Grid Anywhere: Um middleware extensível para grades computacionais desktop

Fabiano Costa Teixeira

Grid Anywhere: Um middleware extensível para grades computacionais desktop

Fabiano Costa Teixeira

Orientador: **Prof. Dr. Marcos José Santana**

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. EXEMPLAR DE DEFESA

USP – São Carlos
Abril de 2012

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Teixeira, Fabiano Costa
T266g Grid Anywhere: Um middleware extensível para grades computacionais desktop / Fabiano Costa Teixeira; orientador Marcos José Santana. -- São Carlos, 2012.
164 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2012.

1. Sistemas distribuídos. 2. Grades computacionais. 3. Computação em nuvem. I. Santana, Marcos José, orient. II. Título.

Resumo

Esta tese de doutorado apresenta investigações, modelos e implementações de um *middleware* para grades computacionais denominado *Grid Anywhere*. Essa plataforma tem como objetivo viabilizar a construção de grades computacionais que permitam um maior número de provedores e consumidores de recursos. Para isso, são apresentadas soluções para gerenciamento de segurança, carregamento de aplicações, hospedagem de objetos, execução remota de métodos, desenvolvimento de aplicações e transporte alternativo de mensagens SOAP (utilizando o sistema de televisão digital interativa e encapsulando uma mensagem SOAP dentro de outro documento desse mesmo protocolo). Como aplicações da plataforma foram criadas duas grades computacionais com arquiteturas distintas. A primeira permite que um ambiente de compartilhamento de recursos possa ser utilizado como infraestrutura para prover plataforma como serviço (PaaS) para usuários convencionais (domésticos ou corporativos) em um ambiente de computação em nuvem. A outra arquitetura tem como foco o fortalecimento de grades computacionais *desktop* por meio da utilização de receptores digitais de TV (*set-top box*) como provedores de recursos onde a distribuição de objetos e as mensagens SOAP ocorrem por difusão. Os modelos foram validados por meio de testes reais feitos utilizando as respectivas implementações, o que demonstrou que são funcionais. Tais implementações disponibilizam produtos que cooperam com a inovação do desenvolvimento de aplicações para grades computacionais e também para outras categorias de sistemas distribuídos.

Abstract

This PhD thesis presents investigations, models and implementations of a computational grid middleware named Grid Anywhere. This platform aims at allowing the build of computational grid systems, which enable the increase of the number of participants as consumers and resource providers. In order to do this, new solutions are presented to manage security policies, load applications, host objects, execute remote methods, develop application and alternative transport of SOAP messages in a flexible way (using the interactive digital television system and inserting one SOAP message inside another document of this same protocol). As applications of this middleware, two different architectures of computational grids were created. The first one enables an environment to share resources that are used as infrastructure to provide platform as a service (PaaS) in a cloud computing system. The goal of the second architecture is to increase the computational power of desktop computational grid systems using digital television receivers (set-top box) as resource providers, where the distribution of objects and SOAP messages occurs via broadcasting. The models were validated by means of real tests using the respective implementations, which showed that the platform is functional. Such implementations provide software products that help the innovation and development of computational grid applications and also others types of distributed systems.

Agradecimentos

Em primeiro lugar agradeço a Deus pela vida a mim concedida, pela oportunidade de poder realizar meus estudos durante todos esses anos e pela proteção nos milhares de quilômetros percorridos durante a realização deste trabalho.

Agradeço a toda minha família pelo bom ambiente, a colaboração, o amor, o incentivo e todo o suporte que foram fundamentais em toda minha jornada.

De maneira muito especial agradeço ao meu orientador Prof. Dr. Marcos José Santana e à Profa. Dra. Regina Helena Carlucci Santana por toda a atenção, empenho e colaboração dedicados a mim durante esses anos.

Transpareço minha gratidão também aos meus amigos pessoais e à Lívia pela companhia e incentivo.

À Universidade de São Paulo agradeço pela oportunidade de poder ter a honra de fazer parte de uma das melhoras universidades do mundo.

Também direciono meus agradecimentos à CAPES pela bolsa concedida durante um período do doutorado.

Por fim, agradeço aos meus colegas da PUC Minas por toda a ajuda e incentivo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Símbolos

AAE	Application Execution Engine
ADSL	Asynchronous Digital Subscriber Line
API	Application Program Interface
ARIB	Association of Radio Industries and Businesses
ATSC	Advanced Television Standard Committee
BCEL	Byte Code Engineering Library
BML	Broadcast Markup Language
BOT	Bag of Tasks
CAN	Content Addressable Network
CC	Continuity Counter Field
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DASE	DTV Application Software Environment
DGET	Data Grid Environment and Tools
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSM-CC	Digital Storage Media – Command and Control
DVB	Digital Video Broadcasting
DVD	Digital Versatile Disc
ES	Elementary Stream
FTP	File Transfer Protocol
GEM	Globaly Executable MHP
GRAM	Grid Resource Allocation and Management
GSD	Gerenciador de Segurança de Domínio
GSH	Grid Service Handle
GSI	Grid Security Infrastructure
GSM	Global System for Mobile Comunication
GSR	Grid Service Reference
GUID	Globaly Unique IDentifier
GuM	Grid Machine Interface
GuMP	Grid Machine Provider Interface
HDTV	High-Definition Television
HOT	Hosted Objects Table
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
ID	Identification
IDL	Interface Description Language
IP	Internet Protocol
ISDB	Integrated System Digital Broadcast

ISDTV	International Standard for Digital Television
JPDA	Java Plataform Debugger Architecture
JSON	Java Script Object Notation
JXTA	Do ingles juxtapose
LRC	Local Replica Catalog
MDS	Monitoring and Discover System
MHP	Multimedia Home Platform
MPEG	Movie Picture Experts Group
NAT	Network Address Translation
OGSA	Open Grid Services Architecture
OGSA-DAÍ	Open Grid Service Architecture - Data Access and Integration
OGSI	Open Grid Services Infrastructure
P2P	Ponto-a-Ponto
PaaS	Platform as a Service
PAT	Program Association Table
PC	Personal Computer
PDA	Personal Digital Assistant
PMT	Program Map Table
PSI	Program Specific Information
QoS	Quality of Service
REST	Representational State Transfer
RFT	Reliable File Transfer
RLI	Replica Location Index
RLS	Replica Location Service
RMI	Remote Method Invocation
ROI	Remote Object Interface
RPC	Remote Procedure Call
RPV	Rendezvous Peer View
SaaS	Software as a Service
SBTVD	Sistema Brasileiro de Televisão Digital
SDTV	Standard-Definition Television
SGBD	Sistema de Gerenciamento de Banco de Dados
SLA	Service Layer Agreements
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SRDI	Shared Resource Distributed Index
TCP	Transmission Control Protocol
TLS	Transporte Layer Security
TS	Transport Stream
TV	Televisão
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UHF	Ultra High Frequency
URI	Uniform Resource Identifier

