 3 (figura 3.1). A figura 3.7 ilustra como o GRAM se encaixa neste modelo:

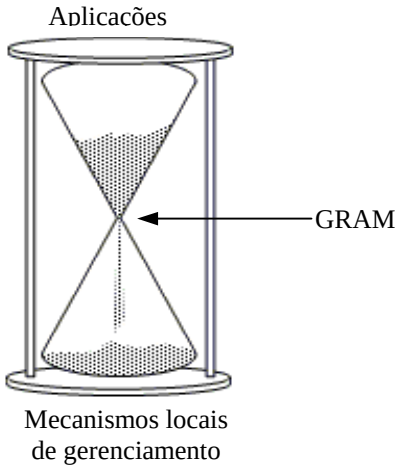

Figura 3.7 – GRAM e o modelo de relógio de areia

Dentre os vários serviços que o Globus oferece, o GRAM é o mais complexo em uma perspectiva de segurança, já que deve garantir a instanciação dinâmica, segura e remota de processos envolvendo interações seguras com um cliente remoto e o sistema operacional local [WELCH 2003]. A figura 3.8 ilustra a invocação de um serviço através do GRAM.

Para invocar um trabalho computacional no *grid*, um cliente deve descrever o trabalho a ser executado em termos de: nome, diretório em que se encontra, onde devem ser armazenados os dados de entrada e saída, e a fila na qual o trabalho deve ser executado. Esta descrição é enviada para o recurso e resulta na criação de um Serviço de Trabalho Gerenciado (*Managed Job Service* – MJS), que é um *Grid Service* que age como uma interface que pode ser associada a um trabalho computacional, instanciando-o e então permitindo que ele seja controlado e monitorado por mecanismos padrão do *grid* e dos *Web Services* [WELCH 2003].

O Serviço de Criação de MJS é responsável por criar um MJS através de uma operação *create*. Este serviço é conhecido por *Master Managed Job Factory Service* (MMJFS), significando “Serviço Principal de Criação de Trabalhos Gerenciados”. Cada recurso possui em execução um MMJFS, que é o responsável pela invocação de “Serviços de Criação de Trabalhos Locais Gerenciados” (*Local Managed Job Factory Services* -

LMJFS) para os usuários, de acordo com a necessidade. O *Proxy Router* é um serviço que direciona requisições do usuário para o seu LMJFS (se presente) ou para o MMJFS, caso o LMJFS do usuário não esteja presente.

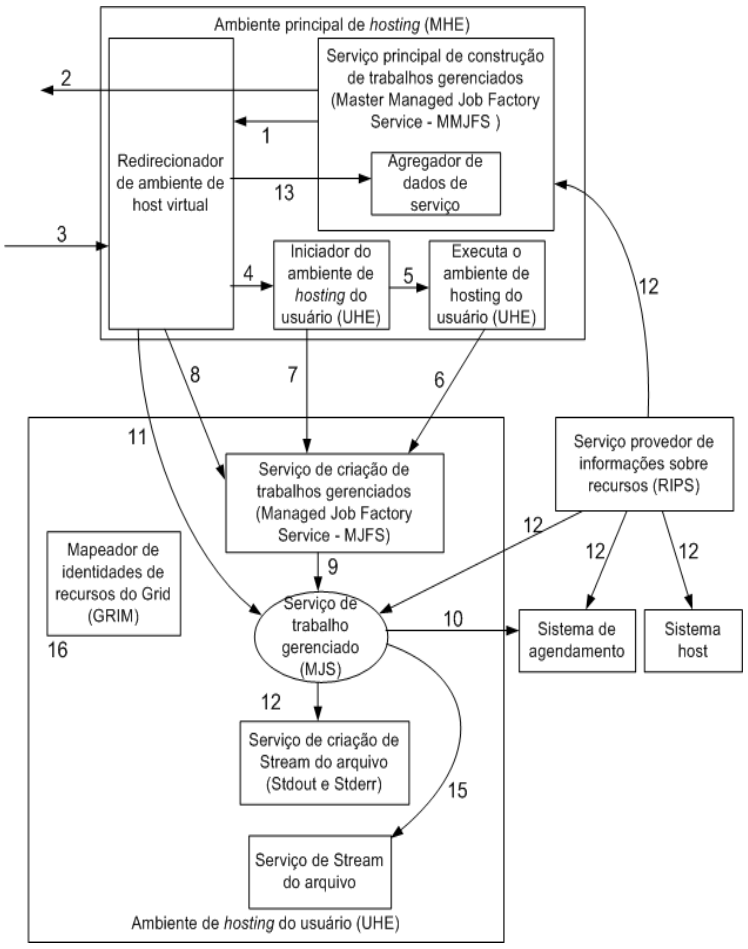

Figura 3.8 – Passos para a invocação de um serviço através do GRAM.

1. O Master é configurado para utilizar o redirecionador para redirecionar requisições para ele e utilizar o Iniciador UHE quando não hover um UHE em execução para o usuário. Uma requisição de criação de serviço para o Master utiliza o redirecionador para invocaro módulo iniciador do UHE e iniciar um novo UHE;
2. O Master publica seu GSH em um repositório remoto;
3. Um cliente envia uma solicitação de criação de serviço, que é recebida pelo redirecionador;
4. O redirecionador invoca o iniciador de UHE que autoriza a solicitação baseado no arquivo *grid-mapfile* para determinar o nome local do usuário e porta, a fim de construir uma URL alvo;
5. O redirecionador tenta encaminhar a chamada para a URL alvo. Se o UHE não estiver ativo, a chamada não será encaminhada e o módulo iniciador é invocado;
6. O lançador UHE cria um novo processo UHE sob o uid (identificador) do usuário local autenticado;
7. O iniciador UHE aguarda até que o UHE esteja ativo e então retorna a URL alvo para o redirecionador;
8. O redirecionador encaminha o pedido de criação de serviço para o MJFS não-modificado, e é realizado o processo de autenticação e autorização;
9. O MJFS cria um novo MJS;
10. O MJS submete o trabalho computacional para um sistema de agendamento;
11. As chamadas posteriores para o MJS serão redirecionadas pelo redirecionador;
12. O RIPS pode fornecer dados para instâncias MJS e para o Master. O RIPS adquire dados do sistema de agendamento local, sistema de arquivo e informações do sistema *host*;
13. Requisições de busca de serviços (*FindServiceData*) para o Master devem resultar em um Elemento de Dados de Serviços (SDE) ou serão redirecionadas para o MJFS do UHE do solicitante.
14. Para a criação de um *stream* de saída e de erros (que serão encaminhados para o cliente), o MJS cria dois serviços de criação de *stream* de arquivo (um para saída normal e outra para erros);
15. O MJS cria instâncias de serviços de *stream* do arquivo, como especificado na solicitação do trabalho computacional;
16. O manipulador GRIM é executado no UHE para criar um certificado de *host* do usuário. Este certificado é utilizado para autenticação mútua entre o serviço MJS e o cliente.

3.5 *Grid* x *Cluster*

Freqüentemente ocorre confusão entre os termos *grid* e *cluster*. Algumas características tornam cada tecnologia única. A tabela abaixo mostra o paralelo entre elas:

GRID	***CLUSTER***
Pode integrar recursos de armazenamento, rede, processamento, entre outros. Contêm máquinas de diferentes fabricantes, e variados sistemas operacionais, caracterizando uma grande heterogeneidade.	Geralmente contêm apenas um tipo de processador e sistemas operacionais
Uma das suas principais características é que são dinâmicos. Os recursos que integram um *grid* podem se conectar ou desconectar a qualquer momento.	Os *clusters* tipicamente contêm um número estático de processadores e recursos.
Os *grids* são distribuídos em redes locais, metropolitanas ou amplamente distribuídos geograficamente.	Os recursos de um *cluster* geralmente são mantidos em um mesmo local, já que a tecnologia de interconexão de seus nodos fornece baixa latência de rede, podendo causar problemas com o aumento da distância entre os recursos.
Grids suportam grande escalabilidade.	O espaço físico onde se situa um cluster dificulta a escalabilidade

Tabela 3.1 – Comparação entre as tecnologias de *Grid* e *Cluster*

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

3.5 Considerações Finais do Capítulo

A computação em grid faz parte de uma nova realidade que está ainda em desenvolvimento. No entanto, grandes organizações já estão fazendo uso da tecnologia e obtendo seus benefícios, como é o caso da Oracle (http://www.oracle.com/ technology/tech/grid/), Intel (http://www.intel.com/business/bss/technologies/grid/), IBM (http://www-1.ibm.com/grid/), e HP (http://www.hp.com/techservers/grid/).

A fim de garantir uma padronização e desenvolvimento coordenado da tecnologia de grids, foi criado o *Global Grid Forum*. Este esforço é fundamental para garantir a interoperabilidade, que é um dos focos principais da computação em *grid*.

Sendo considerada uma das mais importantes tecnologias em desenvolvimento, a computação em *grid* está despertando cada vez mais interesse na comunidade científica.

Projetos que utilizam *grid* para pesquisar a cura para o câncer ou um antídoto para o antraz, ou mesmo para descobrir vida extraterrestre inteligente, são os mais conhecidos (http://www.grid.org). Percebe-se então o potencial dessa tecnologia. Não importa se o objetivo é desenvolver uma arma nuclear, ou encontrar a cura para todas as doenças existentes, a computação em grid é uma grande evolução e possivelmente um marco na ciência da computação e tecnologia da informação.

Capítulo 4

Segurança em Grid

O objetivo principal da computação em *grid* é a formação de organizações virtuais, onde recursos são compartilhados e utilizados de forma coordenada. O que diferencia uma organização virtual de uma convencional, é a localização dos recursos compartilhados, já que em uma organização virtual os recursos não estão localizados necessariamente no mesmo domínio administrativo. A preocupação com a segurança, que sempre foi grande desde o surgimento da internet, aumenta ainda mais, já que existe o consentimento da organização proprietária na abertura para uso dos seus recursos. Mecanismos de segurança devem ser detalhados para garantir que nada além do especificado esteja disponível para os participantes da organização virtual.

Para alcançar os requisitos de segurança dos *grids* deve-se respeitar a infraestrutura de segurança de cada organização. As especificações de segurança estabelecidas no *grid* devem ser consistentes com a segurança do sistema local, ou seja, as soluções de segurança interdomínios utilizada pelo *grid* devem interoperar com os mecanismos de controle de acesso encontrados nos domínios individuais [FOSTER 1998]. Este é um dos desafios encontrados na especificação de um ambiente de *grid* seguro.

Dentre as propriedades de segurança necessárias em sistemas computacionais (confidencialidade, integridade, autenticação, autorização, não-repúdio), a autenticação de usuários é considerada um pré-requisito para que sejam alcançadas as demais. Através da autenticação o sistema reconhece a identidade de um usuário, que por sua vez é submetido a um processo de autorização para poder acessar os recursos a ele pertinentes, e o habilita a verificar a integridade e a garantir a confidencialidade dos seus dados [FOSTER 1998]. Por este motivo, a maioria das soluções de segurança para *grids* são direcionadas principalmente para autenticação de usuários.

4.1 Requisitos de Segurança no *Grid*

A computação em *grid* pode envolver recursos de muitos domínios administrativos simultaneamente, o que diferencia o *grid* da computação distribuída convencional. Dezenas, centenas ou até milhares de nodos podem participar de uma computação, e uma relação de segurança deve ser estabelecida com todos os nodos envolvidos. Além disso, o conjunto de usuários e de recursos é grande e dinâmico, podendo variar rapidamente. Os requisitos básicos do modelo de segurança do OGSA são que os mecanismos de segurança devem ser "plugáveis" e que possam ser descobertos por um serviço requisitante a partir da descrição do serviço. O modelo de segurança do OGSA deve objetivar os seguintes requisitos de segurança:

- *Autenticação*: Necessidade primordial na segurança dos sistemas computacionais, a autenticação no *grid* deve ser possível através de mecanismos diferentes. A autenticação do *grid* deve fornecer "pontos de plugue" para que múltiplos mecanismos de autenticação possam ser escolhidos [NAGARATMAN 2002];
- *Single sign-on*: O *login* de um usuário deve ser requisitado apenas uma vez no *grid*, e recursos podem ser alocados, utilizados e liberados sem que seja necessário se autenticar novamente ao sistema [FOSTER 1998].
- *Delegação*: Uma entidade delega um conjunto de direitos a outra a fim de que esta aja em seu nome. Este procedimento deve ser executado com os cuidados necessários para que este conjunto de privilégios não seja mal utilizado, restringindo

um tempo de vida e apenas o conjunto de direitos necessários para a execução da tarefa em questão [NAGARATMAN 2002].

- *Interoperabilidade com soluções de segurança locais*: O acesso aos recursos locais é definido por uma política de segurança local, imposta por mecanismos de segurança do sistema local [FOSTER 1998].
- *Autorização*: Políticas de autorização são utilizadas para controlar o acesso a serviços OGSA em condições específicas. Além disso, é possível ao cliente especificar os fornecedores de serviço de sua confiança, através de políticas de solicitação, ou seja, em quem o cliente confia para fornecer o serviço desejado [NAGARATMAN 2002].
- *Privacidade*: Permite que o solicitante e o fornecedor do serviço definam e imponham critérios de privacidade, levando em consideração detalhes da identidade de cada um deles (informações que identifiquem unicamente uma entidade), propósito da solicitação do serviço, etc. [NAGARATMAN 2002].
- *Integridade*: Assegura que alterações não autorizadas sejam detectadas pelo destinatário da mensagem [NAGARATMAN 2002].
- *Proteção de credenciais*: Todos os tipos de credenciais suportadas pelo sistema (*tokens*, senhas ou chaves privadas) devem possuir um mecanismo de proteção [FOSTER 1998].
- *Logging seguro:* A gravação de informações operacionais e inclusão de *time stamps* nas mensagens é essencial para possibilitar auditorias e atingir um dos requisitos básicos da segurança computacional, o não-repúdio [NAGARATMAN 2002].
- *Exportabilidade*: Para garantir que o código seja exportável para outras plataformas, a política de segurança não pode impor o uso de criptografia em bloco [FOSTER 1998].
- *Troca de informações sobre a política de segurança*: Para que seja estabelecido um contexto de segurança entre fornecedor e consumidor de um recurso, é necessário que um lado conheça a política de segurança do outro. Entre as informações

trocadas podem estar os requisitos de autenticação, restrições de acesso, regras de privacidade, entre outras [NAGARATMAN 2002].

- *Garantia:* Devem existir meios para qualificar os níveis de garantia de segurança de um ambiente hospedeiro. Em outras palavras, é uma maneira de expressar características da proteção de um sistema (proteção contra vírus, uso de *firewall* para acesso à *internet*, etc), que são informações essenciais que serão levadas em consideração no momento de escolha do ambiente a ser utilizado [NAGARATMAN 2002].
- *Infraestrutura uniforme de credenciais*: Pelo fato de existirem diversos formatos, é necessária a imposição de um padrão para as credenciais utilizadas para identificação do usuário neste contexto multi-organizacional. O padrão mais utilizado atualmente é o X.509v3 [FOSTER 1998].
- *Travessia pelo firewall:* Os *firewalls* são barreiras necessárias dentro de qualquer organização, portanto é necessário que haja uma coexistência com o *grid* [NAGARATMAN 2002].
- *Suporte a múltiplas implementações*: O mecanismo de segurança a ser utilizado deve ser flexível, de forma que seja independente da implementação. Desta forma, uma aplicação é capaz de utilizar o esquema mais propício em cada situação [FOSTER 1998].
- *Suporte a comunicação segura em um grupo*: Uma computação pode compreender a execução de vários processos simultâneos que devem se comunicar para alcançar o seu objetivo. Como cada processo pode estar sendo executado em um domínio (ou computador) diferente, é necessário garantir a segurança desta comunicação [FOSTER 1998].
- *Overhead computacional proporcional à longevidade da colaboração:* A computação em *grid* pode ser usada tanto para colaborações de longo prazo como de curto prazo. No primeiro caso, que pode compreender meses ou anos, existe a necessidade de um esquema de segurança mais robusto. Isto significa que é aceitável (e necessário) que seja gasto mais tempo e processamento no processo inicial de autenticação, devido ao longo período de interação entre as partes

envolvidas. No segundo caso o *overhead* deve ser pequeno na autenticação, pois mesmo que se utilize um tamanho menor de chave, a chance de ocorrer comprometimento da segurança na comunicação é pequena, já que a interação entre as partes é de curta duração [WELCH 2003].

4.2 *Grid Security Infrastructure* (GSI)

O GSI é uma implementação da arquitetura de segurança do *grid*, desenvolvida no projeto Globus. O GSI fornece suporte para *proxies* de usuário, *proxies* de recursos (GRAM – *Globus Resource Allocation Manager*), e autoridades de certificação (CA´s) [FOSTER 1998].

Tendo em vista a impraticabilidade da imposição de padrões em cada domínio em particular, o mecanismo de segurança no *grid* deve ser implementado no contexto interdomínios. O GSI é um sistema de autenticação interdomínios, e cada um destes domínios aplica políticas locais sobre seus usuários e recursos, que permanecem fora da jurisdição do *grid*.