VCEG	Vídeo Coding Experts Group
VHF	Very High Frequency
WSBCL	Web Services Based ClassLoader
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
XML	Extensible Markup Language

Capítulo

1

Introdução

1.1 Considerações Iniciais

Com o passar dos anos a forma com a qual a computação é ofertada tem variado significativamente. No início, com os computadores de grande porte, uma única máquina era utilizada por inúmeros usuários, comumente corporativos, para realizar operações rotineiras como, por exemplo, geração de folha de pagamento.

A chegada dos computadores pessoais permitiu que a relação entre computadores e seres humanos se tornasse "um para um". No entanto, equipamentos da arquitetura PC (quando dotados de um sistema operacional *Time Sharing* como o Unix, por exemplo) ainda podiam ser compartilhados por diversos usuários que faziam uso de aplicações clientes para ter acesso ao sistema computacional, também determinando uma relação de "muitos usuários para um único computador".

Atualmente a relação entre usuário/computador já está invertida. Um único usuário possui inúmeros equipamentos computacionais que o auxiliam em suas tarefas de trabalho, comunicação e entretenimento. Entre estes equipamentos é possível destacar os computadores pessoais (incluindo *notebooks*, *netbooks*, etc), *tablets*, *smartphones*, TVs digitais (incluindo seus respectivos receptores) e automóveis. Isso colaborou para que um número muito grande de equipamentos computacionais existisse atualmente.

O perfil dos usuários destes equipamentos tem uma variação bastante grande. Enquanto alguns subutilizam seus equipamentos permitindo que uma parte de sua capacidade computacional permaneça ociosa, outro grupo necessita de computadores mais potentes para realizar suas tarefas. Observando este cenário é possível verificar

que a possibilidade do compartilhamento de recursos entre os usuários é bastante interessante.

Em meados da década de 90 um novo paradigma de computação distribuída chamada Grade Computacional surgiu com o objetivo de permitir que recursos heterogêneos pudessem ser compartilhados entre usuários remotos [1] [2] [3] [4] [5]. Atualmente esse tipo de sistema distribuído já vem sendo utilizado por inúmeras instituições e visa a auxiliar na resolução de problemas que demandam grandes potências computacionais e grandes volumes de dados.

Para isso, as instituições que colaboram entre si formam "organizações virtuais", onde os recursos ociosos podem ser anunciados, encontrados, requisitados e consumidos, obedecendo a regras previamente estabelecidas. Nessa arquitetura um único participante pode atuar em dois papéis: consumidor e provedor de recursos. No entanto, observa-se mais comumente a ocorrência de poucos provedores e muitos consumidores.

Com o objetivo de disponibilizar ambientes dessa natureza alguns *middlewares* foram desenvolvidos, implementados e apresentam bons resultados. Entre eles se destacam o Globus Toolkit [6] [7] e o OurGrid [8] [9] [10].

As "organizações virtuais" são compostas por participantes que possuem equipamentos de alta potência computacional como *clusters* e supercomputadores. Entretanto, usuários comuns espalhados por todo o planeta possuem computadores pessoais ociosos que podem contribuir com o processamento paralelo de aplicações. Com foco nesse tipo de recursos uma variação da grade computacional chamada "grades computacionais *desktop*" foi criada [11].

Uma grade computacional *desktop* normalmente adota um modelo arquitetural mestre-trabalhador [11]. Nesse modelo, uma instituição (mestre) que precisa de uma alta potência computacional para executar aplicações paralelas faz uso de computadores pessoais voluntários (trabalhadores), espalhados por todo o mundo, que fornecem percentuais de sua potência computacional para realizar esse processamento. Nesse modelo o mestre sempre atua no papel de consumidor e o trabalhador na função de provedor de recursos. Sendo assim, esse tipo de grade computacional comporta poucos consumidores e muitos provedores.

Projetos como o SETI@Home (cujo propósito envolve o processamento de sinais de radio vindos do espaço em busca de padrões não convencionais que indiquem a possibilidade de vida extraterrestre) [12], LHC@Home (responsável por processar os dados gerados pelo Grande Colisor de Hádrons do Cerne) [13], Rosetta@home (que trabalha na simulação de proteínas) [14], entre outros, são exemplos bem sucedidos de grades computacionais *desktop* que fazem uso de um *middleware* chamado BOINC, o qual foi desenvolvido pela universidade de Berkeley [11].

É possível observar que os esforços para permitir que equipamentos ociosos possam ser compartilhados já estão bastante evoluídos. Supercomputadores, *clusters*, dispositivos de armazenamento e computadores pessoais, entre outros, já podem ser utilizados por instituições que demandem potência computacional remota para suprir suas necessidades, principalmente nas áreas científicas e corporativas.

No entanto, muitos outros equipamentos existentes atualmente podem ser empregados, quando ociosos, para auxiliar nas tarefas de processamento de alto desempenho. Além disso, é possível pensar na utilização da grade computacional para executar aplicações comuns, permitindo que usuários convencionais possam fazer uso de recursos remotos para ampliar a potência computacional disponível para eles.

1.2 Motivação

No cenário atual, cada *middleware* para grade computacional tem um foco de atuação bem definido. É importante a existência de soluções que permitam que expansões possam ser realizadas de maneira mais simples, viabilizando o aumento do alcance das grades computacionais. Esse aumento pode ser aplicado aos provedores (permitindo que novas categorias de equipamentos compartilhem seus recursos) e aos consumidores (viabilizando novas aplicações que utilizam potência computacional remota ociosa) de recursos.