Quando um sujeito (um usuário, um processo que atua em seu benefício ou um recurso) de um domínio chamado “A” deseja utilizar um recurso situado em um domínio chamado “B”, isto deve ser feito através do mapeamento de um nome de usuário global para um nome local, neste domínio. Este nome local pode ser estático, dinâmico ou ser atribuído a um grupo de usuários que compartilham dos mesmos privilégios. Através da existência deste sujeito global é possível suprir o requisito de *single sign-on*, onde o usuário só efetua o processo de *login* uma única vez. Concluído o processo de autenticação, mesmo estando conectado remotamente o sistema considera o sujeito como se estivesse operando de dentro deste domínio [FOSTER 1998].

A importância do GSI é evidente pela natureza das organizações virtuais, onde o acesso aos serviços é controlado com base na identidade do usuário (verificada no processo de autenticação), e na verificação da sua permissão (autorização).

4.2.1 GSS-API

O Globus suporta vários mecanismos de segurança como chaves secretas compartilhadas, Kerberos, e criptografia de chave pública. Para alcançar esta flexibilidade e independência de protocolos, o GSI faz uso da interface GSS-API (*Generic Security Services Application Programming Interface*).

O GSI é implementado no topo do GSS-API, para que os protocolos de segurança do *grid* sejam traduzidos em chamadas GSS (Generic Security Services – Serviços de Segurança Genéricos). Portanto, pode-se explorar vários mecanismos de segurança do *grid* sem alterar a implementação do GSI (figura 4.1) [FOSTER 1998].

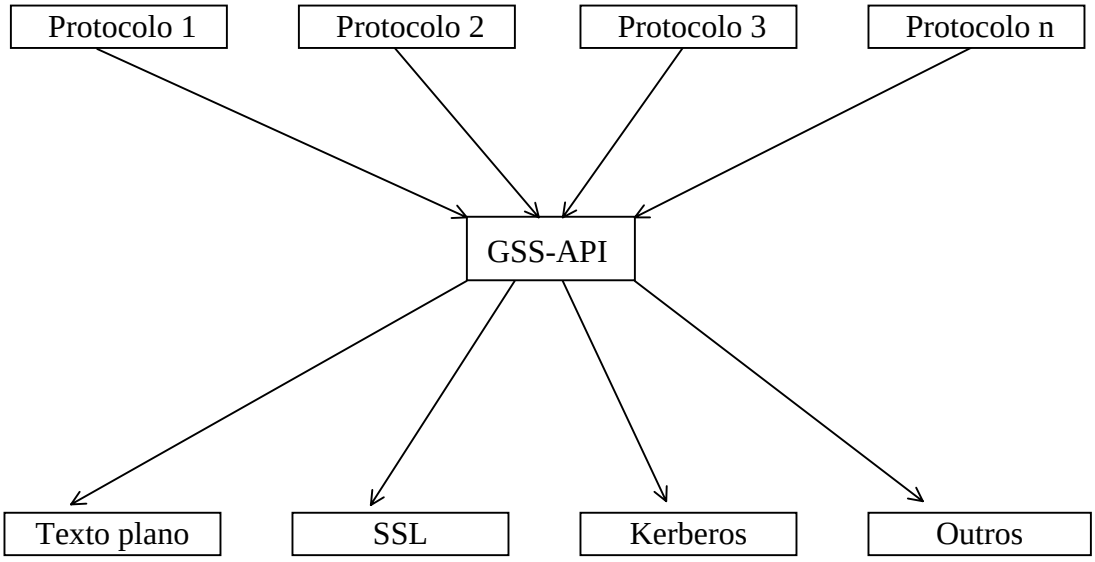

Figura 4.1: Mapeamento entre os protocolos P_i e os mecanismos M_i através do GSS-API

As vantagens do GSS-API que motivaram os pesquisadores do projeto Globus a utilizar esta interface na implementação do GSI são [LINN 1997]:

- Independência de mecanismo: Tanto abordagens criptográficas baseadas em chaves secretas ou chaves públicas podem ser utilizadas na implementação dos vários

serviços de segurança suportados pelo GSS-API, como autenticação, confidencialidade e integridade;

- Independência de associação com protocolos: Os serviços do GSS-API não estão associados a nenhum protocolo em particular, possibilitando que a mesma implementação seja utilizada por vários protocolos, ou até mesmo que uma aplicação faça uso destes serviços diretamente;
- Adequabilidade a vários ambientes de implementação: Não é necessário que o cliente GSS-API esteja dentro do perímetro TCB (*Trusted Computing Base*) onde é implementada a interface, de forma que seus serviços de segurança possam ser invocados tanto de dentro como de fora deste perímetro.

Apesar das vantagens citadas, o uso do GSS-API não oferece uma solução definitiva para delegação de direitos (a um processo) e nem para gerenciamento de contextos de grupo.

4.2.2 Arquitetura de Segurança do Grid

Um usuário pode enviar para o *grid* várias requisições separadas em momentos distintos, ou, em outros casos, manter uma aplicação executando por um grande período de tempo. Nestas duas situações, é comum a utilização de vários recursos durante uma computação. Portanto, para garantir a continuidade e fluidez do trabalho computacional, o *proxy* do usuário [FOSTER 1998] é utilizado para que o usuário não precise realizar o processo de autenticação separadamente para cada recurso em particular. O *proxy* carrega a identidade do usuário sem necessidade de intervenções. O *proxy* do usuário age em seu benefício por um limite determinado de tempo (o padrão é doze horas). No momento da criação do *proxy*, o usuário utiliza a sua chave privada para assinar o certificado do *proxy*, garantindo assim que o *proxy* foi criado por este usuário [FOSTER 1998].

Outro elemento desta arquitetura é o *proxy* do recurso. Sua finalidade é agir como um intermediário, fazendo as conversões entre operações de segurança do *grid* e mecanismos de segurança locais.

Baseada nas experiências vivenciadas até o momento, a arquitetura assume algumas operações no *grid*: a alocação de recursos por um usuário, alocação de recursos por um processo e comunicação entre processos interdomínios.

Os protocolos descritos nas seções abaixo confiam na integridade dos dados transferidos quando estes apresentam a devida assinatura, que consiste em um *hash* do arquivo, cifrado com a chave privada do emissor.

Os protocolos descritos a seguir descrevem como ocorre a criação do *proxy* do usuário. Os responsáveis pela alocação de recursos são o *proxy* do usuário e os processos criados durante uma computação.

4.2.2.1 Protocolo 1: Criação do Proxy do Usuário

A arquitetura de segurança do *grid* estabelece a criação de um *proxy* que executa ações em prol do usuário. Para isto ocorrer, o *proxy* deve carregar a identidade do usuário (como a sua chave privada, por exemplo), para obter os mesmos direitos de acesso que o seu criador. No entanto, esta abordagem apresenta problemas de segurança que são contornados com o uso de credenciais temporárias para o *proxy*, que além de mais difícil de ser descobertas, não coloca em risco as credenciais do usuário.

A credencial do *proxy* é assinada pelo usuário com a sua chave privada e contém informações essenciais para sua segurança como: a sua validade, os *hosts* que podem iniciar uma requisição e os *sites* que possuem os recursos a serem acessados. A segurança desta credencial é responsabilidade do *host* onde o *proxy* do usuário executa.

4.2.2.2 Protocolo 2: Alocação de Recursos

Este protocolo corresponde ao procedimento que o *proxy* do usuário executa ao emitir uma requisição para alocação de um recurso, através do *proxy* do recurso. Todas as

requisições feitas para um recurso são intermediadas pelo seu *proxy*, que é responsável pela organização dos pedidos e mapeamento das computações para os recursos.

O *proxy* do recurso é responsável pela validação de uma requisição, negando acesso para qualquer uma das seguintes situações: falha de autenticação, falha de alocação ou falha de autorização:

- Uma falha de autenticação acontece quando um determinado usuário não é reconhecido;
- Na falha de autorização o *proxy* do recurso reconhece o usuário, mas o invalida por falta de privilégios de acesso. Um exemplo: o usuário tem acesso ao recurso, mas apenas para leitura, e não para escrita. Outra possibilidade de ocorrência desta falha ocorre quando as credenciais do *proxy* do usuário já estão fora do período de validade;
- A falha de alocação ocorre quando o recuso não está disponível no momento da requisição, seja por motivo de manutenção ou excesso de usuários acessando no momento.

Quando o usuário utiliza um recurso, um ou vários processos decorrentes da computação serão criados. O protocolo de alocação de recursos cria uma credencial temporária para estes processos da mesma forma que acontece com o *proxy* do usuário. Assim, o processo possui também autonomia para alocação de recursos sempre que necessário, utilizando sua credencial para autenticar a si mesmo e o usuário o qual representa.

4.2.2.3 Protocolo 3: Alocação de Recursos por um Processo

A alocação de recursos no *grid* inicia mais freqüentemente dos processos criados dinamicamente durante uma computação, do que diretamente do usuário. Isto ocorre porque durante uma computação vários processos podem ser criados, enquanto o usuário (através do seu *proxy*) apenas inicia a computação.

O primeiro passo consiste na autenticação mútua entre o *proxy* do usuário e o processo, através das credenciais. Em seguida, o processo envia uma requisição (assinada) ao *proxy* do usuário contendo os detalhes da alocação. Caso o *proxy* do usuário decida continuar com a requisição, o protocolo 2 é utilizado para emitir a requisição de alocação de recurso ao *proxy* do recurso em questão. Finalmente, o identificador de um processo é assinado pelo *proxy* do usuário, delegando a ele seus direitos de acesso.

4.2.2.4 Protocolo 4: Registro de Mapeamento

O mapeamento entre o sujeito global e o sujeito local é feito através de uma tabela de relacionamento, que é criada pelo administrador do sistema local. O protocolo 4 descreve uma forma de se criar um mapeamento entre sujeitos globais e locais pelo próprio usuário, para reduzir a carga de trabalho do administrador do sistema local, e diminuir a possibilidade de erros.

Inicialmente os *proxy* do usuário e do recurso devem efetuar uma autenticação mútua e enviar uma requisição assinada (MAP-SUBJECT-UP) contendo como argumentos o nome do sujeito global e do recurso a ser acessado. Em seguida o usuário efetua o *login* no recurso, e inicia o processo de registro de mapeamento, que envia uma requisição MAP-SUBJECT-P para o *proxy* do recurso juntamente com os nomes do sujeito global e do recurso como argumento. Na seqüência, ocorrem os seguintes passos:

1. O *proxy* do usuário se autentica com o *proxy* do recurso;
2. O *proxy* do usuário (UP – *user proxy*) emite uma solicitação assinada MAP-SUBJECT-UP para o *proxy* do recurso, passando como argumento os nomes do sujeito global e do recurso;
3. O usuário efetua o *login* no recurso utilizando o método de autenticação do recurso e inicia o processo de registro de mapeamento;
4. O processo (P) de registro de mapeamento emite uma solicitação MAP-SUBJECT-P para o *proxy* do recurso, passando como argumento os nomes do sujeito global e do recurso;

5. O *proxy* do recurso aguarda as requisições MAP-SUBJECT-UP e MAP-SUBJECT-P com argumentos compatíveis;
6. O *proxy* do recurso se assegura que o processo de registro de mapeamento pertence ao recurso especificado na requisição de mapeamento;
7. Caso haja uma combinação, o *proxy* do recurso configura o mapeamento e envia uma confirmação para o processo de registro de mapeamento e para o *proxy* do usuário;
8. Se não houver acordo dentro de um tempo limite (MAP-TIMEOUT), o *proxy* do recurso descarta a requisição e envia um retorno ao requisitante;
9. Caso este retorno não seja recebido pelo requisitante dentro de MAP-TIMEOUT, assume-se a ocorrência de falha na requisição.

4.3 Segurança no OGSA

O *Open Grid Service Architecture* foi projetado para prover flexibilidade e extensibilidade ao *grid*, e isto se aplica também ao aspecto de segurança. Podem ser utilizados vários mecanismos de autenticação, e podem ocorrer variações inclusive dentro do mesmo mecanismo, como é o caso do PKI (*Public Key Infrastructure*) [SIEBENLIST 2002].

Como existem diferentes implementações de PKI (entre as quais a principal é a X.509) é necessário que haja um serviço de validação de certificado para ser utilizado no OGSA para:

- Busca de informações no certificado através de *parsing*;
- Validação conforme a política de segurança local; e
- Busca de informações de atributos, para possibilitar o uso de formatos diferentes de certificados.

No caso do *Kerberos*, os *tickets* e as mensagens trocadas durante a autenticação são encapsuladas em protocolos do OGSA (*OGSA WS Protocols*).

Desta forma, tanto as comunicações entre cliente e Kerberos como entre cliente e PKI, utilizam o mesmo protocolo, as mesmas funcionalidades de roteamento e os mesmos mecanismos de descoberta de serviços, o que facilita o roteamento através de *firewalls* e sistemas NAT (*Network Address Translation*) [SIEBENLIST 2002].

De acordo com [NAGARATMAN 2002], os princípios fundamentais do modelo de segurança podem ser classificados como:

1. Invocação segura de *grid services*: Antes de requisitar um serviço, o requisitante deve definir os detalhes sobre a sua Qualidade de Proteção (*Quality of Protection* - QoP) a fim de selecionar o serviço mais adequado a sua política de segurança. Esta política de segurança pode exigir um tipo específico de credencial e requisitos apropriados de confidencialidade e integridade. A arquitetura de segurança do *grid* deve possibilitar a seleção dinâmica de serviços em conformidade com as restrições da política de segurança do requisitante do serviço, já que este pode selecionar um provedor de serviço que melhor combina com as suas políticas.
2. Serviços de segurança do *grid*: Um ambiente OGSA possui um conjunto de serviços e primitivas de segurança que são os blocos fundamentais que fornecem um conjunto de serviços para a lógica da aplicação. Uma implementação OGSA pode decidir, através de uma política de autorização, se um determinado serviço pode ou não ser instanciado.

4.3.1 Modelo de Segurança do OGSA

Um cliente, antes de requisitar um serviço no *grid*, expressa a qualidade de serviço que deseja utilizar durante a computação, incluindo detalhes sobre as funções de segurança como autenticação, integridade e confidencialidade. Um acordo entre cliente e provedor do serviço é feito após a verificação mútua das políticas de segurança, e então é estabelecido um canal seguro. O serviço de busca e descobrimento do serviço mais adequado a cada situação é uma das várias funcionalidades herdadas dos *web services*.

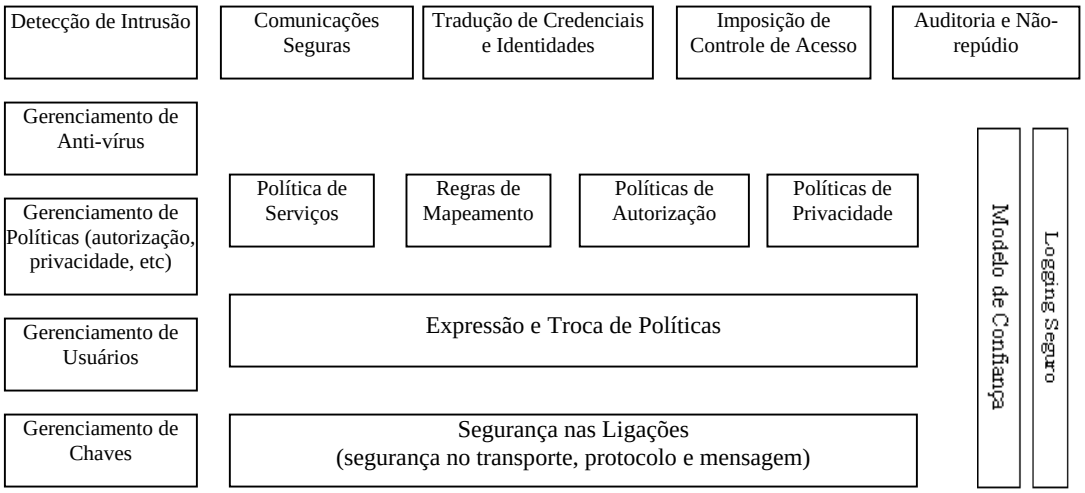

Figura 4.2: Arquitetura de segurança do OGSA. [NAGARATMAN 2002]

Na figura 4.2, os componentes de segurança são dispostos em camadas para evidenciar a dependência entre eles. Uma breve descrição de cada camada segue abaixo:

- A camada mais baixa (Segurança nas Ligações) é responsável pela interação segura das camadas superiores com os protocolos de transporte;
- A camada acima, Expressão e Troca de Políticas, é utilizada para definição e gerenciamento das políticas de segurança situadas nas camadas superiores a ela (Política de Serviços, Regras de Mapeamento, Políticas de Autorização e Políticas de Privacidade);
- No topo estão os serviços de segurança que são diretamente invocados pelas aplicações;
- Os componentes à direita são destinados a definição da base de confiança (Modelo de Confiança) e auditoria (*Logging* Seguro);
- Os componentes da pilha à esquerda são reservados à gerência e administração da infraestrutura: gerenciamento de usuários, gerenciamento das chaves, gerenciamento da política de segurança, anti-vírus e detecção de intrusos [SIEBENLIST 2002].

4.3.2 Modelo de Autorização no OGSA

Como os componentes de segurança no OGSA são modelados como serviços, isto possibilita a existência de várias implementações de uma mesma função de segurança.

Em um ambiente de *grid*, cada recurso possui seu conjunto de usuários com autorização para utilizá-lo. No modelo *Pull*, esquematizado na figura 4.3, quando um recurso recebe uma requisição, depois de autenticar a entidade requisitante ele verifica com o serviço de autorização se a entidade possui permissão para proceder com a operação.