Observando o cenário atual nota-se a necessidade de expandir as grades computacionais em pelo menos dois sentidos: permitir que elas possam ser utilizadas para aumentar a potência computacional de equipamentos de usuários domésticos (ou mesmo corporativos) para a execução de aplicações comuns e aumentar o domínio de equipamentos provedores que compõem essa grade computacional.

Esta tese de doutorado está relacionada ao desenvolvimento de um *middleware* que seja altamente adaptável, possibilitando assim atender novos requisitos.

1.2.1 Grades Computacionais para Execução de Aplicações Comuns

Atualmente, usuários domésticos e corporativos executam suas aplicações em equipamentos de diversas naturezas (computadores pessoais, *tablets*, *smartphones*, etc). Com o passar dos anos a complexidade das aplicações tende a aumentar, exigindo que novos equipamentos substituam os atuais para atender à demanda de potência computacional das aplicações modernas.

Essa necessidade de atualizações de *hardware* para dar suporte à execução de novas aplicações gera custos aos usuários e pode dificultar o processo de inclusão digital. Por isso, torna-se interessante a possibilidade de permitir que usuários convencionais façam uso de recursos remotos para executar suas aplicações.

A adaptação do modelo de oferta de plataforma como serviço (PaaS – *Platform as a Service*) adotado pela computação em nuvem pode ser uma alternativa para esse tipo de problema. Na abordagem original, PaaS oferece ambientes de desenvolvimento, hospedagem e execução de aplicações em um provedor de serviços [15] [16].

De uma maneira semelhante, porém mais dinâmica que esta, uma aplicação poderia exportar parte dos objetos que a compõem para serem hospedados e executados remotamente. Com isso, uma conexão de rede eficiente pode-se auxiliar de maneira significativa o desempenho dessa aplicação em equipamentos mais simples.

Tecnicamente, a adoção da computação em nuvem auxiliaria na resolução do problema citado anteriormente, mas o modelo de negócios onde o usuário pode ser cobrado pelo que consome apresenta-se como um obstáculo, tendo em vista a intenção de se tornar a *infraestrutura* de hardware menos onerosa financeiramente ao usuário, se comparada àquela que ele precisaria ter originalmente.

Desta forma, torna-se interessante a possibilidade da utilização de uma grade computacional como forma de disponibilizar a infraestrutura necessária para que o conceito de plataforma como serviço seja ofertada sem custos ao usuário. A possibilidade de utilização conjunta de ambos os paradigmas é demonstrada por Foster *et Al* [15].

Uma grade computacional para essa finalidade apresenta um modelo contendo muitos provedores e muitos consumidores de recursos. Sendo assim, um mesmo usuário pode atuar em determinados momentos como cliente (utilizando computadores de outros usuários para viabilizar a execução de suas aplicações) e em outros momentos como servidor (permitindo que os outros participantes da grade façam uso de seus recursos locais).

Para que cenários como esses possam ser viabilizados, são exigidos esforços em diversos pontos. Entre eles, destacam-se:

- *Conectividade*: participantes podem estar localizados em situações onde não seja possível o estabelecimento de conexões (como, por exemplo, *firewalls*, *proxies* e NATs).
- *Arquitetura de grade computacional*: modelos atuais de grade computacional se preocupam com ambientes onde há uma organização virtual ou uma relação mestre-trabalhador. É preciso determinar como abrigar os participantes que colaboram entre si voluntariamente.
- *Hospedagem e execução de objetos*: ambientes que permitam a execução de objetos remotos, viabilizando a migração de estados e códigos executáveis em uma grade com essas características.
- *Segurança*: a existência de muitos consumidores na grade e a possibilidade de que os usuários provedores podem ser leigos em computação exigem um modelo de segurança flexível (preferencialmente transparente) e de simples utilização.
- *Modelo de programação*: é preciso que o desenvolvedor de aplicações para a grade computacional possua ferramentas para facilitar e agilizar este processo.

Uma vez que a hospedagem e execução de objetos em ambientes remotos é a chave principal para a construção desse tipo de grade computacional, a utilização do conceito de serviços *Web* torna-se muito atrativa. No entanto, os requisitos que normalmente são apresentados para que se possa ofertar um serviço podem ser inviáveis para um participante que deseja compartilhar seus recursos.

A necessidade de um servidor HTTP, por exemplo, pode ser um fator limitador em diversos potenciais provedores por alguns motivos, estando entre eles:

- A potência computacional gasta por esse servidor pode inviabilizar o provedor.
- Atuação como um agente passivo na arquitetura cliente-servidor pode não permitir que usuários com conectividade limitada atuem como provedores de recursos.

Por isso, de maneira a permitir que as vantagens já conhecidas dos serviços *Web* [17] possam ser desfrutadas pela grade computacional aqui proposta, novas formas de transporte do protocolo SOAP [17] precisam ser estudadas, propostas e implementadas.

1.2.2 Utilização de outras Categorias de Equipamentos na Composição de uma Grade Computacional Desktop

Além dos computadores pessoais, muitos outros equipamentos possuem processadores que podem ser utilizados na composição de uma grade computacional *desktop*.

Atualmente, o Brasil vem passando pela implantação do sistema brasileiro de televisão digital terrestre. Para que o telespectador possa fazer uso dessa tecnologia é preciso que ele possua um aparelho televisor preparado para esse padrão ou que adquira um receptor digital, também conhecido por *set-top box* [18].

O Brasil tem um potencial para atingir um número total de 80 milhões de receptores de TV digital nos próximos anos [19]. O conjunto de processadores formados por esses equipamentos pode ser utilizado para processamento paralelo de aplicações que podem ser enviadas via difusão (*broadcasting*), juntamente com áudio e vídeo.

A literatura apresenta esforços de um pequeno número de pesquisadores que já demonstram excelentes trabalhos e resultados referentes à viabilidade da utilização dos receptores de TV Digital para o processamento de aplicações paralelas [19] [20]. Esses trabalhos fazem uso da rede de difusão de sinal de TV para transmitir, via DSM-CC [21], aplicações que são executadas nos receptores dos usuários. No entanto, é possível ir além e permitir que chamadas remotas de métodos possam ser realizadas sobre esse ambiente.