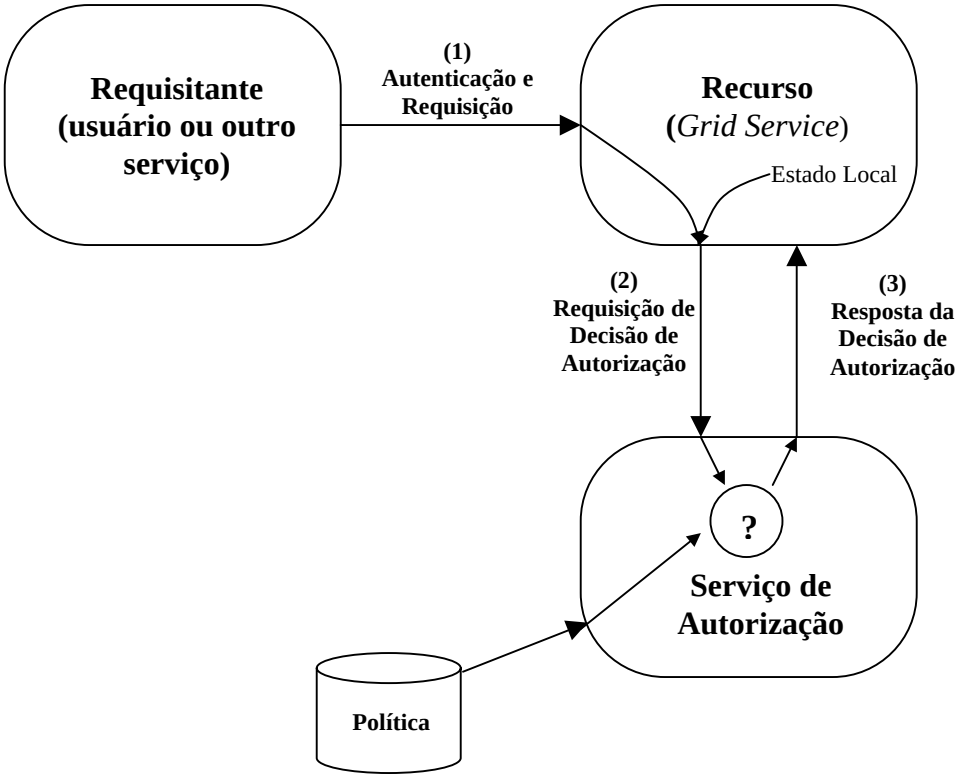

Figura 4.3: Modelo de autenticação *Pull*.

No modelo *Push* (figura 4.4), antes de solicitar o uso do recurso, o requisitante deve se comunicar com o serviço de autorização a fim de obter as permissões desejadas. Em posse deste conjunto de direitos, o recurso é capaz de avaliar diretamente se permite ou não a execução da operação solicitada.

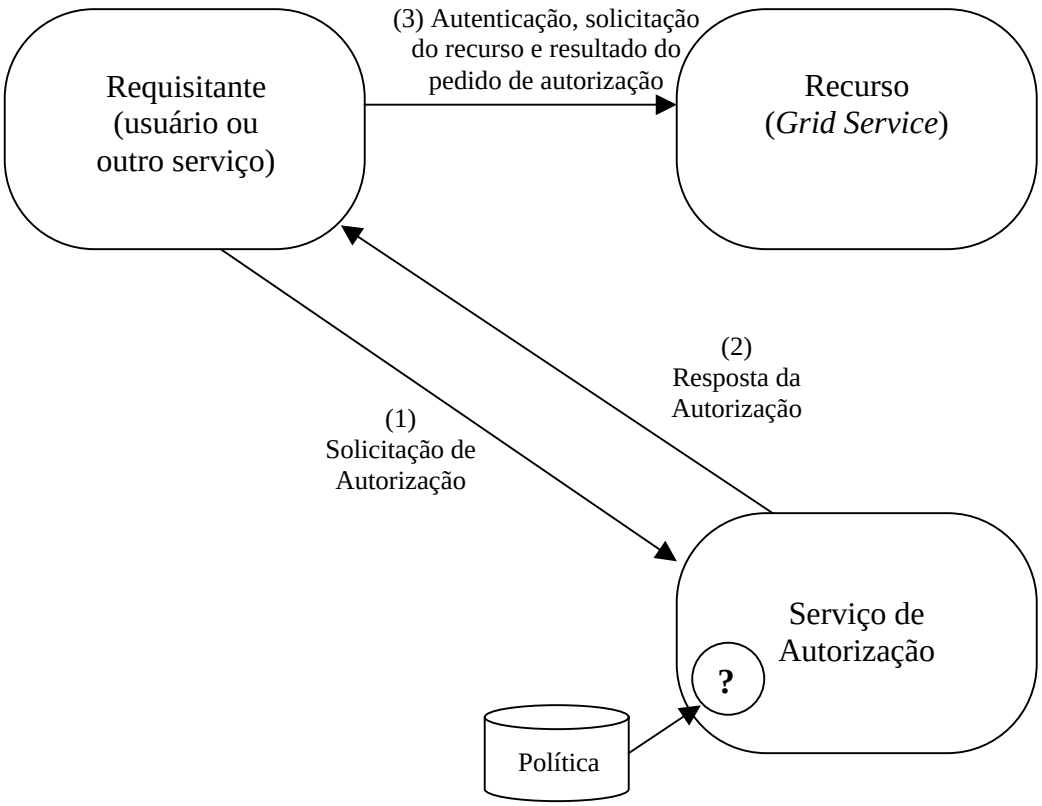

Figura 4.4: Modelo de autorização *Push*.

4.4 Segurança no *Globus Toolkit*

O modelo de segurança do GT2 se baseia no GSI. A implementação GSI utiliza como padrão de certificado o X.509, e como protocolo de segurança o SSL ou TLS, situados na camada de transporte. A combinação destes padrões permite que seja realizada a autenticação e a autorização dos usuários, além da proteção contra leitura e alteração das mensagens trocadas.

A versão 3 do Globus Toolkit se diferencia da anterior por utilizar o conceito de *Grid Services*, uma integração da tecnologia de *grid* e a de *Web Services.* Esta abordagem consiste na abstração de recursos como serviços e em um mecanismo de descoberta destes recursos.

A tecnologia de *Web Services*, na qual o OGSA se baseia, fornece alguns mecanismos de segurança. Os componentes que formam a arquitetura *WS-Security* (*WS-Security*, *WS-Policy*, *WS-Federation*, *WS-SecureConversation*, *WS-Privacy*, *WS-Trust* e *WS-Authorization*) constituem a base da segurança no OGSA. Nem todos os requisitos de segurança do OGSA são preenchidos pela *WS-Security* e, portanto, as especificações da *WS-Security* devem ser alteradas de forma a preencher tais requisitos [SIEBENLIST 2002].

A proposta do mecanismo de segurança do GT3 (GSI3) é fornecer a segurança necessária ao usuário como parte da infraestrutura, sem que o usuário precise instanciá-la na aplicação.

Como se baseia nos *Web Services*, o GT3 utiliza o protocolo SOAP (*Simple Object Access Protocol*) para comunicação, por facilitar a passagem por *firewalls*, já que utiliza a porta 80 (o SOAP geralmente é encapsulado dentro das requisições HTTP).

O GSI3, da mesma forma que na versão anterior, permite o estabelecimento de um contexto de segurança entre duas entidades, porém ao invés de utilizar um protocolo da camada de transporte (TLS), utiliza módulos do *WS-Security* (*WS-SecureConversation* e *WS-Trust*). Os módulos *XML-Signature* e *XML-Encryption* são responsáveis pela proteção das mensagens.

4.5 Outras Plataformas

4.5.1 Modelo de Segurança do Legion

O projeto Legion consiste em uma plataforma de computação distribuída multi-organizacional com o objetivo de prover ao usuário a impressão de estar utilizando um único computador, extremamente poderoso e bem equipado [LEWIS 1996] [GRIMSHAW 1994].

O Legion é um sistema baseado no modelo orientado a objetos, portanto todas as entidades participantes do sistema são modeladas como objetos. Cada objeto implementa um conjunto de métodos e possui um estado. No Legion, os objetos podem ser encontrados em dois estados: ativos ou inertes. No estado ativo executam seus papéis, atendendo a invocações de outros objetos, e objetos no estado inerte permanecem com seus estados inalterados.

Cada entidade no Legion é identificada por um identificador único que cada objeto possui (LOID – *Legion Object Identificator*), que contém vários campos, incluindo informações de segurança como, por exemplo, um certificado x.509. Desta forma, os objetos podem enviar mensagens cifradas e fazer verificações de integridade e autenticidade nas mensagens recebidas. A chave privada de uma entidade é mantida no disco rígido, ou dispositivos móveis, como *smart cards*. Para facilitar a interação humana com os objetos, o Legion mapeia os LOIDs em *strings* através de um serviço de diretório chamado “espaço de contexto” (*context space*).

Ao invés de se propor a fornecer fortes esquemas de segurança, o Legion se baseia em três princípios [LEWIS 1996]:

(1) Ser tão minucioso quanto possível sobre o nível de confidencialidade que um usuário pode ter;

(2) Tornar a confidencialidade “boa o suficiente”, e “barata o suficiente” para um grande grupo de usuários; e

(3) Fornecer um contexto que permite ao usuário ganhar níveis de confidencialidade adicionais necessários, estando ciente do aumento proporcional do custo.

No Legion, os usuários são responsáveis pela própria segurança. A responsabilidade de determinar a política a ser imposta, e o quão rígida será esta imposição, é do próprio usuário. A seguir serão abordados alguns aspectos de segurança do Legion, como autorização, autenticação e política de privacidade.

4.5.1.1 Controle de Acesso

Um objeto Legion pode representar um arquivo, um serviço, um dispositivo ou qualquer outro tipo de recurso. O controle de acesso no Legion não é centralizado; cada objeto é responsável por impor sua própria política de controle de acesso.

Cada chamada de método para um objeto passa por uma camada chamada *MayI*, que decide se o acesso deve ser fornecido ou não, baseado em uma política que esta camada implementa. A camada *MayI* pode simplesmente permitir todos os acessos, ou pode utilizar uma lista de controle de acesso (ACL) e checagem de credenciais. Neste último caso, pode-se atribuir ACLs para cada método de um objeto. Para cada método são utilizadas duas listas: a lista de permissões (*allow*) e a lista de exclusão (*deny*), que contêm os LOIDS dos solicitantes que podem (ou não podem) acessar o método. Quando uma chamada de método é recebida, as credenciais recebidas com a chamada são verificadas pela camada *MayI* e comparadas com as listas *allow* e *deny* [FERRARI 1999].

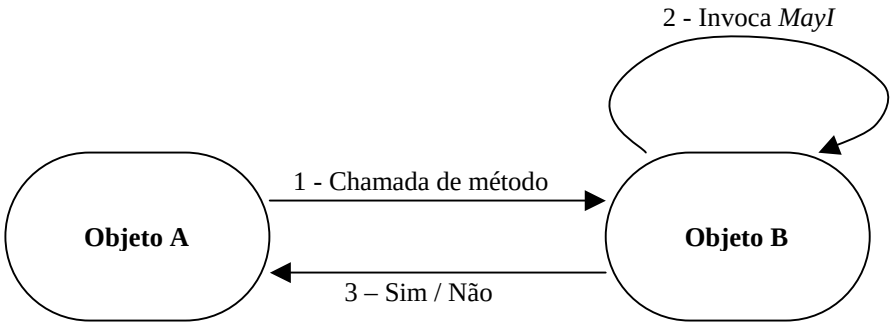

Figura 4.5 – Decisão de acesso a um objeto Legion [WULF 1996]

Para aumentar a segurança do processo, o tempo de vida de uma credencial é limitado. Caso este tempo seja insuficiente para finalizar a tarefa, a credencial pode ser renovada através do objeto de renovação de credenciais (*Credential Refresh Object*) do usuário.

4.5.1.2 Autenticação

Quando um determinado objeto deseja utilizar os serviços de outro objeto, ele deve enviar uma credencial de autenticação junto com a requisição. Esta credencial se aplica apenas a esta chamada de método impossibilitando, assim, a captura da credencial para maliciosamente atribuí-la a outra chamada.

Para prevenir que o objeto solicitado não se aproveite de uma credencial de autenticação para acessar indevidamente outro objeto, estas credenciais incluem o identificador (LOID) do objeto alvo.

4.5.1.3 Privacidade e Integridade

Quando um objeto Legion efetua uma chamada de método, esta se divide em várias mensagens, às quais se aplicam os procedimentos de criptografia (confidencialidade e integridade). Esta mensagem pode ser enviada nos modos: sem proteção, privado, ou protegido, onde no modo privado a mensagem é cifrada, e no modo protegido existe apenas o resultado de uma função *hash* para verificação da integridade.

Para garantir que as credenciais não são enviadas sem proteção, o modo padrão (caso não seja escolhido nenhum modo em particular) é o protegido. Além disso, cada mensagem contém um "rótulo" cifrado (gerado automaticamente no momento da chamada do método) que equivale àquele trabalho computacional especificamente. Isto é feito para relacionar as mensagens com a chamada de método que as gerou.

4.5.1.4 Legion-G

O Legion-G é uma combinação entre o Legion e o Globus para que cada plataforma compartilhe os benefícios que a outra oferece. As funcionalidades que o Legion pode obter através do Globus são: a interface de troca de mensagens de alto desempenho, os protocolos GRAM (*Grid Resource Allocation and Management*) e os protocolos de segurança do Globus. Os usuários Globus se beneficiam com o sistema de arquivos do Legion [WHITE 01], suas ferramentas gráficas para estudo de espaço de parâmetros, e com o seu modelo de orientação a objetos [HUMPHREY 2003].

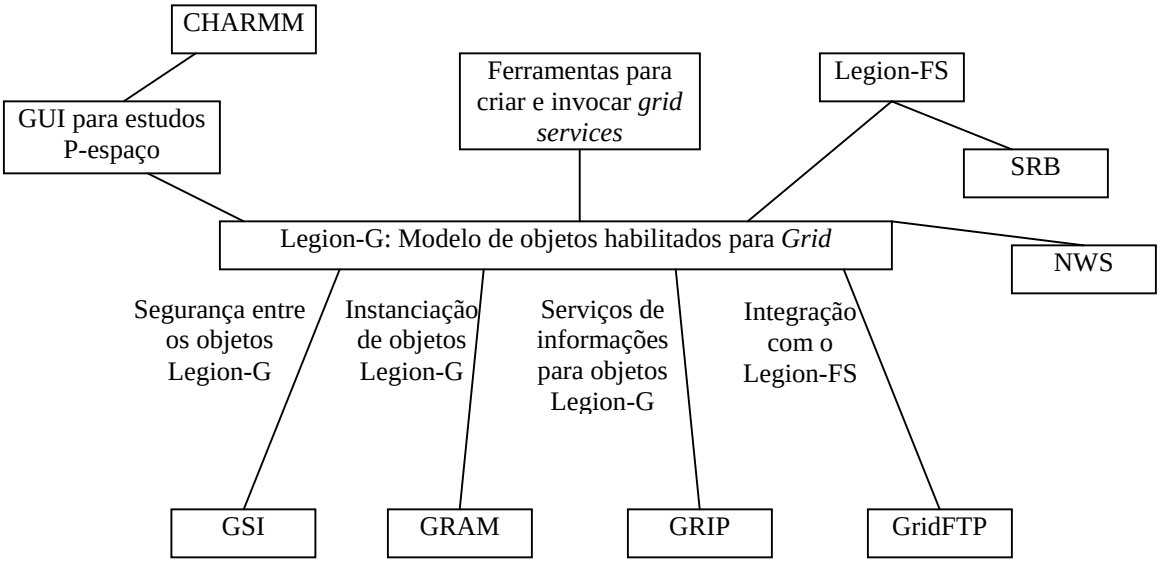

Figura 4.6 - Arquitetura do Legion-G [HUMPHREY 2003]

4.5.2 Projeto Condor

O projeto Condor surgiu no começo da década de 80, na universidade de Wisconsin, contrastando com o modelo de computação centralizada predominante na época. O conceito de computação distribuída do Condor tem como diferencial a possibilidade de cada participante poder definir o seu grau de participação no sistema, contribuindo tanto quanto possível ou desejado.

O projeto Condor é composto hoje por mais de 30 universidades e participa de várias pesquisas de computação em *grid* como *Grid Physics Network*, *International Virtual Data Grid Laboratory*, *TeraGrid*, *Nasa Information Power Grid* entre outras, sendo também membro fundador do *National Computational Science Alliance* (NCSA) e colaborador do projeto Globus [THAIN 2003].

No contexto prático do Condor, existem dois tipos de máquinas participantes: os computadores que gerenciam trabalhos (máquinas requisitantes) e os computadores destinados à gerência de recursos (máquinas executantes). Para exercer cada um dos papéis, existe um software para cada tipo: o Condor e o Condor-G.