Dessa forma, objetos podem ser transmitidos por uma emissora de TV aos seus receptores que posteriormente tem seus métodos invocados paralelamente. Para isso, torna-se interessante a utilização do protocolo DSM-CC como transporte para mensagens SOAP.

1.3 Objetivos

Esta tese de doutorado tem como objetivo principal apresentar investigações, propostas e implementações de um núcleo de um *middleware* para grades computacionais que permita, de maneira simplificada, que alterações possam ser realizadas para que novos equipamentos possam compartilhar seus recursos. Além disso, o *middleware* deve ser flexível o bastante para permitir também, sob demanda, que novas aplicações possam fazer uso de recursos compartilhados.

Os modelos desenvolvidos e as extensões já propostas e implementadas deverão ser suficientes para viabilizar:

- A execução de novas categorias de aplicações em uma grade computacional, incluindo aquelas utilizadas por usuários domésticos (ou corporativos).
- Aumento do número de provedores de recursos por meio da utilização de novas categorias de equipamentos, favorecendo aplicações paralelas de uso científico ou até mesmo corporativo.

Para que isso seja possível o *Grid Anywhere* deve ser apresentado como:

- Alternativa para prover uma grade computacional como infraestrutura de um ambiente de computação em nuvem para ofertar plataforma como serviço.
- Solução para fazer uso de processadores ociosos de receptores do sistema de televisão digital interativa.

Uma vez que um *middleware* dessa natureza requer o tratamento de diversos requisitos, um conjunto de soluções é apresentado. No entanto, cada requisito, bem como sua solução, pode ser encontrado em outros problemas da computação distribuída.

Por isso, a divisão dos trabalhos em módulos independentes (que se integram para a formação do *Grid Anywhere*) deve permitir que seus modelos e suas implementações possam ser utilizados para resolver problemas de outras categorias de sistemas que necessitem de:

- Ambientes seguros de execução com flexibilidade de configuração.
- Mecanismos para carregamento de classes com interoperabilidade e capacidade de execução em ambientes de conexão limitada.
- *Container* para hospedagem de objetos remotos
- Formas alternativas de transporte do protocolo SOAP.

1.4 Organização dos Capítulos

No capítulo 2 é feito um estudo sobre sistemas distribuídos que parte da implementação de SOA utilizando serviços *Web* WS-* e RESTFull até os paradigmas de computação em grade e em nuvem.

Tendo em vista que o foco desta tese é a construção de um *middleware* para grades computacionais, no capítulo 3 são apresentadas algumas soluções que permitem a construção de grades computacionais compostas por instituições que compartilham recursos entre si, grades computacionais *desktop* e ambientes de processamento paralelo que fazem uso de receptores digitais de televisão na composição de uma grade.

Para concluir a revisão da bibliografia é apresentado, no capítulo 4, um estudo sobre sistemas de televisão digital interativa, incluindo modelos, protocolos, *middlewares*, entre outros.

No capítulo 5 é descrito o *Grid Anywhere*. São demonstradas a arquitetura, as propostas, modelos e implementações de cada um de seus módulos.

Por fim, no capítulo 6 são apresentados a conclusão e os trabalhos futuros.

Capítulo 2

Sistemas Distribuídos

2.1 Considerações Iniciais

A computação em grade e a computação em nuvem são paradigmas de computação distribuída que permitem, respectivamente, que recursos possam ser compartilhados e ofertados como serviço [15]. No entanto, a implementação desses tipos de paradigmas utiliza diversas técnicas de construção de sistemas distribuídos como base.

A possibilidade de realizar a distribuição das aplicações entre computadores remotos vem sendo alvo de estudos há muitos anos. Muitos projetos como, por exemplo, RPC, RMI e CORBA, focaram a possibilidade de invocar procedimentos e métodos remotos [22].

Com o passar dos anos, novos requisitos foram especificados (entre eles destaca-se fortemente a interoperabilidade) e novas técnicas foram desenvolvidas. Entre estas técnicas destaca-se a arquitetura orientada a serviço (SOA), bastante utilizada em computação em grade e em nuvem. Os serviços *Web* formam uma maneira bastante comum de implementar SOA.

Juntamente com SOA, os agentes móveis são encontrados na literatura referente aos *middlewares* para grades computacionais, se tornando também importantes para a área [23].

2.2 Serviços Web WS-*

Um serviço *Web* [7] [17] é um sistema de *software* projetado para fornecer formas de interação entre aplicações através de uma rede de computadores. Assim, uma

determinada aplicação pode ser implementada na forma de um serviço e disponibilizada para ser invocada remotamente.

Analisando o contexto no qual os serviços *Web* são projetados, podem-se encontrar alguns requisitos importantes:

- Diversas empresas e profissionais podem desenvolver serviços *Web* que implementam recursos e soluções para determinados assuntos. Sendo assim, é necessário que esses sejam publicados para, da forma mais simples possível, serem localizados e consequentemente utilizados.
- Os serviços *Web* apresentam uma interface pela qual as aplicações clientes podem interagir com esses serviços. Sendo assim, é necessário que a interface de serviços fornecidos esteja disponível para aplicações consumidoras.
- A interoperabilidade entre sistemas diferentes é uma questão bastante importante no âmbito de um sistema distribuído. Por isso, a troca de mensagens entre dois serviços deve ser estabelecida independentemente da plataforma ou linguagem de programação utilizada na construção do serviço.
- As mensagens trocadas entre as aplicações precisam, de alguma forma, ser transportadas entre a máquina origem e destino.

Para que a interoperabilidade seja mantida, diversas especificações e protocolos são utilizados. De uma maneira abstrata, é possível analisar o modelo empregado pelos serviços *Web* através da estrutura de camadas ilustrada na figura 1 [24].