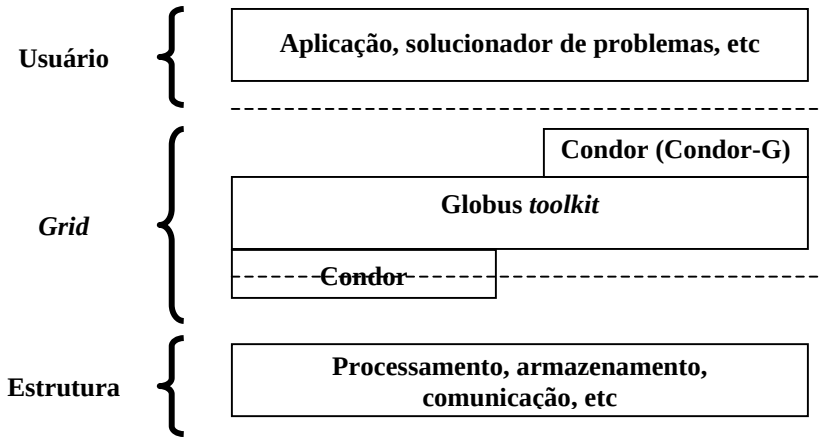

Figura 4.7: União entre o Condor e o Globus. Fonte: Condor and the Grid [THAIN 2003].

O Condor *High Throughput Computing System*, ou simplesmente Condor, é um sistema de gerenciamento de recursos que organiza as requisições de trabalhos ao serem recebidas, determinando a máquina que será utilizada e o momento da execução. O Condor-G utiliza a infraestrutura de segurança do Globus e o acesso padronizado a uma grande quantidade de recursos disponíveis, já que o objetivo de um ambiente computacional de alto rendimento (high-throughput) é fornecer grande quantidade de poder computacional utilizando efetivamente todos os recursos disponíveis na rede.

A figura 4.7 ilustra o *middleware grid* composto pelo *Condor High Throughput Computing System*, que é responsável pelo planejamento e agendamento de utilização dos recursos, o Condor-G, que é utilizado como um meio de submissão segura e gerenciamento de trabalhos computacionais, e o Globus Toolkit, que serve como uma interface entre eles [THAIN 2003].

Os elementos fundamentais do Condor são: agentes, recursos e “combinadores” (*matchmakers*). O usuário envia a sua solicitação a um agente, que é responsável por

armazenar a solicitação em um meio de armazenamento persistente enquanto busca recursos disponíveis para executar a tarefa. O agente e o recurso publicam informações no elemento combinador, que é responsável por apresentar recursos a agentes potencialmente compatíveis [THAIN 2003]. Uma mesma máquina pode executar instâncias de agentes e de recursos e é capaz de submeter e executar tarefas. Várias instâncias de um agente podem ser executadas na mesma máquina, por exemplo, cada usuário pode ter em execução o seu agente pessoal [THAIN 2003].

Agentes, recursos e elementos combinadores possuem, individualmente, a responsabilidade de impor a política dos seus donos. O agente impõe a política do usuário solicitante em recursos que são confiáveis e adequados para executar tarefas. Por sua vez, o recurso impõe as políticas da máquina em que o usuário é conhecido e confiável. Já o elemento combinador é responsável por impor políticas da comunidade como controle de admissão, ou seja, pode-se admitir ou rejeitar participantes baseando-se no seu nome ou endereço [THAIN 2003].

4.6 Considerações Finais do Capítulo

Com a virtualização dos serviços proposta pelo OGSA surgem requisitos de segurança para acessar um serviço. Os serviços de segurança devem também ser virtualizados, para fornecer uma maneira padronizada de segmentar componentes de segurança (autenticação, controle de acesso, etc) e fornecer maneiras padronizadas de habilitar a federação de múltiplos mecanismos de segurança.

A arquitetura de segurança do *grid* deve garantir que os serviços OGSA, quando invocados por um sujeito solicitando um serviço, estejam de acordo com as restrições de política, como especificado pelo ambiente de *host*.

Capítulo 5

Trabalhos Relacionados

Este capítulo tem como objetivo mostrar alguns trabalhos relacionados na área de segurança em *grid*. Os trabalhos aqui abordados tratam de aspectos diversos como autorização, gerência de política e controle de integridade.

5.1 Sistema PRIMA

O sistema PRIMA (*Privilege Management*) [LORCH 03] foi desenvolvido direcionado a fornecer um mecanismo de gerência de privilégios que suportasse um grau maior de detalhes na especificação das regras de controle de acesso, que podem ser delegadas de um usuário para outro, ou do administrador para um usuário.

O princípio de privilégio mínimo define que sujeitos devem possuir apenas o conjunto de privilégios mínimos para realizar uma determinada tarefa. Para alcançar tal premissa, algumas ações devem ser tomadas:

- Criação automática de contas de usuários, sem a intervenção do administrador do sistema. O tempo de existência destas contas deve ser limitado, de tal forma que não exija a atenção do administrador para encerrá-las.
- Refinamento da política de autorização. Tradicionalmente os mecanismos de autorização não expressam com grandes detalhes quem pode ou não acessar determinados recursos. Com mecanismos de maior expressividade pode-se detalhar melhor os direitos e limites de cada usuário.

A principal característica do sistema PRIMA é a descentralização do modelo através do compartilhamento de certos poderes do administrador com os usuários:

- Os próprios usuários podem delegar privilégios, detalhadamente, para outros usuários acessarem os recursos a eles pertinentes;
- O modelo permite que a confiança seja estabelecida entre dois usuários, independente se eles estão no mesmo domínio administrativo ou não.

O sistema PRIMA foi desenvolvido seguindo os princípios do privilégio mínimo e os princípios da separação de responsabilidades. Seus usuários têm interesse em proteger os próprios recursos e, portanto, liberam apenas o nível suficiente de acesso a terceiros.

Cada usuário possui um certificado de atributos padrão X.509, que além de relacionar um usuário com seus atributos, relaciona recursos com as políticas que os governam. As políticas de autorização se baseiam nos atributos dos sujeitos, e é desta forma que se garante que os usuários terão acesso apenas ao conjunto de recursos necessários para a realização das suas tarefas.

Um certificado de atributos pode conter um conjunto de privilégios e pode também estar relacionado a um conjunto de entidades. Isto permite que sejam criados certificados de atributos para a definição de papéis (seguindo o modelo RBAC [FERRAIOLO 2001]), onde um conjunto de privilégios é alocado a um papel específico e pode estar ligado a um grupo de usuários.

O sistema se integra ao Globus Toolkit como um módulo, agindo como um ponto de decisão de política (PDP), que se baseia nos recursos solicitados e nos atributos e políticas associados aos usuários requisitantes, para efetuar a decisão de controle de acesso.

Esta decisão é feita em dois níveis: primeiro é feito uma decisão de grosso modo (*coarse grained*), e caso seja positiva, é executado o mapeamento para a conta de usuário local. Só então é feita uma configuração detalhada, refinando as permissões de acesso.

Os elementos que caracterizam o modelo PRIMA são:

- Criador de privilégios: esta é uma ferramenta utilizada para um usuário “A” criar um certificado de atributo para outro usuário “B”, utilizando as credenciais de “A” para assinar o novo certificado. Os privilégios, especificados no certificado de atributos, são armazenados em um formato independente de plataforma, e consiste no tipo de privilégio, o nome do *host* onde os privilégios se aplicam, o caminho do recurso e uma lista dos direitos (separados por vírgulas) que o usuário possui;
- Criador de política: o criador de política é similar ao criador de privilégios, porém, os certificados de atributos criados contêm declarações de política em XACML emitidas pela autoridade de política. O dono do certificado de atributos é o recurso para o qual as políticas se aplicam;
- Combinador de privilégio: o combinador de privilégios permite que o usuário relacione privilégios (selecionados do certificado de atributos) com a sua identidade. Durante a criação de um certificado de *proxy*, o certificado de atributos do usuário é embutido (sendo que o certificado contém os privilégios selecionados previamente, e informações do usuário para ajudar no processo de validação). Embutindo o certificado de atributos dentro do certificado de *proxy*, garante-se um endosso seguro pelo usuário, sobre quais dos seus privilégios devem ser utilizados para um acesso específico;
- Revogador de privilégio: cada recurso no modelo PRIMA contém um revogador de privilégio, que é acionado automaticamente quando um conjunto de privilégios e/ou contas de usuários criadas dinamicamente expiram.

5.2 Matchmaking

A palavra *match*, em inglês, tem como significado “ser compatível”, ou “combinar”. No contexto que está apresentado abaixo, um *match* ocorre quando uma entidade é compatível com outra, e é feita a apresentação entre as duas entidades compatíveis. Já a palavra *claim* tem como significado literal “declaração” ou “afirmação”, e é utilizada neste contexto para denotar uma relação de trabalho entre as entidades compatíveis [RAMAN 1998].

A idéia básica do *Matchmaking* é que entidades que fornecem ou solicitam um serviço publiquem suas características e requisitos em *classads* (*classified advertisement*). Os *classads* consistem em um modelo de dados semi-estruturado utilizado para a especificação de informações estáticas e dinâmicas sobre um recurso, como um conjunto de restrições que controlam a alocação do recurso pelos solicitantes. Os *classads* podem receber solicitações de pesquisas, de forma que se obtenha informações essenciais para que se possa acessar o recurso [RAMAN 1998].

Em ambientes distribuídos, é importante que os recursos tenham sua política de utilização controlada através da definição de uma política. Vários tipos de restrições podem ser impostas, como uma faixa de horários em que os usuários podem acessar o recurso, número máximo de usuários acessando o recurso simultaneamente, controle de acesso baseado na identidade dos usuários, etc [RAMAN 1998].

A função do serviço *Matchmaker* é encontrar consumidores e provedores compatíveis através da verificação dos seus *classads*. Caso as restrições impostas por ambos estejam satisfeitas mutuamente, foi encontrada uma compatibilidade e então o *matchmaker* apresenta um ao outro. As entidades, então, entram em contato para estabelecer uma negociação e possivelmente entrar em um acordo de utilização do recurso. De acordo com [RAMAN 1998], este *framework* pode ser decomposto em cinco componentes:

- Especificação do *classad*, que define uma linguagem para expressar as características e restrições, e uma semântica de avaliação destes atributos;

- Protocolo de publicação, que define um conjunto de convenções a respeito do que o *matchmaker* espera encontrar em um *classad*, se o anúncio precisa ser incluído no processo de *matchmaking*, e como o *matchmaker* espera receber o anúncio do anunciante;

- Algoritmo de *matchmaking*, que define como o conteúdo dos anúncios e o estado do sistema são relacionados ao retorno do processo de *matchmaking*;

- Protocolo de *matchmaking*, que define como as entidades compatíveis são notificadas, e quais informações elas recebem, no caso de compatibilidade;

- Protocolo de invocação, que define as ações que as entidades compatíveis devem tomar para acionar a liberação de um serviço.

Uma característica desta abordagem que vale ressaltar é que o *matchmaker* apenas encontra as entidades compatíveis e as "apresenta". Isto não quer dizer que um consumidor esteja alocando (ou solicitando alocação) para um recurso.

5.2.1 *Classads*

Um *classad* é um modelo flexível e extensível para representação de dados, e pode ser utilizado para representar serviços e restrições de utilização de serviços, quando for solicitada uma alocação.

Um *classad* representa um mapeamento entre nomes de atributos e expressões. Os atributos podem ser tipos de dados simples, como inteiro, real ou *string*, e podem ser também expressões complexas construídas com operadores lógicos e aritméticos, e construtores de registros e de listas.

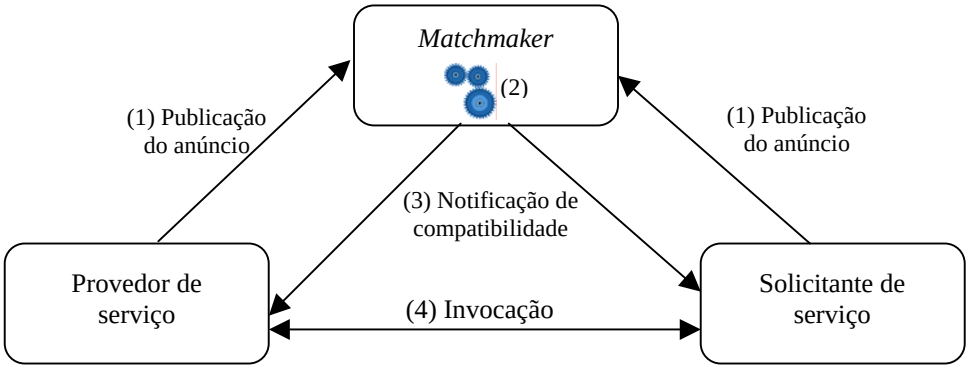

Figura 5.1 – Funcionamento do *Matchmaking*

A figura 5.1 ilustra o funcionamento do *matchmaker*. A descrição das três fases de busca de elementos compatíveis segue abaixo:

1. Cada entidade (solicitante e fornecedor de serviço) cria um *classad* descrevendo a entidade, e então o envia para o *matchmaker*;

2. O *matchmaker* invoca o algoritmo de busca de compatibilidades, avaliando as expressões em um ambiente onde cada *classad* possa acessar atributos de outros *classads*;

3. O algoritmo *matchmaker* invoca o protocolo *Matchmaking* para notificar os elementos envolvidos que são compatíveis (solicitante e fornecedor de serviço), e envia os anúncios publicados para eles.

```
[
Type = "Machine";
Activity = "Idle";
DayTime = 36107 // hora atual
// em segundos, desde meia-noite
KeyboardIdle = 1432; // em segundos
Disk = 323496; // kbytes
Memory = 64; // megabytes
State = "Unclaimed";
LoadAvg = 0.042969;
Mips = 104;
Arch = "INTEL";
OpSys = "SOLARIS251";
KFlops = 21893;
Name = "leonardo.cs.wisc.edu";
ResearchGroup = { "raman", "miron",
                  "solomon", "jbasney" };
Friends = { "tannenba", "wright" };
Untrusted = { "rival", "riffraff" };
Rank =
  member(other.Owner, ResearchGroup) * 10
  + member(other.Owner, Friends);

Constraint =
   !member(other.Owner, Untrusted)
    && Rank >= 10
      ? true
    : Rank > 0
     ? LoadAvg<0.3 && KeyboardIdle>15*60
    : DayTime < 8*60*60
     || DayTime > 18*60*60;
]
```

Figura 5.2.a Um *classad* descrevendo uma estação

```
[
Type = "Job";
QDate = 886799469;
// Submit time secs. past 1/1/1970
CompletionDate = 0;
Owner = "raman";
Cmd = "run_sim";
WantRemoteSyscalls = 1;
WantCheckpoint = 1;
Iwd = "/usr/raman/sim2";
Args = "-Q 17 3200 10";
Memory = 31;
Rank =
KFlops/1E3 + other.Memory/32;
Constraint =
other.Type == "Machine"
&& Arch == "INTEL"
&& OpSys == "SOLARIS251"
&& Disk >= 10000
&& other.Memory >= self.Memory;
]
```

Figura 5.2.b Um *classad* descrevendo uma tarefa

A figura 5.2 mostra as descrições de duas entidades em termos de *classads*. O *classad* da figura 5.2.a descreve uma estação de trabalho no Condor, e o *classad* da figura 5.2.b descreve uma solicitação de trabalho a ser executada em uma máquina que seja compatível com as características descritas. Alguns atributos que indicam compatibilidade estão destacados na figura 5.2, como indica o atributo *constraint* no *classad* da tarefa (figura 5.2.b):

Restrições da tarefa	**Características da máquina**	
other.Type == "Machine"	*Type = "Machine";*	OK
&& Arch == "INTEL"	*Arch = "INTEL";*	OK
&& OpSys == "SOLARIS251"	*OpSys = "SOLARIS251";*	OK
&& Disk >= 10000	*Disk = 323496; // kbytes*	OK
&& other.Memory >= self.Memory (*Memory* = 31;);	*Memory = 64; // megabytes*	OK
		Match!! (compatíveis!!)

Tabela 5.1: Verificação de compatibilidade entre a solicitação de tarefa e as características da máquina

5.2.1 Gangmatching

Um usuário Condor que compra licenças para vários pacotes de *software* e deseja executar a tarefa "X" que utiliza estes pacotes, precisa alocar tanto uma máquina como uma licença para executar a tarefa. Porém, os termos de uma licença podem apresentar algumas restrições (licenças que podem ser válidas apenas em algumas máquinas, ou em algumas sub-redes, por exemplo), tornando necessário que se trate licenças de *software* também como recursos.

Esta dependência multilateral (entre estação de trabalho, tarefa computacional e licença de *software*) torna o uso do *matchmaking* inadequado. O poder do modelo *matchmaking* está em gerenciar recursos cujas propriedades e dependências não são conhecidas no início. O *gangmatching* [RAMAN 2003] é uma implementação multilateral do *matchmaking* capaz de preparar grupos de candidatos, onde os tipos específicos de candidatos e suas interdependências são definidos apenas pelos próprios candidatos; não existe esquema central legislado.

Este modelo cria grupos (*gangs*) de *classads* ligando os *classads* individuais por uma operação de *matchmaking*. Ao contrário do *matchmaking* que produz um único par de

entidades compatíveis, o *gangmatching* produz uma lista explícita de pares compatíveis necessários, e permite que os *classads* acessem informações contidas em outros pares encontrados.

Cada *classad* possui um atributo *Ports*, através do qual é feita a ligação entre os *classads.* Os *ports* servem como interface para o *matchmaking,* permitindo que informações sejam fornecidas para candidatos, independente de como a informação é gerada. Desta forma, os *matchmakings* bilaterais ocorrem entre cada um dos atributos *Ports* dos *classads*, ao invés do *classad* inteiro. A figura 5.3 ilustra *classads* com o atributo *Ports*.

O elemento *Label* (rótulo) do atributo *Ports* substitui o atributo fixo *other* do *classad* utilizado no *matchmaking* bilateral, onde é necessário referenciar apenas a outra entidade candidata envolvida.

O atributo *Ports da* figura 5.3.a possui um *label cpu* para que as expressões do atributo *constraints* (restrições) possam referenciar uma estação de trabalho. As restrições ***“cpu.Type==”machine” && cpu.Arch==”INTEL” && cpu.OpsSys==”LINUX”&& cpu.Memory>=ImageSize”*** fazem referência aos dados contidos em um *classad* com atributo ***Type==”Machine”***, enquanto que as restrições ***“licence.Type==”License” && License.app==Cmd”***, utilizam o *label license* para referenciar um *classad* com atributo ***Type==”License”***.

```
[ Type = "Job";
   // alguns atributos comuns
   Owner = "raman";
   QDate = 'Mon Feb 28 14:22:22 2000
(CST) -06:00';
   Cmd = "run_sim";
   Ports = {
      [ // solicitação de uma estação de
trabalho
         Label = cpu;
         ImageSize = 28M;
         Rank = cpu.KFlops/1E3 +
cpu.Memory/32;
         Constraint = cpu.Type=="Machine"
&&
               cpu.Arch == "INTEL" &&
               cpu.OpSys == "LINUX" &&
               cpu.Memory >= Imagesize;
      ],
      [ // solicitação de uma licença
         Label = license;
         Host = cpu.Name; // cpu name
         Rank = 0;
         Constraint =
license.Type=="License" &&
                              license.App
== Cmd;
      ]
   }
]
```

Figura 5.3.a – C*lassad* representando uma tarefa no *gangmatching*

```
[ Type = "Machine";
   Activity = "Idle";
   KeybrdIdle = '00:23:12'; // h:m:s
   Disk = 323.4M; // mbytes
   Memory = 256M; // mbytes
   State = "Unclaimed";
   LoadAvg = 0.042969;
   Mips = 104;
   Arch = "INTEL";
   OpSys = "LINUX";
   KFlops = 21893;
   Name = "foo.cs.wisc.edu";
   Subnet = "128.105.175";
   Ports = {
      [ Label = requester;
         Rank = 1/requester.ImageSize;
          Constraint = requester.Type
       =="Job"
       &&
       requester.Owner!="riffraff" &&
       LoadAvg < 0.3 &&
       KeybrdIdle>'00:15'
      ]
   }
]
```

Figura 5.3.b – C*lassad* representando uma máquina no *gangmatching*

```
[ Type = "License";
   App = "sim_app";
   ValidHost = "foo.cs.wisc.edu";
   Ports = {
      [ Label = requester;
        Rank = 0;
       Constraint =   requester.Type=="Job" &&
       requester.Host==ValidHost
      ]
   }
]
```

Figura 5.3.c – C*lassad* representando uma licença no *gangmatching*

5.3 Considerações Finais do Capítulo

Para o propósito desta dissertação foram citados dois principais trabalhos relacionados: o sistema PRIMA e o *framework Matchmaking*. Cada trabalho possui características que levaram ao desenvolvimento da idéia apresentada no capítulo 6:

- O sistema PRIMA fornece ferramentas para usuários e administradores gerenciarem privilégios para os recursos aos quais têm autorização para acessar. Isto é feito através de certificados de atributos X.509, que carregam privilégios e declarações de políticas. Uma função para decisão de controle de acesso, na forma de um módulo de autorização para o *Globus Toolkit*, autoriza solicitações que se baseiam na combinação dos privilégios de acesso do usuário com as políticas dos recursos;

- O *Matchmaking*, utilizado no sistema *Condor*, apresenta um mecanismo para encontrar pares compatíveis (solicitante de uma tarefa → fornecedor de um serviço). Os *classads* são utilizados para descrever uma tarefa ou uma estação de trabalho e, para que uma tarefa encontre uma estação de trabalho compatível, deve enviar seu *classad* para o módulo *matchmaker*, onde outros *classads* também são armazenados e analisados, incluindo os *classads* das estações de trabalho. O *matchmaker* encontra as compatibilidades e informa as partes interessadas, enviando para elas o *classad* do elemento correspondente para posteriormente solicitar a alocação, através dos mecanismos pertinentes.

Analisando os pontos relevantes dos trabalhos citados, pode-se extrair pontos positivos e negativos. Um ponto negativo do *Matchmaking* é a utilização de uma estrutura de dados que não se baseia em nenhum padrão – os *classads*. Já um ponto positivo do *Matchmaking* é a separação entre os mecanismos de buscas de pares compatíveis e o módulo de alocação em si, pois desta forma pode-se ter opções de escolha, no caso da existência de vários módulos de alocação, sem perder as vantagens do sistema *matchmaker*.

O sistema PRIMA tem este nome por ser um sistema de gerenciamento de privilégios (*PRIvilege MAnagement*), que é composto por componentes da arquitetura de autorização que gerencia os privilégios de sujeitos, combinada com os mecanismos de gerenciamento de atributos e políticas dos recursos [LORCH 03]. PRIMA é um sistema descentralizado, possibilitando uma delegação de poderes de usuário para usuário.

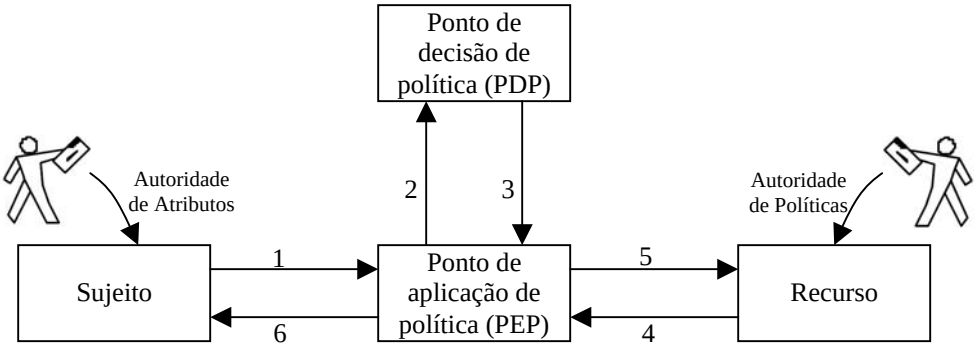

Figura 5.4 – Elementos básicos de um sistema de autorização no *grid* [LORCH 03]

No PRIMA um usuário não recebe permissão de acesso aos recursos baseado na sua identidade, e sim nos seus privilégios. Respeitando as particularidades de cada sistema, eles apresentam uma similaridade de idéias que difere principalmente na fase de alocação de recursos. Tais idéias foram base do tema desta dissertação, assunto tratado no capítulo 6.

Capítulo 6

Refinamento de Buscas de Serviços no *Globus Toolkit*

6.1 Introdução

O *Globus Toolkit* pode ser imaginado como na figura 6.1, uma sólida base de segurança sustentando os três pilares, equivalentes às áreas de: gerência de recursos, serviços de informação e gerência de dados, todas de grande importância e dependentes entre si. O foco abordado nesta dissertação é a área de serviços de informação (GIS – *Grid Information Service*), à qual o MDS pertence.

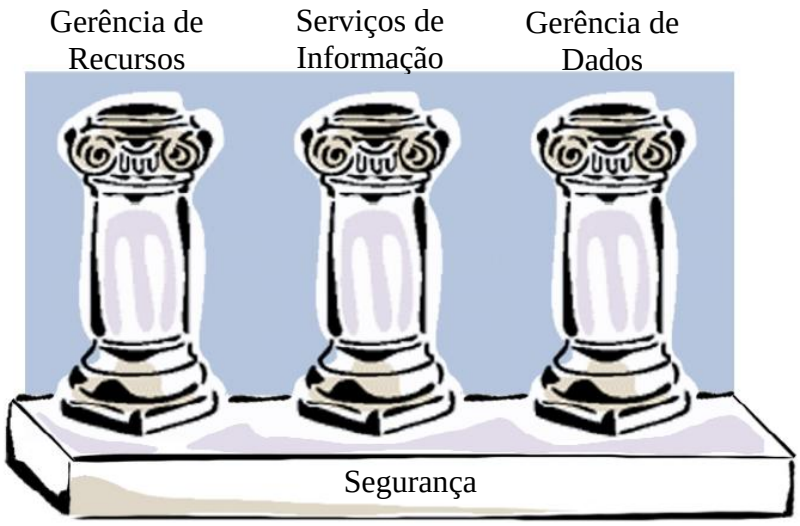

Figura 6.1 – Os três pilares fundamentais da infraestrutura do *Globus*

O MDS, como descrito no capítulo 3, é responsável por encontrar os recursos que estão disponíveis no instante em que foi realizada a pesquisa (baseado nos critérios especificados pelo sujeito), e as características dos recursos, como mostra a figura 6.2. Os requisitos especificados pelo usuário (menor custo, ou maior velocidade, por exemplo) serão levados em consideração pelo MDS (mais especificamente o serviço de índice) para encontrar o servidor adequado.

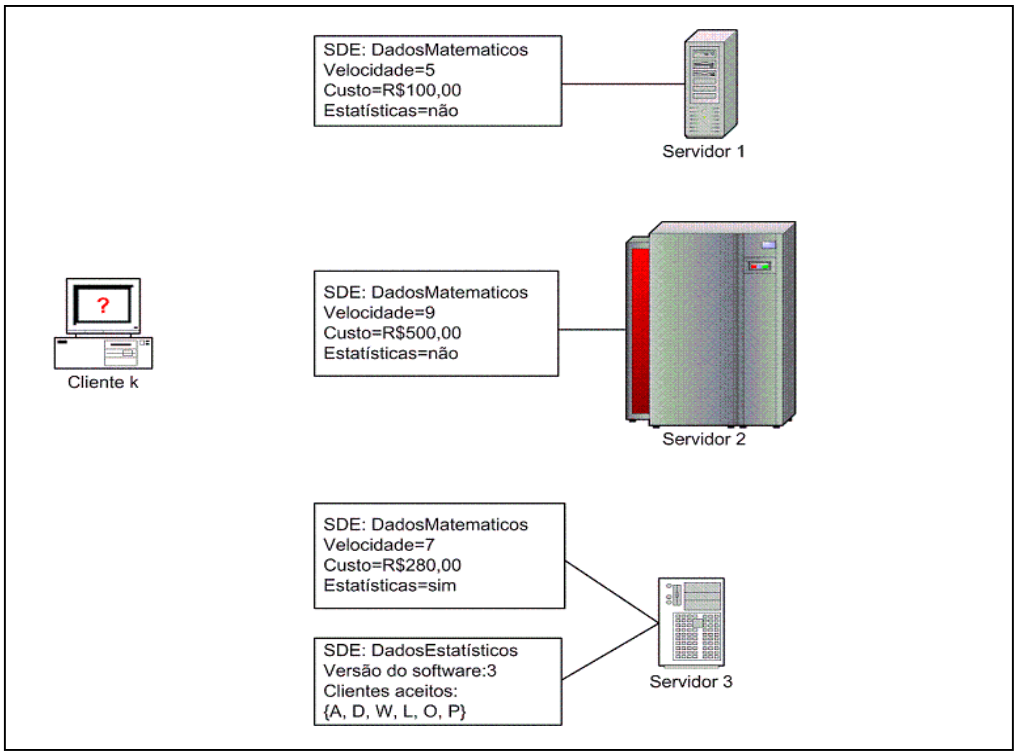

Figura 6.2 – Cliente com a possibilidade de escolher entre servidores que oferecem um mesmo serviço [GDP 2004]

O propósito do serviço de índice (*index service*) do MDS se resume em reunir dados sobre os serviços disponíveis em um local que pode ser consultado posteriormente pelas entidades requisitantes. Tipicamente, cada organização virtual (VO) possui um serviço de índice, a menos que esta VO seja muito ampla. Neste caso, cada *site* deve possuir seu próprio serviço de índice, e este deve estar registrado no serviço de índice da VO. Os dados de serviços encontrados são relacionados à performance do equipamento, custo do serviço, número de usuários conectados, características do serviço oferecido, entre outros.

A possibilidade de uma busca retornar uma quantidade muito grande de recursos depois de uma pesquisa no MDS motivou o desenvolvimento desta dissertação, que propõe um modelo para refinamento do resultado. Este filtro leva em consideração a política de segurança imposta pelo provedor do serviço e os atributos do sujeito, caracterizando um sistema que pode ser definido como sistema de pré-autorização.

O sistema de pré-autorização descrito neste capítulo é similar ao *framework Matchmaking* do sistema Condor. O objetivo é encontrar provedores de serviço cujas políticas impostas sejam compatíveis com os atributos do sujeito que busca um serviço. Porém, a tarefa de alocação do serviço continua sendo do módulo GRAM.

6.2 Descrição do Módulo de Refinamento de Buscas de Serviços

O que foi chamado de pré-autorização, no começo deste capítulo, é na verdade uma análise dos requisitos do serviço e dos atributos do usuário, com o objetivo de eliminar da lista (resultante da busca de serviços) os serviços aos quais o usuário não teria autorização para acessar.

Uma vantagem de se utilizar um módulo de refinamento baseado nas permissões do usuário, é a melhoria que se obtém na visualização dos resultados retornados, que é sensivelmente reduzida. Outra vantagem adquirida é a redução de interações com o módulo de alocação de recursos (GRAM), que é o responsável pela autorização dos usuários. Isto ocorre porque um sujeito pode escolher alocar um serviço, mas ao encaminhar a solicitação ao GRAM o acesso pode ser negado, fazendo com que o sujeito tente sucessivamente, até que encontre um serviço ao qual tem direito de acesso.

Cada provedor de serviço possui restrições que devem ser expressas através de uma linguagem de definição de políticas de segurança. De acordo com [FOSTER 2002], não é suficiente a existência de um GSR válido para garantir acesso a um *grid service*; as restrições da política local podem, também, negar a solicitação de um usuário.

Para alcançar o objetivo do módulo de refinamento aqui proposto, deve ser adotada uma linguagem de especificação de políticas de segurança que tenha capacidade de

detalhamento em mínimos detalhes (*fine grained*). Outra necessidade é a utilização de um mecanismo de expressão de atributos de um sujeito, para que a política do recurso seja capaz de analisar a solicitação e retornar uma permissão ou uma negação de acesso.

O módulo proposto neste trabalho é responsável pela verificação de compatibilidade entre as políticas do requisitante e do provedor de serviço, de maneira que retornem apenas serviços possíveis de serem acessados.

6.2.1 Definição de Atributos através de SAML

O padrão SAML (*Security Assertion Markup Language*) [SAML 2001] define um *framework* para troca de informações de segurança (mais especificamente de autenticação e autorização) entre entidades que, de alguma forma, interagem em um ambiente *grid*.

Através do SAML podem ser transferidas informações de identidade, autenticação, atributos e autorização, que são expressas através da combinação de um conjunto de blocos básicos, formando asserções.

Os provedores de serviço devem decidir quanto ao acesso (ou não) de um usuário aos seus serviços, baseados em um conjunto de declarações sobre este indivíduo. Tais declarações são chamadas de asserções. Um exemplo de asserção seria: “Este usuário é o João da Silva; seu endereço de *e-mail* é dasilva@xyz.com; foi autenticado no sistema através do mecanismo de senha” e possui o *status* “ativo”. Isto poderia ser utilizado como base para um provedor de serviço decidir fornecer ou negar acesso ao usuário João da Silva. As partes integrantes de uma asserção SAML são:

- Declarações de autenticação: são emitidas pela entidade que autenticou com sucesso um sujeito definindo quem emitiu a asserção, o sujeito autenticado, o período de validade e outras informações;
- Declarações de atributos: contêm detalhes específicos sobre o usuário em relação ao sistema como, por exemplo, que João da Silva possui *status* “ativo”;

- Declarações de decisão de autorização: identifica o que um usuário tem permissão de fazer.

O SAML é usado neste modelo para expressar os atributos do sujeito. O motivo pelo qual foi escolhido o SAML, ao invés do certificado de atributos X.509, é a estreita relação com a linguagem de definição de políticas XACML, além de ser um padrão baseado em XML.

As especificações do SAML e do XACML contêm algumas características projetadas especificamente para facilitar o seu uso combinado. O Perfil de Atributos XACML (*Attribute Profile*) [HUGHES 2005] define como os atributos SAML podem ser mapeados para atributos XACML. SAML fornece um *schema* para facilitar este mapeamento.

O perfil SAML V2.0 do XACML também define um tipo de pesquisa de decisão de autorização projetado especificamente para ser utilizada em um ambiente XACML. Este método de pesquisa estende o *schema* do protocolo SAML e fornece uma solicitação (*request*) e resposta (*response*) com o conteúdo exato das entradas (*inputs*) e saídas (*outputs*) definidas pelo XACML.

Duas características adicionais estendem o *schema* SAML. A primeira é o XACML *Policy Query*, que permite que o protocolo SAML seja utilizado para buscar a política XACML que pode ser aplicável para uma decisão de acesso [HUGHES 2005]. A segunda característica é que o envelope de asserções SAML pode ser utilizado para empacotar uma política XACML, tornando disponível para o XACML características como: o responsável por uma autenticação, o intervalo de tempo válido e a assinatura. Isto permite uma reutilização de código e de conhecimento entre SAML e XACML, evitando também problemas de redundância e inconsistência.