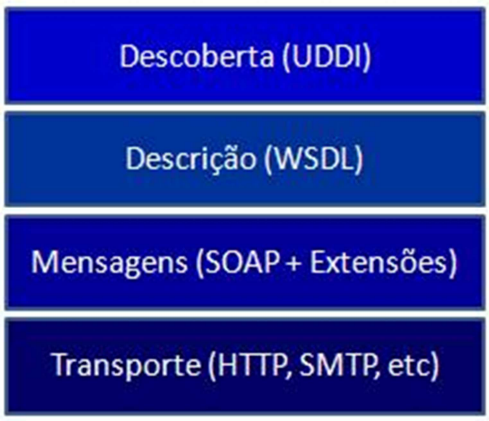

Figura 1: Modelo de camdadas dos serviços *Web*
Baseado em [24]

O UDDI (*Universal Description, Discovery and Integration*) [17] fornece mecanismos para armazenamento de descrições e descoberta de serviços *Web*. Através

de uma consulta a um repositório é possível localizar serviços e obter informações que são necessárias para sua utilização. As informações sobre os serviços podem ser pesquisadas em [25]:

- **Páginas brancas:** As informações são fornecidas através dos nomes dos serviços e dos detalhes para contato.
- **Páginas amarelas:** As informações são fornecidas através dos tipos dos serviços ou das empresas.
- **Páginas verdes**: As informações são fornecidas através de detalhes técnicos como, por exemplo, tipo de transporte a ser utilizado.

Para que uma aplicação qualquer possa interagir com um serviço *Web* é necessário que essa interface seja fornecida de maneira que certas informações sejam conhecidas como, por exemplo, número e tipos dos parâmetros, tipo de retorno, entre outros. O WSDL (*Web Services Description Language*) [25] [17] é um documento em XML [26] (desenvolvido através de um consórcio mantido pela W3C) para descrever um serviço *Web* de maneira que outra aplicação possa trocar informações com esse serviço. É possível efetuar uma analogia simples entre WSDL - serviços *Web* e IDL (*Interface Description Language*) – *Corba* [27].

Compiladores foram desenvolvidos com a finalidade de analisar um documento WSDL e gerar um código que implemente *stubs* para o aplicativo cliente e *skeleton* para o servidor. *Stub*s e *skeletons* possuem a função de abstrair todo o processo de troca de mensagens entre a aplicação cliente e servidora, de forma que para o desenvolvedor seja fornecida uma maneira transparente de interagir com um serviço remoto.

É muito importante que a troca de mensagens entre duas aplicações possa ocorrer mesmo quando existam diferenças entre plataformas e linguagens de programação. Para isso, é feito uso do XML, uma linguagem padrão que vem se destacando muito no que diz respeito à troca de informações e independência de plataforma.

Por isso, o protocolo SOAP (*Simple Object Access Protocol*) [28] [17] para troca de mensagens e invocação de procedimentos remotos é baseado em XML. Esse protocolo define como deve ser o formato de uma mensagem trocada entre duas aplicações e utiliza protocolos já existentes para o envio das mensagens como, por exemplo, HTTP, SMTP, entre outros [25].

Uma mensagem SOAP é organizada em um arquivo XML que representa, através de suas *Tags*, um envelope (elemento raiz) contendo uma área de cabeçalho e uma área para o corpo da mensagem. Essa estrutura pode ser vista na figura 2.

O elemento *Header* é opcional e pode ser utilizado para funções específicas da aplicação no transporte da mensagem. Os intermediários, através dos quais a mensagem passa entre a origem e o destino final, podem processar esses cabeçalhos de maneira a implementar diversas funcionalidades como, por exemplo, autenticação, controle de transação, entre outros [28].

O elemento *Body* contém a mensagem propriamente dita que está sendo trocada entre as aplicações [25]. Esse corpo da mensagem pode implementar dois tipos de interação entre as aplicações:

- ***Document-Style*:** Nesse tipo de interação o elemento *Body* transporta uma mensagem XML que diz respeito a um documento que está sendo trocado entre as aplicações como, por exemplo, um pedido de compra contendo todos os itens, quantidade e preço. Dessa forma, o serviço destino recebe essa mensagem e processa esse documento.
- ***RPC-Style***: Nesse tipo de interação o elemento *Body* transporta um documento que carrega informações sobre uma chamada de procedimento remoto, contendo qual procedimento deve ser invocado e o valor de cada um dos seus parâmetros.

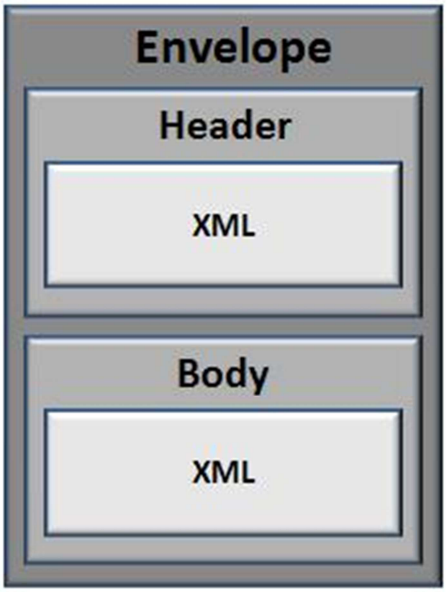

Figura 2: Formato de uma mensagem SOAP

Baseado em [29]

Conforme ilustrado pela figura 3, o processo de utilização de um serviço *Web* inicia quando ele é desenvolvido e publicado em um registro UDDI. O interessado em utilizar tal serviço efetua uma busca no repositório e o encontra. O cliente, então, pode efetuar o mapeamento da interface do serviço através da descrição WSDL e finalmente interagir com o serviço *Web* através do protocolo SOAP.