6.2.2 Definição de Políticas Através de XACML

Os conceitos descritos a seguir são importantes para o entendimento do modelo de controle de acesso do XACML [MOSES 2005].

- PAP – Ponto de Administração de Política (*Policy Administration Point*). É a entidade que cria uma política ou um conjunto de políticas;
- PEP – Ponto de Aplicação de Política (*Policy Enforcement Point*). É a entidade que executa o controle de acesso fazendo solicitações de decisão e aplicando decisões de autorização;
- PIP – Ponto de Informação de Política (*Policy Information Point*). É a entidade do sistema que age como uma fonte de valores de atributos;
- PDP – Ponto de Decisão de Política (*Policy Decision Point*). É a entidade do sistema que avalia uma política aplicável e toma a decisão de autorização.

O XACML (linguagem de marcação extensível para controle de acesso) é um padrão OASIS, assim como o SAML, cujo objetivo é definir a sintaxe e semântica de uma linguagem para expressar e avaliar políticas de controle de acesso [HUGHES 2005].

O XACML foi considerado uma escolha adequada para ser utilizado no modelo proposto por fornecer:

- Um método para combinar regras e políticas em um único conjunto de políticas <`PolicySet`>, que são aplicadas em uma solicitação de decisão;
- Um método para definição flexível de um procedimento pelo qual regras e políticas são combinadas;
- Um método para basear uma decisão de autorização em atributos do sujeito e do recurso;
- Um método para lidar com atributos multivalorados;

- Um conjunto de operadores lógicos e matemáticos nos atributos do sujeito, do recurso e do ambiente;
- Um método para manipulação de um conjunto distribuído de componentes de política, enquanto abstrai o método para localizar, retornar e autenticar os componentes de política;
- Um método para rapidamente identificar as políticas que se aplicam a uma determinada ação, baseada nos valores dos atributos do sujeito, recurso e ação;
- Uma camada de abstração que isola o criador da política dos detalhes do ambiente da aplicação;
- Um método para especificar um conjunto de ações que devem ser executadas em conjunto com o ambiente de política [MOSES 2005].

Os elementos básicos do XACML são: `<Rule>` (regra), `<Policy>` (política) e `<PolicySet>` (conjunto de políticas). O elemento `<Rule>` possui uma expressão *booleana* que pode ser avaliada isoladamente, mas não é suficiente para ser enviada para um ponto de decisão de acesso. As `<Rule>´`s podem existir isoladamente apenas nos PAPs XACML (Pontos de Administração de Políticas) formando um repositório destas unidades básicas de gerenciamento que podem ser reutilizadas em políticas posteriormente [MOSES 2005].

O elemento `<Policy>` contém um conjunto de elementos `<Rule>` e um procedimento que é especificado para combinar os resultados da avaliação. O `<Policy>` é o elemento básico que pode ser enviado para um PDP e, portanto, forma a base para uma decisão de autorização [MOSES 2005].

O elemento `<PolicySet>` contém um conjunto de elementos `<Policy>` ou outros elementos `<PolicySet>`, e contém um procedimento especificado para combinar os resultados desta avaliação. Com o uso do `<PolicySet>` pode-se combinar políticas separadas em uma única política combinada [MOSES 2005].

Os resultados individuais das avaliações das regras isoladas são combinados através de um algoritmo de combinação de regras, para finalmente chegar a uma decisão de autorização. O mesmo ocorre com o algoritmo de combinação de políticas. Os algoritmos de combinação padrão são definidos por [MOSES 2005]:

- *Deny-overrides*: caso um dos resultados (de uma regra ou de uma política) seja *deny* (negado), o resultado do algoritmo será *deny*, não importando o resultado da outra regra (ou política);
- *Permit-overrides*: da mesma forma que o *Deny-overrides*, se um dos resultados for *Permit* (permissão), o resultado do algoritmo será *Permit;*
- *First-applicable*: neste caso, o resultado combinado é igual ao resultado da avaliação do primeiro elemento `<Rule>`, `<Policiy>` ou `<PolicySet>` na lista de regras, cujo *target* (alvo) é aplicado à requisição de decisão;
- *Only-one-applicable*: este algoritmo é utilizado apenas em políticas. Quando exatamente **uma** política é aplicável, o resultado do algoritmo de combinação é igual ao resultado da avaliação da única política (ou conjunto de políticas) aplicável.

A figura abaixo demonstra uma regra simples em um arquivo XACML, informando que um usuário que utiliza um endereço de email do domínio ufsc.br tem direito de acesso irrestrito a qualquer recurso na rede.

```
(...)
<Rule
RuleID=”urn:oasis:names:tc:xacml:2.0:exemplo:RegraSimples1”Effect:”Permit”>
<Description>Qualquer pessoa com um email que pertença ao domínio ufsc.br pode executar qualquer
operação em qualquer recurso</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”>ufsc.Br
          </AttributeValue>
          <SubjectAttributeDesignator AttributeID=”urn:oasis:names:tc:xacml:1.0:subject:subject-id”
              DataType=”urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”/>
       </SubjectMatch>
    </Subject>
  </Subjects>
 </Target>
</Rule>
(…)
```

Figura 6.3 – Exemplo de uma regra na linguagem XACML

6.2.3 Modelo Proposto

No modelo geral do MDS, o módulo de refinamento proposto está posicionado entre o usuário e o serviço de descoberta e monitoramento. As solicitações de buscas são enviadas para o refinador, que por sua vez realiza a busca, filtrando os resultados.

A figura 6.4 mostra o procedimento de pesquisa de serviços com a utilização do módulo de refinamento. Para o usuário a diferença é quase inexistente, a pesquisa será encaminhada da mesma forma como se fosse para o MDS.

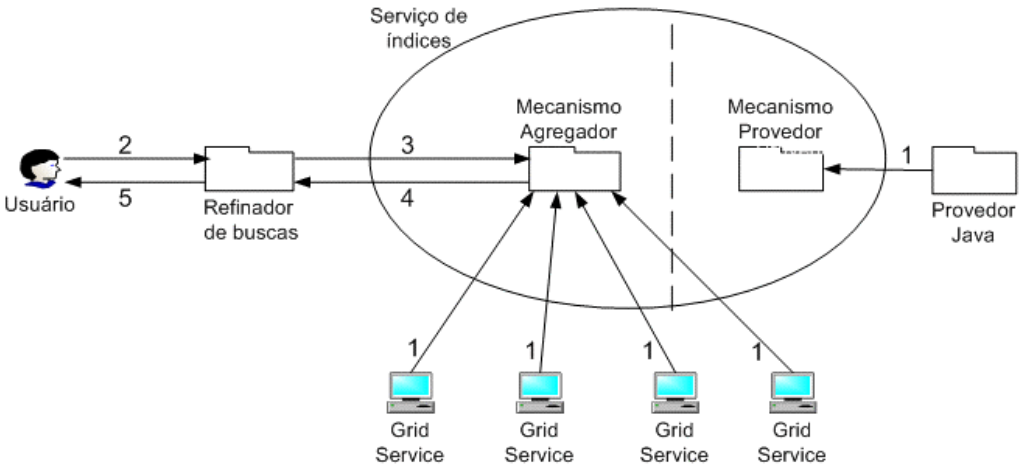

Figura 6.4 – Pesquisa de serviços com o módulo de refinamento

1. Os *grid services* publicam seus dados de serviços no serviço de índice;
2. O usuário envia seu pedido de pesquisa de serviço ao refinador de buscas;
3. A busca pelo tipo de serviço desejado é feita normalmente;
4. O resultado da busca é enviado ao módulo refinador;
5. Como resultado, o requisitante recebe uma lista de serviços (GSH) que estão em conformidade com a sua pesquisa e com os seus direitos de acesso;

O módulo de refinamento utiliza o SAML para a definição dos atributos do sujeito, que serão analisados no momento em que for realizada uma pesquisa. O ponto de decisão de política XACML (PDP) verifica se os atributos do sujeito seriam aceitos posteriormente pelo módulo de autorização do GRAM. Um usuário desejando encontrar um determinado serviço envia sua pesquisa para o módulo refinador, que por sua vez a encaminha para o MDS. A seguir, a figura 6.5 ilustra as interações entre os componentes deste modelo.

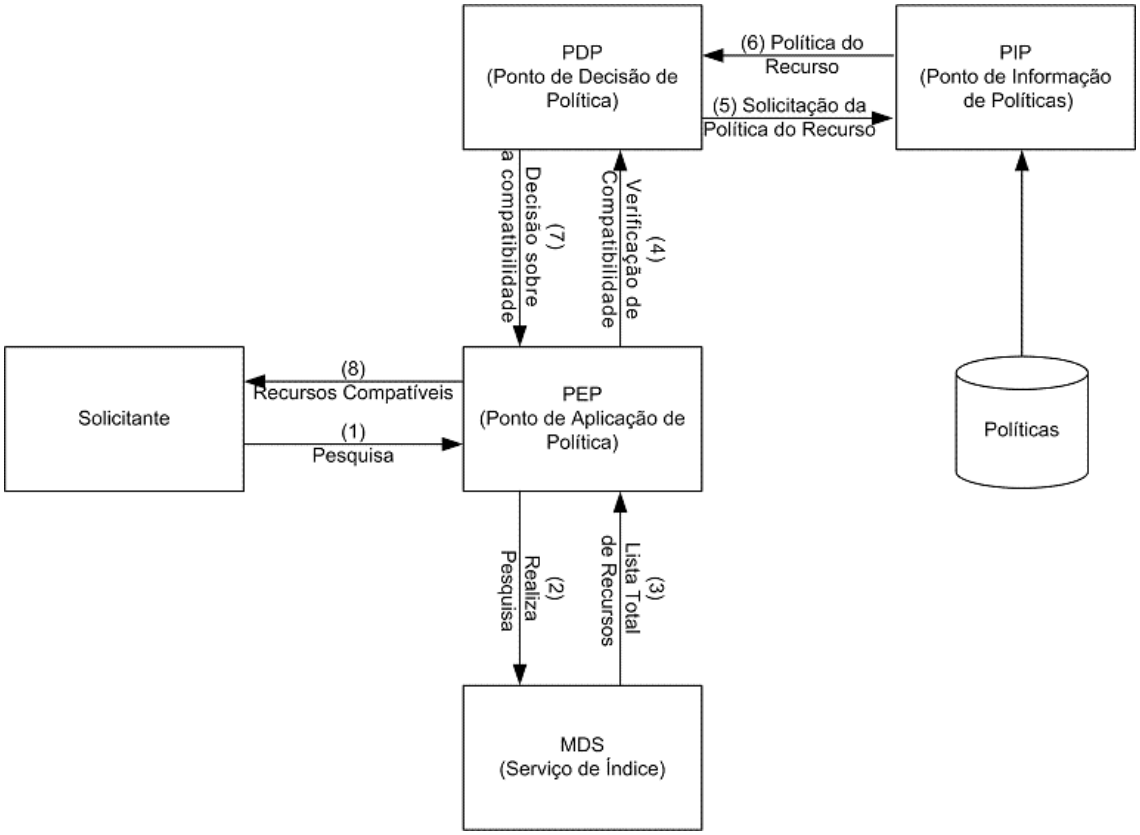

Figura 6.5 – Interações entre os componentes do modelo proposto para refinamento de buscas no MDS

(1) O sujeito realiza uma pesquisa passando como argumento um nome que descreva o serviço, ou uma expressão *Xpath* (linguagem de consulta em documentos XML);

(2) O PEP intercepta a solicitação de pesquisa, e acessa o SAML enviado pelo usuário, contendo seus atributos;

(3) O PEP recebe a lista de todos os serviços (apenas a referência GSH) que retornariam normalmente se o usuário acessasse diretamente o MDS;

(4) O PDP recebe a solicitação de decisão de acesso entre o sujeito que requisitou a busca e cada um dos serviços retornados do serviço de índice;

(5) O PDP solicita para o ponto de informação de política o conjunto de políticas (em XACML) do serviço;

(6) O PDP recebe as políticas do recurso, expressas em XACML, e analisa a compatibilidade com o arquivo SAML contendo os atributos do sujeito;

(7) O PEP recebe a decisão de autorização feita pelo PDP, aplicando a decisão descartando ou armazenando o serviço na lista filtrada;

(8) O usuário recebe a lista dos serviços que ele poderá acessar no momento da alocação, do ponto de vista da autorização.

Os atributos do usuário (especificados com o SAML) podem ser enviados no momento da solicitação de pesquisa, ou verificados no ponto de informação de políticas (PIP). No modelo da figura 6.5 o usuário encaminha seus atributos junto com a pesquisa, por questão de simplificação.

6.3 Implementação

6.3.1 Ambiente

O sistema operacional utilizado para a implementação do protótipo foi o *Linux Fedora Core 3*, onde foi instalado o *Globus Toolkit* 3.2.1. As ferramentas necessárias, além do próprio *toolkit*, envolvem: o *Java Development Kit*, o *Apache Ant*, o *JUnit*, um compilador "C", o Xalan, o *YACC/Bison*, e o *GNU tar*. Mais detalhes sobre as ferramentas complementares são descritos a seguir:

- Java [JAVA]: A linguagem de programação Java foi desenvolvida pela *Sun Microsystems*, e se baseia no poder das redes de computadores e na idéia de portabilidade entre plataformas (de *hardware* e *software)*. O Java foi projetado para executar programas com segurança em redes de computadores. A versão do Java que foi instalada foi a 1.4.2_07, versão recomendada para o *Globus Toolkit* 3.2.1;

- *Apache Ant* [APACHE ANT]: O *Ant* é uma ferramenta similar ao *make*, utilizada para compilar e montar os componentes que fazem parte de uma aplicação (código-fonte e bibliotecas, por exemplo) criando, como resultado, o código binário. A vantagem do *Ant*, em relação ao *make*, por exemplo, é que em vez de se utilizar comandos *shell*, o *Ant* utiliza arquivos de configuração baseados em XML. A versão do *Apache Ant* que foi instalada é a 1.5.4;

- *JUnit* [JUNIT]: O *JUnit* é um *framework* para realização de testes que devem ocorrer repetidamente. A versão instalada do *JUnit* é a 3.8;

- *GCC* [GCC]*:* Compilador *Ansi C* para o sistema *Linux;*

- *Xalan* [XALAN]*:* Processador XSLT que foi utilizado para converter um arquivo SAML para XACML;

- *YACC Bison* [YACC BISON]: Gerador de *parsers* de propósito geral;

- *GNU Tar* [GNU TAR]: Ferramenta de agrupamento (ou extração) de arquivos em um arquivo “tar”;

6.3.2 Protótipo

O *Globus Toolkit* pode ser visto como um conjunto de serviços “plugáveis”. A segurança no *Globus* também é fornecida através de serviços, o que simplifica uma situação onde seja necessária a utilização de um serviço de segurança diferente. Alguns dos serviços de segurança presentes no *Globus* são: serviço de processamento de credenciais, serviço de autorização, serviço de conversão de credenciais, serviço de mapeamento de identidades, serviço de auditoria e serviço de monitoramento e descoberta. O módulo de refinamento de serviços segue a idéia de orientação a serviços do *Globus* sendo, portanto, implementado também como um serviço.

O protótipo foi implementado em Java, linguagem utilizada no *Globus Toolkit* 3.x. Foram utilizadas ainda duas APIs externas, a OpenSAML, com a função de expressar os atributos dos usuários, e a SunXACML para especificação das políticas de segurança dos serviços.

Para facilitar a criação de políticas de seguranças, foi criado um programa que possibilita a criação de arquivos XACML através de uma interface gráfica amigável. Esse aplicativo pode tanto gerar arquivos de políticas, como também requisições de acesso. A figura 6.6 mostra a interface do programa.

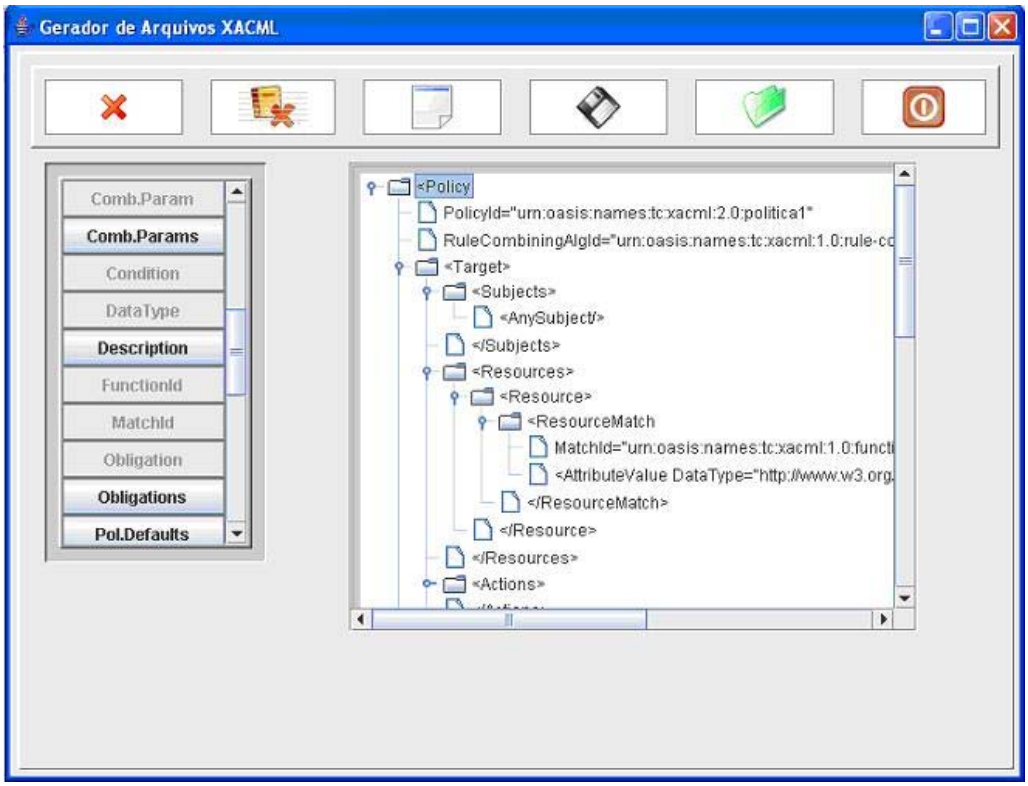

Figura 6.6 – Interface gráfica do aplicativo gerador de arquivos XACML

Um arquivo de política XACML contém vários elementos, que devem seguir o padrão da linguagem. Caso este padrão seja desrespeitado, o arquivo torna-se inválido. Este programa foi desenvolvido para evitar a inserção de um elemento inválido no arquivo, ou então um elemento válido, mas em uma posição errada.