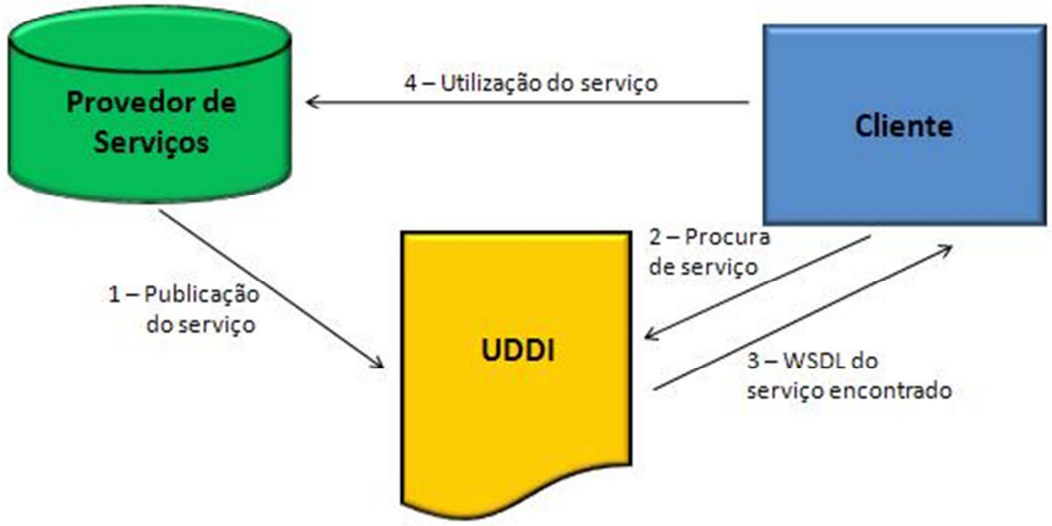

Figura 3: Processo de utilização de um serviço *Web*

A arquitetura básica de um ambiente de execução utilizando serviços *Web* e o protocolo HTTP no transporte das mensagens é mostrada na figura 4 [17]. No cliente são encontrados os seguintes componentes:

- *Implementação do cliente:* Aplicação cliente que irá invocar um serviço remoto.
- *Stub:* Módulo que é gerado pelo compilador através do processamento da interface do serviço *Web* descrito em WSDL. Tem o papel de abstrair a implementação necessária para a invocação do serviço remoto.
- *SOAP Engine:* Responsável pelo tratamento das mensagens SOAP. Pode estar acoplado ao aplicativo cliente.
- *HTTP Engine:* Responsável pelo transporte das mensagens entre os sistemas.

No servidor (provedor de serviços) são encontrados os seguintes componentes:

- *Implementação do servidor:* Aplicação que será executada em função de uma requisição efetuada pelo cliente.

- *Skeleton:* Possui um papel parecido com o *stub* existente no cliente, pois abstrai a implementação necessária para o tratamento da invocação, recebimento de parâmetros, entre outras funções.

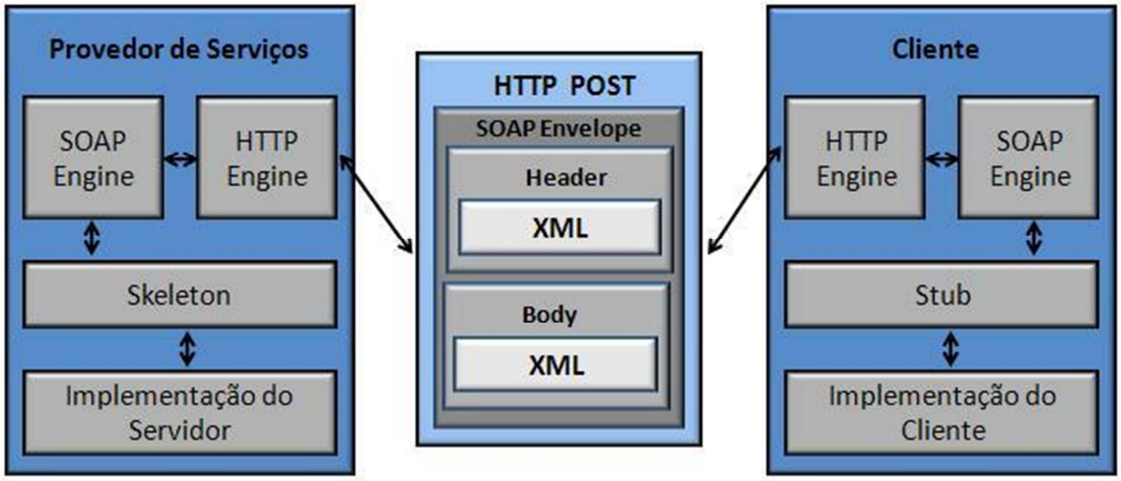

Figura 4: Arquitetura de um ambiente de execução utilizando serviços *Web*

Fonte [17]

2.3 Serviços Web RESTFul

Roy Fielding (um dos contribuidores do HTTP),em sua tese de doutorado, apresentou o conceito denominado *Representational State Transfer* (REST) [30]. Esse conceito é aplicado à *Web* e consiste em clientes enviando requisições a um servidor que retorna uma representação do estado do recurso requisitado. No caso de uma aplicação *Web*, o HTML é utilizado para realizar esta representação. Mas, outros formatos podem ser empregados como, por exemplo, XML e JSON [30].

Uma vez que REST foi inicialmente definido para representar sistemas hipermídia distribuídos, o HTTP é o protocolo base de transporte em arquiteturas dessa natureza [31]. O HTTP apresenta uma interface uniforme composta basicamente de quatro métodos que podem ser utilizados para realizar a interação entre cliente e servidor. São eles: GET, PUT, POST e DELETE que, de forma natural, definem o que deve ser feito pela requisição, conforme tabela 1.

Em uma arquitetura REST o estado da aplicação não é mantido entre as interações cliente e servidor. Assim, quando o estado é um requisito faz-se necessário que sua representação seja transferida a cada nova requisição.

Tabela 1: Mapeamento dos métodos HTTP para ações do serviço

Método	Ação
PUT	Inserir
GET	Recuperar
POST	Atualizar
DELETE	Remover

Fonte [32]

Para que o recurso requisitado possa ser identificado é feito uso extensivo de URIs (*Uniform Resource Identifier*). Uma vez que o formato de um URI permite o estabelecimento de hierarquias, a organização e, consequentemente, identificação dos recursos torna-se bastante flexíveis.

A grande popularidade da *Web* tornou esses protocolos, formatos e padrões amplamente difundidos e utilizados. Por isso, a criação de sistemas que fazem uso deles é relativamente simples, pois ferramentas para manipulação de mensagens HTTP, servidores, permissões de acesso (*firewalls*, por exemplo) são questões rotineiras e bem resolvidas. Sendo assim, a possibilidade de desenvolver serviços *Web* utilizando esses elementos pode tornar a curva de aprendizado menor e minimizar a necessidade por novas ferramentas.

Serviços *Web* RESTFul são implementados utilizando o conceito arquitetural básico REST e fazendo uso dos recursos já existentes no HTTP, sem a necessidade de encapsulamento de outros protocolos como o SOAP.