Ao clicarmos em um nodo da árvore gráfica, são habilitados apenas os elementos que podem pertencer àquele nodo. Por exemplo, se o nodo “Policy” estiver selecionado, elementos como “Comb.Params” (parâmetros da política), “Description” (descrição da política) e “Obligations” (imposições da política) são habilitados. A figura 6.6 exemplifica esta funcionalidade.

Outra vantagem que se tem ao criar a política através deste aplicativo é que todos os elementos e parâmetros relevantes já estão disponíveis em caixas de seleção, sendo desnecessária a consulta freqüente à documentação do XACML.

Ao finalizar a configuração da árvore XACML, deve-se clicar no botão “Salvar”, (representado com a figura de um disquete na interface gráfica), e será criado o arquivo de política já validado, como ilustra a figura 6.7.

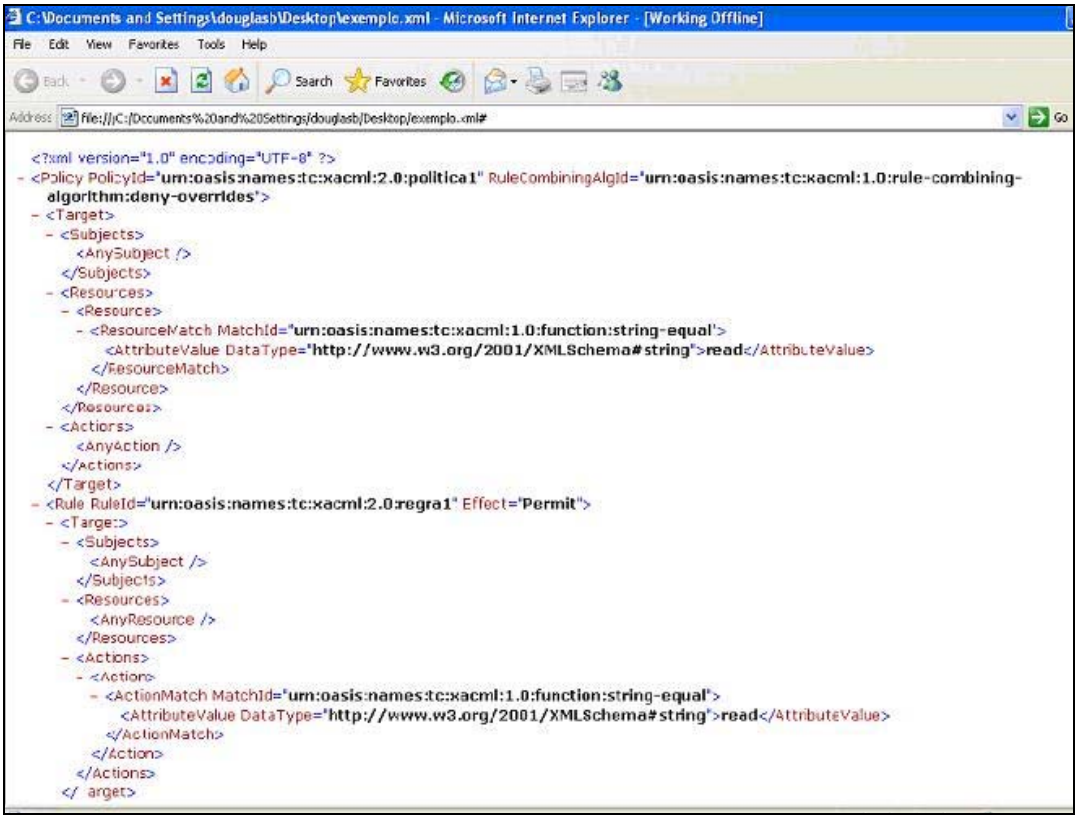

Figura 6.7 – Arquivo XACML gerado pelo programa

A pesquisa por recursos, que seria encaminhada para o MDS, agora é enviada para o módulo de refinamento, que vai funcionar como um *proxy*. Além de receber os dados para a realização da pesquisa, o serviço de índice recebe também um arquivo SAML (enviado através do protocolo SOAP) contendo os atributos do usuário. Este arquivo de atributos, quando chega no servidor, é transformado de SAML para XACML através de uma transformação XSLT, e então aplicado ao PDP para ser avaliado.

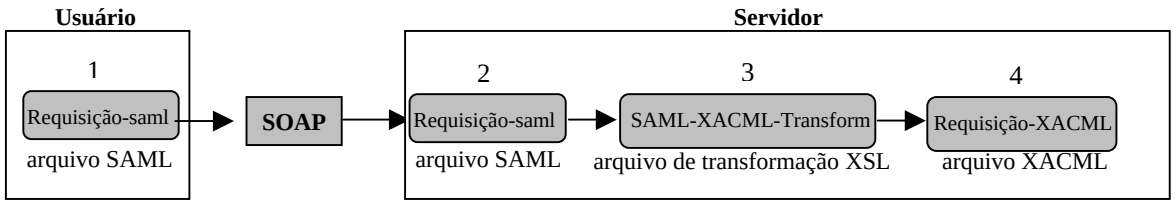

Figura 6.8 – Envio de um arquivo SAML através do SOAP e sua transformação para XACML

Munido dos dois arquivos necessários para exercer seu papel (arquivo de política e arquivo de atributos), o PDP é capaz de retornar um resultado com a decisão de acesso. Caso a decisão seja favorável, os atributos do usuário são considerados compatíveis com a política do recurso, e a referência para este recurso (seu GSH) estará disponível para o usuário. O resultado final é uma lista dos GSHs dos serviços que o usuário tem acesso. A seção 6.4 descreve um estudo de caso, onde este procedimento é ilustrado mais claramente.

6.4 Estudo de Caso

O cenário descrito neste estudo de caso considera um usuário que deseja acessar um servidor de arquivos com permissão de leitura. Para isto, o usuário precisa solicitar a pesquisa para encontrar um dos vários serviços existentes do tipo “servidor de arquivos”.

Cada serviço no *grid* possui uma política de segurança. Neste arquivo de política podem estar especificadas restrições como: domínio do qual o usuário faz parte, um intervalo de horários para acesso, tipo de acesso (leitura, escrita, execução, alteração ou criação), cargo que o usuário ocupa na instituição, etc. Esta política é expressa através da linguagem XACML.

A figura 6.9 ilustra a política de um dos serviços existentes no *grid*. Esta política declara uma regra que se aplica ao *login* do usuário. Caso as premissas sejam verdadeiras, o efeito é permitir o acesso (*Effect=“Permit”*). Neste caso, dois requisitos devem ser cumpridos: o horário da requisição deve ser maior ou igual a 9:00 e menor ou igual a 18:00.

```
- <Rule RuleId="LoginRule" Effect="Permit">
    <!-- Only use this Rule if the action is login -->
  - <Target>
    + <Subjects></Subjects>
    + <Resources></Resources>
    - <Actions>
      - <Action>
        - <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">leitura</AttributeValue>
            <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="AcaoServidor"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <!-- Only allow logins from 9am to 6pm -->
  - <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    - <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal">
      - <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">09:00:00</AttributeValue>
      </Apply>
    - <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal">
      - <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">18:00:00</AttributeValue>
      </Apply>
    </Condition>
</Rule>
```

Figura 6.9 – Arquivo de política XACML restringindo o horário de acesso

Ao fazer a pesquisa por recursos, o usuário envia um arquivo SAML contendo a descrição do tipo de recurso que deseja acessar, a especificação do tipo de acesso e seus dados pessoais (neste exemplo estamos utilizando apenas o nome, por questões de simplificação). Este arquivo SAML é enviado ao PEP através do protocolo SOAP, e ao chegar no servidor, é convertido em um arquivo *XACML Request* utilizando o XALAN. As figuras abaixo ilustram estes dois arquivos.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Request RequestID="47823081" MajorVersion="0" MinorVersion="28" IssueInstant="2006-03-01T14:50:00" xmlns="http://www.oasis-
  open.org/committees/security/docs/draft-sstc-schema-protocol-28.xsd" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:saml="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-28.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-open.org/committees/security/docs/draft-
  sstc-schema-protocol-28.xsdD:\MYDOCU~1\Standards\XACML\V12SCH~1\draft-sstc-schema-protocol-28.xsd">
 - <AuthorizationDecisionQuery Resource="//localhost:8080/axis/services/refinador">
  - <saml:Subject>
      <saml:NameIdentifier>Julius Hibbert</saml:NameIdentifier>
    </saml:Subject>
  - <saml:Request>
    - <saml:Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        <saml:AttributeValue>ServidorArquivo</saml:AttributeValue>
      </saml:Attribute>
    </saml:Request>
    <saml:Action>leitura</saml:Action>
  </AuthorizationDecisionQuery>
</Request>
```

Figura 6.10 – Arquivo SAML especificando o serviço a ser pesquisado e dados do solicitante