Devido à ausência de uma descrição formal de interfaces de serviços, como a WSDL, e de um protocolo específico para descrever uma operação de chamada de método, os elementos do próprio protocolo HTTP e o conteúdo da requisição devem ser suficientes para determinar o serviço a ser invocado e os parâmetros que devem ser recebidos.

2.3.1 Associações

A especificação da Oracle, JAX-RS [33], para serviços *Web* RESTFul apresenta a forma de mapeamento de requisições HTTP para uma determinada classe (*Resource Class*) e seus respectivos métodos (*Sub Resources*).

Cada classe é associada a um URI. Sendo assim, quando uma requisição HTTP é recebida pelo servidor de aplicações é realizada uma comparação dessa requisição com as classes mapeadas, aquela que for encontrada tem sua instância criada para realizar a

execução. Se uma classe *C1* é associada ao URI "/clientes", as requisições que possuem o URI começando com "/clientes" serão atendidas por uma instância da classe *C1*.

Depois de instanciado o objeto é preciso decidir qual método deve ser invocado. Para isso, cada método da classe também é associado a um URI. Como exemplo, o método *M1* da classe *C1* pode ser associado ao URI "/clientes/consulta". Dessa forma, após instanciar a classe o servidor busca entre os métodos qual é aquele que está mapeado e o executa.

A associação pode considerar partes estáticas e partes dinâmicas do URI. Se um método *M2* é associado ao URI "/clientes/{id}/pedido", é considerado que "/clientes/" e "/pedido" são partes estáticas e "{id}" é variável. Sendo assim, as requisições que possuem o URI com "/cliente/xxx/pedido" (onde "xxx" pode ser qualquer valor) serão atendidas pelo método *M2*.

2.3.2 Parâmetros

Uma vez instanciado o objeto para atender a classe e determinado o método a ser executado, é preciso determinar os parâmetros a serem enviados, caso existam. Isso também é feito por meio de mapeamento dos parâmetros do método com a forma de recuperação do valor. A tabela 2 apresenta as possíveis formas.

Tabela 2: Tipos de parâmetros

Forma	Extrai o valor de
@MatrixParam	Matriz de parâmetros da URI
@QueryParam	Query de parâmetros da URI
@PathParam	*Template* da URI
@Cookie	*Cookie*
@HeaderParam	Cabeçalho HTTP

Fonte [33]

2.3.3 Clientes e Servidores

Serviços RESTFul podem retornar ao cliente um documento descrevendo o resultado da requisição. Entre outros, esse documento pode ser formatado como XML ou JSON. Uma vez que não existe nenhuma forma padronizada de especificar o retorno de um serviço, é preciso que esse serviço disponibilize documentação para que o cliente

implemente o mecanismo de processamento das mensagens resultantes da interação (*do it yourself* [31]).

Essa característica pode gerar um maior trabalho por parte do programador quando comparado aos serviços WS-*, pois a utilização de padrões como SOAP e WSDL viabiliza de forma simples a troca de mensagens entre cliente e serviço.

2.3.4 Protocolo de Transporte

O estudo realizado por *Pautasso et Al* [31] apresenta uma comparação importante referente aos protocolos de transporte que podem ser utilizados pelos serviços RESTful quando comparados ao WS-*.

É possível notar analisando a tabela 3 que os serviços *Web* RESTFul são fortemente atrelados ao protocolo HTTP, o que em determinadas situações pode ser um fator importante na escolha de qual optar.

Tabela 3: Protocolos suportados pelos serviços *Web*

Protocolo	**RESTFul**	**WS-***
HTTP	Sim	Sim
Waka	Sim	Não
SMTP	Não	Sim
JMS	Não	Sim
MQ	Não	Sim
BEEP	Não	Sim
IIOP	Não	Sim

Fonte [31]

2.4 Agentes Móveis

A construção de softwares utilizando o paradigma de sistemas distribuídos pode ser realizada de diversas formas. Entre elas é possível destacar o uso de RPC, RMI [22] e Serviços *Web* [17]. Sistemas construídos utilizando essas ferramentas normalmente apresentam uma arquitetura cliente/servidor onde o código da aplicação é armazenado no servidor e o cliente ao invocar um serviço informa os dados do contexto por meio de parâmetros que são enviados através da rede de computadores.

Os agentes móveis formam uma categoria de sistemas distribuídos onde tanto o código da aplicação quanto seu estado são migrados entre dois sistemas remotos [34]. Nesse tipo de aplicação um computador pode receber um agente, permitir seu processamento e depois enviá-lo para outro destino. Sendo assim, a arquitetura cliente/servidor deixa de ser adequada, visto que um mesmo computador pode operar tanto como servidor (quando recebe um agente) quanto como cliente (quando envia um agente).

A utilização de agentes móveis, entre outras vantagens, pode diminuir o tráfego de dados na rede, pois é possível que o código do programa seja movido para o mais próximo possível da fonte de dados a ser manipulada por ele. Se for imaginado o processamento de dados de uma tabela de um SGBD (sistema de gerenciamento de banco de dados), um aplicativo cliente/servidor convencional faz a busca dos dados no servidor, realiza seu processamento no cliente e devolve os resultados novamente ao servidor. Considerando uma tabela com um grande volume de dados, o tráfego na rede poderá ser alto.

Com a utilização de agentes móveis para este tipo de aplicação, uma vez que o agente pode ser migrado para a mesma máquina onde está o SGBD, o único tráfego de dados será oriundo do transporte do código executável da aplicação, que em comparação com o volume de dados poderia ser considerado muito pequeno.

Sistemas dessa categoria podem ser construídos utilizando-se ferramentas de inteligência artificial, o que determina os agentes móveis inteligentes. Esse tipo de agentes tem uma grande aplicação em sistemas de diversas naturezas, como, por exemplo, mineração de dados.

A dotação de um componente de *software* com inteligência e a capacidade de se movimentar que ele apresenta justifica a utilização do nome “agente”. Quando uma tarefa precisa ser realizada em diversos computadores (ou equipamentos) diferentes, o usuário pode delegar esta função à aplicação que o representa como um agente.

2.4.1 Mobilidade de Agentes

A capacidade de movimentação dos agentes entre dois sistemas remotos é o foco desta seção da tese. Para que tal movimentação aconteça é preciso que tanto o estado do agente quanto seu código sejam migrados para o computador de destino.