```
- <Request xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context cs-xacml-schema-context-01.xsd">
  - <Subject>
    - <Attribute AttributeId="Nome" DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>Julius Hibbert</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>ServidorArquivo</AttributeValue>
      </Attribute>
    </Resource>
  - <Action>
    - <Attribute AttributeId="AcaoServidor" DataType="http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>leitura</AttributeValue>
      </Attribute>
    </Action>
  </Request>
```

Figura 6.11 – Arquivo XACML Request, depois de aplicada a transformação no arquivo SAML acima

A transformação do arquivo SAML para XACML é necessária para que o arquivo de requisição esteja no mesmo formato do arquivo de política, podendo assim ser avaliado quanto à permissão. O resultado desta avaliação pode ser *Permit* (permissão concedida) ou *Deny* (permissão negada).

Desenvolvemos uma interface que recebe os dados digitados pelo usuário e gera um arquivo SAML expressando aqueles dados. Em seguida, este arquivo é enviado para o PEP (Ponto de Aplicação de Política) para que este realize a pesquisa no MDS. A figura 6.12 ilustra este processo:

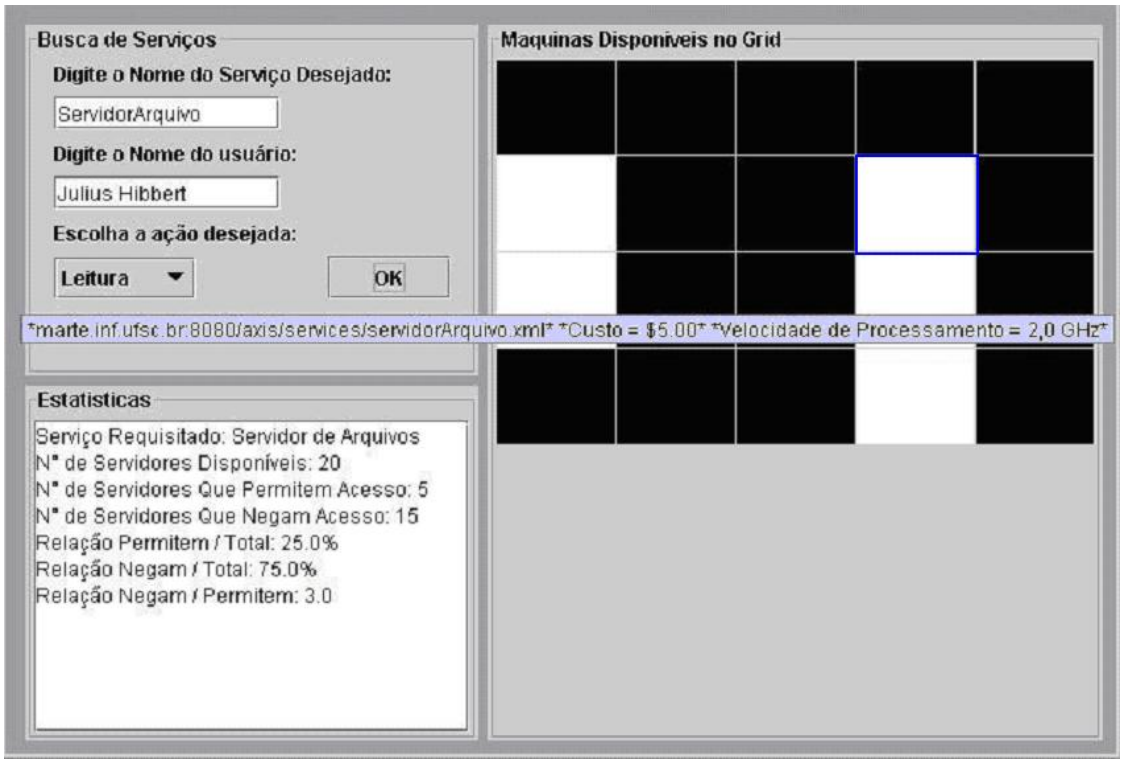

Figura 6.12 – Interface desenvolvida para o estudo de caso

O resultado é que apenas 25% dos servidores encontrados têm uma política compatível com a solicitação do usuário. Os servidores de arquivos encontrados que permitem acesso a usuários externos no período entre 9:00 e 18:00 são representados pelos quadros brancos (a pesquisa foi realizada às 16 horas). Os quadros pretos representam os servidores que o usuário não pode acessar. Os dados de serviço do servidor selecionado (com borda azul) são mostrados em um quadro flutuante, para que o usuário escolha o servidor que irá acessar baseado nestes dados.

O gráfico da figura 6.13 nos mostra a quantidade de serviços encontrados em cada horário do dia. Caso o usuário realizasse a busca às 18:00, o resultado seria de apenas um servidor válido. Se a pesquisa fosse realizada sem o refinador, a pesquisa resultaria em vários servidores, sendo que apenas um seria acessível.

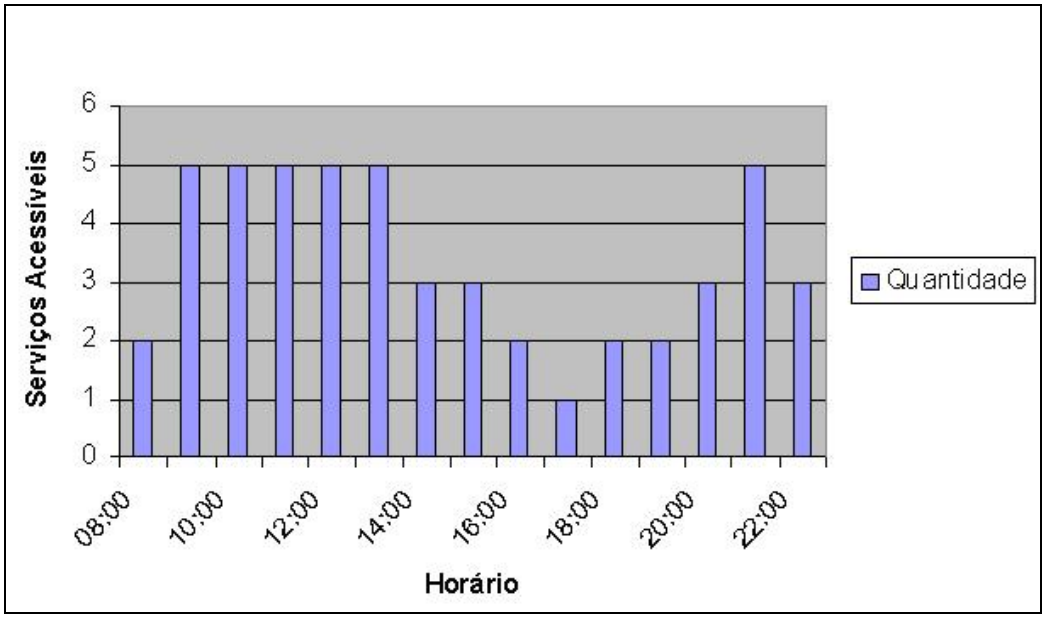

Figura 6.13 – Número de serviços acessíveis em relação aos horários

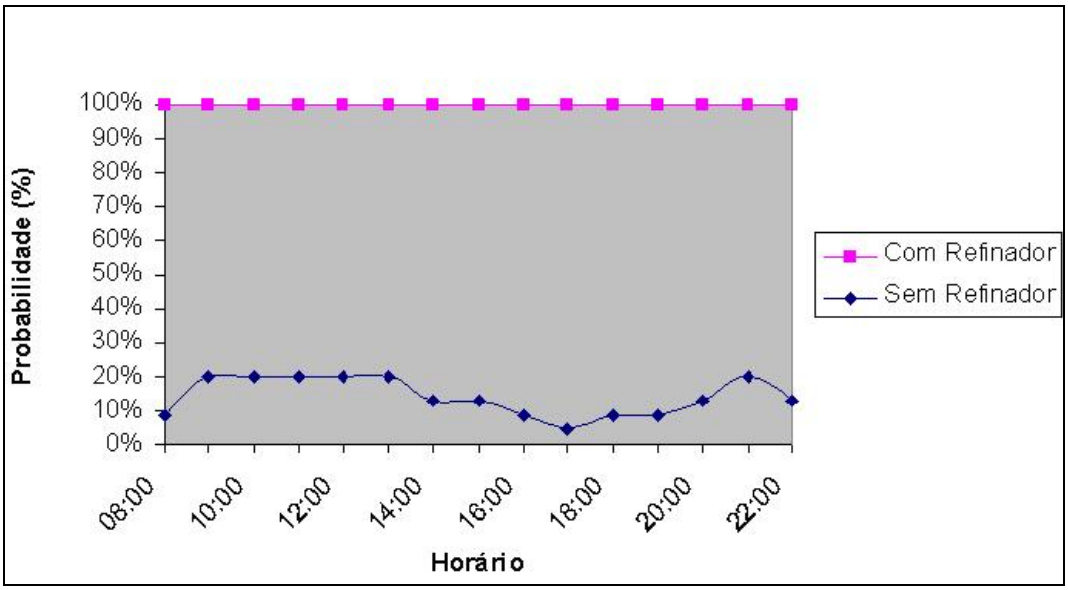

Figura 6.14 – Probabilidades de acesso permitido em relação aos horários

Como mostra a figura 6.14, a pesquisa com o refinamento aponta apenas resultados válidos, e o percentual de sucesso ao se tentar acessar um serviço da lista de resultados é de 100%, eliminando as interações desnecessárias com o GRAM.

Capítulo 7

Conclusão

Este trabalho apresenta uma extensão ao módulo MDS do *Globus Toolkit* 3.2.1, que filtra os resultados da busca de recursos baseado em uma pré-autorização, que leva em consideração os atributos do usuário solicitante e as políticas dos serviços procurados.

A escolha das linguagens SAML e XACML, ambas baseadas em XML, foi tomada para entrar em conformidade com o padrão utilizado na arquitetura OGSA, a qual as versões 3.x do Globus (e agora as versões 4.x) implementam. Além de apresentarem bastante compatibilidade entre si, foram desenvolvidas pela mesma organização, mesma comissão técnica, e praticamente simultaneamente. Apesar destas duas linguagens terem sido criadas com o objetivo de haver uma interação entre elas, elas não estão restritas a serem utilizadas apenas em conjunto. SAML define mensagens com formato XML para pesquisas (*queries*) e respostas (*responses*). As pesquisas são enviadas para um serviço de tomada de decisão, e as respostas são o resultado, que é enviado em forma de asserções para o solicitante. O serviço de tomada de decisões se baseia em políticas de segurança, que no caso do módulo de refinamento proposto, são representadas através de XACML.

7.1 Contribuições do Trabalho

De acordo com [NAGARATMAN 2002], um serviço *grid* deve ser capaz de definir ou publicar a sua Qualidade de Proteção (QoP) e seus atributos de segurança. Isto

possibilita que um solicitante descubra serviços baseados nas características de segurança do solicitante. Além disto, os solicitantes devem ser capazes de avaliar suas políticas de invocação baseados nos atributos de segurança dos serviços. É importante que os requisitantes estejam a par das políticas associadas a um serviço alvo.

Depois de uma extensa pesquisa na plataforma *Globus*, verificou-se que esta funcionalidade não está presente nas versões existentes hoje. O modelo existente no projeto Condor inspirou a criação do módulo refinador descrito nesta dissertação.

As vantagens obtidas são:

- a redução de interações do usuário com o módulo de alocação de serviços, pois o usuário saberá previamente quais recursos ele terá acesso. No Globus, o usuário recebe uma grande lista de recursos e solicita acesso a um deles, mas existe a possibilidade de acesso negado, e o usuário deverá escolher outro recurso na lista. Com o módulo refinador proposto, todos os serviços que retornam da busca do usuário serão acessíveis;
- maior facilidade para visualizar os serviços compatíveis com a sua busca. Como a busca resulta apenas nos serviços cujo acesso ao usuário é permitido, o resultado é uma lista mais enxuta, facilitando a visualização do usuário para encontrar o recurso mais adequado;
- facilidade de criação de políticas de segurança XACML com o aplicativo gráfico criado. Atualmente, para criar um arquivo de políticas XACML, deve-se recorrer inúmeras vezes à sua documentação para garantir a validade do arquivo. Com o aplicativo criado, as opções disponíveis em cada parte do arquivo seguem a especificação, garantindo a validação do arquivo durante a sua criação;

7.2 Considerações com Relação aos Trabalhos Relacionados

A idéia deste trabalho se baseou principalmente nos modelos *Matchmaking* e *Gangmatching*, ambos voltados para o projeto Condor. O serviço de busca de recursos nestes dois modelos inclui a verificação da política de segurança dos recursos. No Globus

Toolkit 3.2, plataforma escolhida para este trabalho, o serviço de busca de recursos não leva em consideração as políticas envolvidas, sendo que a verificação ocorre no momento da alocação do serviço.

No Condor, utiliza-se a representação dos recursos e suas políticas em termos de *classads*, um modelo de dados semi-estruturado desenvolvido para este fim. No modelo proposto nesta dissertação, para aplicar o conceito similar (de encontrar pares compatíveis entre sujeito e recurso) no Globus, são utilizados padrões baseados em XML, desenvolvidos pela OASIS (SAML e XACML), já que o Globus segue a arquitetura OGSA que também utiliza padrões baseados em XML.

7.3 Trabalhos Futuros

Existem duas formas de se realizar uma busca: de forma síncrona ou de forma assíncrona. A forma síncrona foi a adotada pelo módulo de refinamento, fornecendo um retorno imediato dos recursos correntemente disponíveis. Já a forma assíncrona é tratada no *Globus* com o mecanismo de notificação, onde o usuário especifica seus requisitos, e quando houver um serviço disponível que segue as características especificadas, o usuário recebe uma notificação.

Esta segunda maneira é a mais parecida com a do *Condor Matchmaking*, o qual efetua o algoritmo de busca de pares compatíveis e em seguida “notifica” as partes envolvidas. Esta é uma idéia a ser seguida, para complementar o módulo de refinamento proposto neste trabalho.

Referências Bibliográficas

[ANSI/INCITS 2004] AMERICAN NATIONAL STANDARD FOR INFORMATION TECHNOLOGY. **Role Based Access Control.** Draft 359, 2003. Disponível em http://csrc.nist.gov/rbac.

[APACHE ANT] http://ant.apache.org/

[BERNERS LEE 1998] BERNERS LEE, T., **Uniform Resource Identifiers (URI): Generic Syntax.** RFC 2396, 1998.

[BIESZCZAD 1998] BIESZCZAD, A; PAGUREK B; WHITE, T. **Mobile agents for network management.** IEEE Communication Surveys, vol. 1, n. 1, 1998.

[BLAKLEY 2001] BLAKLEY, Bob; MCDERMOTT, Ellen; GERR Dan. **Information security is information risk management.** In: Proceedings of the 2001 workshop on New security paradigms, Cloudcroft, New Mexico, pp. 97-104, 2001. ISBN:1-58113-457-6.

[BUTLER 00] BUTLER, R. et al. **A National-Scale Authentication Infrastructure**. IEEE Computer 33, No. 12, pp. 60-66, December 2000.

[CORNELLI 2002] CORNELLI, F. et al. **Choosing Reputable Servents.** In a P2P Network, In: Eleventh International World Wide Web Conference, Honolulu, Hawaii, United States pp. 376-386, 2002.

[COS] Site do projeto CosmoGrid. http://www.escience.cam.ac.uk/projects/cosmogrid/

[FERRAIOLO 2001] FERRAIOLO, David F. et al. **Proposed NIST standard for role-based access control.** In: ACM Transactions on Information and System Security

(TISSEC), Volume 4, Issue 3, ACM Press, New York, NY, USA, pp. 224-274, August 2001. ISBN:1094-9224.

[FERRARI 1999] FERRARI, Adam. et al. **A Flexible Security System for Metacomputing Environments.** In: High Performance Computing and Networking Europe (HPCN Europe 99), 1999.

[FOSTER 1997] FOSTER, Ian. e KESSELMAN, Carl. **Globus: A Metacomputing Infrastructure Toolkit**. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[FOSTER 1998] FOSTER, Ian. et al. **A Security Architecture for Computational Grids**. In: 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

[FOSTER 2000] FOSTER, Ian. **Internet Computing and the Emerging Grid**. Nature Web Matters, December 2000. Disponível em http://www.nature.com/nature/webmatters/grid/grid.html

[FOSTER 2001] FOSTER, Ian., KESSELMAN, Carl., and TUECKE, Steven. **The Anatomy of the Grid: Enabling Scalable Virtual Organizations**. In International J. Supercomputer Applications, 15(3), 2001. Disponível em http://www.globus.org

[FOSTER 2002] Foster, Ian., KESSELMAN, Carl., NICK, Jeffrey M. and TUECKE, S. **The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration**. Globus Project, 2002. Disponível em http://www.globus.org/research/papers/ogsa.pdf

[FOSTER 2004] FOSTER, Ian. e KESSELMAN, Carl. **The Grid: Blueprint for a New Computing Infrastructure.** Ed. Morgan-Kaufmann, 2004. 2ª edição. ISBN: 1558609334.

[FOSTER 2004a] FOSTER, Ian. et al., **Modeling Stateful Resources with Web Services, versão 1.1**, 2004. Disponível em http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

[GCC] http://gcc.gnu.org/

[GDP 2004] Sotomayor, Borja. The Globus Toolkit 3 Programmer's Tutorial http://gdp.globus.org/gt3-tutorial/

[GNU TAR] http://www.gnu.org/software/tar/

[GRIMSHAW 1994] GRIMSHAW, Andrew et al. **Legion: The Next Logical Step Toward a Nationwide Virtual Computer**. UVa CS Technical Report. CS-94-21, June 8, 1994

[GRIMSHAW 2004] GRIMSHAW, Andrew; TUECKE, Steven. **Grid Services Extend Web Services – A solid foundation for service consumer reliability**. Web Services Journal, 2004. Disponível em http://www.sys-con.com/webservices

[HUGHES 2005] HUGHES, John; MALER Eve. **Security Assertion Markup Language (SAML) V2.0 Technical Overview.** Working Draft 07, 13 July 2005. Disponível em http://www.oasis-open.org/committees/documents.php?wg_abbrev=security

[HUMPHREY 2003] HUMPHREY, Marty. **From Legion to Legion-G to OGSI.NET: Object-Based Computing for Grids.** In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IEEE Computer Society, Washington, DC, USA, pp. 207.1, 2003. ISBN:0-7695-1926-1.

[IBM] IBM. **What is grid computing** Disponível em http://www-1.ibm.com/grid/about_grid/what_is.shtml

[JAV 1999] Javasoft Java RMI Team. **Java Remote Method Invocation Specification**, versão 1.3.0. Sun Microsystems, 1999.

[JAVA] http://java.sun.com/

[JUNIT] http://www.junit.org/index.htm

[LANDWEHR 2001] LANDWEHR, Carl E. **Computer security.** International Journal of Information Security, v. 1, p. 3-13, 2001

[LEWIS 1996] LEWIS, Mike; GRIMSHAW, Andrew. **The Core Legion Object Model**. In: Proceedings of the Fifth IEEE Conference on High Performance Distributed Computing (HPDC5), Syracuse NY, pp. 551-561, August 1996.

[LINN 1997] LINN, John. **Generic security service application program interface**, versão 2. Internet RFC 2078, 1997.

[LORCH 03] LORCH Markus., et al. **The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments**. In: 4th Int. Workshop on Grid Computing - Grid 2003, Phoenix, AR, USA, November 17 2003.

[MIC 1996] Microsoft Corporation, DCOM Technical Overview, 1996.

[MOFFETT 1995] MOFFETT, Jonathan D. **Distributed Systems Security**. Encyclopaedia of Microcomputers, vol 15. New York: Marcel Dekker Inc.

[MOSES 2005] MOSES, Tim. **eXtensible Access Control Markup Language (XACML).** Version 2.0. OASIS Standard, 1 Feb 2005. Disponível em http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[MYER 2003] MYER, Thomas. **Grid computing: Conceptual flyover for developers**. IBM developer Works, 2003. Disponível em http://www-136.ibm.com/developerworks/

[NAGARATMAN 2002] NAGARATMAN, Nataraj et al. **Security Architecture for Open Grid Services**. GGF OGSA Security Workgroup, 2002.

[OMG 04] **Object Management Group. Common Object Request Broker Architecture: Core Specification**, versão 3.0.3, 2004.

[PEARLMAN 2002] PEARLMAN, Laura et al. **A Community Authorization Service for Group Collaboration**. In: Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

[PLA] **Platform Computing, The Evolution of the Grid – The Three Stages of Grid Computing**. Disponível em http://www.platform.com/grid/evolution.asp

[RAM 1998] RAMAN, Rajesh; LIVNY, Miron; SOLOMON, Marvin. **Matchmaking: Distributed Resource Management for High Throughput Computing. In: Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing.** Chicago, IL, July, 1998.

[RAMAN 2003] RAMAN, Rajesh; LIVNY, Miron; SOLOMON, Marvin. **Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching**. In: Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing, Seattle, WA, June, 2003

[SAMARATI 2000] SAMARATI, Pierangela; VIMERCATI; Sabrina C. **Access Control: Policies, Models, and Mechanisms Source.** In: Lecture Notes In Computer Science, Vol. 2171, London, UK, pp. 137-196, 2000. ISBN:3-540-42896-8

[SAML 2001] SAML Specification, 2001. Disponível em http://www.oasis-open.org/cover//draft-sstc-ftf3-saml-spec-00.pdf.

[SANDHOLM 2003] SANDHOLM, Thomas; GAWOR, Jarek. **Globus Toolkit 3 Core – A Grid Service Container Framework**, 2003. Disponível em http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf

[SIEBENLIST 2002] SIEBENLIST, Frank et al. OGSA **Security Roadmap**, 2002. Disponível em http://www.globus.org/ogsa/security/

[SKILLICORN 2002] SKILLICORN, David B. **Motivating Computational Grids.** In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 401, 2002. ISBN:0-7695-1582-7.

[SNELL 2001] SNELL, James; TIDWELL, Doug; KULCHENKO, Pavel. **Programming Web Services with SOAP.** Ed. O´Reilly & Associates, 2001. 1st edition, Ed. O´Reilly & Associates, 2001. ISBN: 0596000952

[SRINIVASAN 1995] SRINIVASAN Raj. Open **Network Computing RPC: Remote Procedure Call Protocol Specification**, RFC 1831, Versão 2, 1995.

[STALLINGS 1999] STALLINGS, William. **Cryptography and Network Security – Principles and Practice**. Prentice Hall, 1999. 2ª ed. ISBN: 0-13-869017-0.

[TANENBAUM 1997] TANENBAUM, Andrew S. **Redes de Computadores**. Ed. Campus, 1997. 3ª edição. ISBN: 85-352-0157-2

[THAIN 2003] THAIN, Douglas; TANNENBAUM, Todd; LIVNY, Miron. **Condor and the Grid.** In: Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003. ISBN: 0-470-85319-0.

[TUECKE 2002] TUECKE, Steven. **Grid Service Specification.** Draft 3, 2002. Disponível em http://www.gridforum.org/ogsi-wg.

[TUECKE 2003] TUECKE, Steven et al. **Open Grid Service Infrastructure Specification**, versão 1.0, 2003. Disponível em http://www.ggf.org/ogsi-wg

[WANG 2004] WANG, Hao et al. **Security Policy Reconciliation in Distributed Computing Environments.** In: IEEE Fifth International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), Yorktown Heights, New York, June 2004.

[WELCH 2003] WELCH, Von et al. **Security for Grid Services**. In: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03), IEEE Computer Society Washington, DC, USA, pp. 48, 2003. ISBN:0-7695-1965-2.

[WELCH 2003a] Welch, Von. et al. **OGSA Authorization Requirements**, 2003. Disponível em http://www.globus.org/ogsa/Security/

[WHITE 01] White, Brian S. et al. **LegionFS: A Secure and Scalable File System Supporting Cross Domain High Performance Applications**. In: Proceedings of the 2001 ACM/IEEE conference on Supercomputing, Denver, Colorado, pp. 59-69, 2001

[WOO 1996] WOO, Thomas Y. C.; LAM, Simon S. **Authentication for Distributed Systems, from Computer, January, 1992.** In: William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.

[WULF 1996] WULF, Wm A.; WANG Chenxi; KIENZLE, Darrel. **A new model of security for distributed systems.** In: Proceedings of the 1996 workshop on New security paradigms, Lake Arrowhead, California, United States, pp. 34-43, 1996. ISBN:0-89791-944-0.

[XALAN] http://xml.apache.org/xalan-j/

[YACC BISON] http://dinosaur.compilertools.net/