O estado de uma aplicação que deve ser migrada é dividido em duas partes, conforme ilustrado pela figura 5.

Figura 5: Estado de um Agente Móvel

Baseado em [34]

O estado de dados armazena as variáveis locais e os atributos do objeto que representa o agente móvel. Esse estado é utilizado para reconstruir o objeto no destino mantendo os dados gerados e manuseados pela aplicação da mesma forma que estava na origem.

De maneira a ser capaz de reiniciar o agente na máquina de destino do mesmo ponto onde ele parou na origem é utilizado o estado de tempo de execução, que é responsável por indicar como estava a pilha de chamada de métodos e o contador de programas.

Há duas formas diferentes de realizar a migração de um agente:

- *Strong Migration*: Também conhecida como migração transparente, ela permite que um agente em execução possa ser migrado a qualquer momento e quando chegar ao destino sua execução continua do exato ponto onde parou na origem. Para isso, é preciso fazer o transporte dos estados de dados e de tempo de execução.

- *Weak Migration*: É a migração não transparente. Nesse tipo de migração de agente não há a continuidade de execução do agente do ponto onde ele estava sendo executado na origem. Na chegada ao destino, um novo ciclo de chamadas é iniciado, o que dispensa a migração do estado de tempo de execução.

Além da transferência dos estados citados é preciso realizar a migração do código que implementa o agente. No entanto, é possível que o agente solicite instâncias de novas classes no seu destino, sendo preciso carregar os códigos que as implementam. Sendo assim, novas transferências se tornam necessárias [35].

2.4.2 Comunicação entre Agentes Móveis

Agentes móveis podem se comunicar, originalmente, por meio de um protocolo chamado *Agent Communication Language* (ACL) [36] que é especificado pela FIPA (*Fundation for Intelligent Physical Agents*) [37].

Para construir sistemas mais complexos, muitas vezes é preciso realizar a comunicação entre aplicações distintas. Essa integração é atualmente suportada de maneira padronizada por meio do uso de serviços *Web* [17]. Sendo assim, torna-se bastante interessante a possibilidade de realizar a comunicação com agentes móveis por meio da chamada a serviços *Web*.

Por isso, algumas soluções foram desenvolvidas com o objetivo de permitir que a comunicação com agentes móveis fossem realizadas por meio de serviços *Web* [36]:

- *WSDL2JADE*: Responsável por traduzir chamadas em ACL para requisições a um serviço *Web*. Por meio do processamento de um documento WSDL, o WSDL2JADE cria um *proxy* para realizar esta integração.
- *WSAI*: Permite que agentes móveis sejam publicados como serviços *Web*. Para isso um componente chamado WSAG (*Web Services Agent Gateway*) recebe mensagens formatadas utilizando o protocolo SOAP e as converte para ACL que são encaminhadas ao agente móvel.
- *WSIG*: É uma evolução do WSAI que permite que o processo de integração entre agentes móveis e serviços *Web* seja feito de forma transparente. Para isso, existe um elemento que permite que tanto

serviços *Web* quanto agentes se registrem para que suas mensagens sejam interceptadas e convertidas.

2.4.3 Ferramentas para Construção de Agentes Móveis

A construção de um sistema distribuído envolve diversos requisitos que precisam ser implementados como, por exemplo, migração de código, migração de objetos, segurança, ambiente de execução, comunicação, entre outros.

Algumas ferramentas oferecem o ambiente de execução de agentes móveis que provêm abstração dos requisitos básicos, permitindo que o programador trabalhe em um nível mais alto de desenvolvimento. Na literatura é comum encontrar referências para as seguintes ferramentas:

- *Aglets*: Totalmente desenvolvida em Java pelo laboratório de pesquisa da IBM no Japão, a última versão é do ano de 2004 [38].
- *Voyager*: Plataforma comercial mantida pela empresa *Recursion Software* [39]. Permite a construção de aplicações utilizando agentes móveis para diversas plataformas, estando entre elas: Java, Dot Net e Android.
- *Concordia*: Também implementado em Java o Concordia tem como foco a mobilidade flexível, colaboração, transmissão confiável de agentes e segurança [40].
- *JADE*: Implementação de uma plataforma para agentes móveis totalmente baseado na especificação FIPA [41]. Mantido pela Telecom Itália, o JADE é totalmente implementado em Java e sua última versão é de novembro de 2011.

2.5 Grades Computacionais

A maior complexidade dos problemas a serem resolvidos por computadores exige uma grande potência computacional [42]. Um computador pessoal no ano 2001 era tão rápido quanto um supercomputador de 1990, mas a complexidade dos trabalhos realizados também se tornou efetivamente maior. Por exemplo, essa complexidade

evoluiu da análise de uma estrutura molecular para a análise de uma construção complexa de macromoléculas.

A popularidade da Internet, o avanço e a proliferação das redes de alta velocidade tornaram possível a interligação de diversos recursos computacionais de maneira a compor um único sistema [4]. Na década de 90 foi criado o termo “Grade Computacional” para denotar essa infraestrutura de sistema distribuído em que recursos (podendo variar desde processadores e discos até licenças de *software*) geograficamente distribuídos e sob administrações independentes possam ser compartilhados entre os participantes da Grade Computacional.

O agrupamento de recursos distribuídos em uma Grade Computacional permite o compartilhamento desses recursos entre os participantes e torna o sistema suficientemente potente para contribuir para a solução, de forma colaborativa, de diversos problemas sem a necessidade de os participantes adquirirem novos equipamentos. Ao redor de todo o mundo é possível encontrar um grande número de computadores cuja capacidade não é totalmente explorada. Dessa forma, o agrupamento dos percentuais de ociosidade desses equipamentos pode oferecer uma potência computacional considerável para a solução de muitos problemas.

Grades computacionais podem se apresentar de maneiras diferentes no que diz respeito à forma de compartilhamento e utilização de recursos. Em uma abordagem possível e bastante comum, instituições de pesquisa, universidades e empresas podem gerar uma “Organização Virtual” para compartilhar recursos como supercomputadores, *clusters*, entre outros, que podem auxiliar no processamento de alto desempenho. *Globus Toolkit* [6] [43] e *OurGrid* [10] [44] [9] são dois exemplos de *middlewares* que implementam grades computacionais dessa natureza.

As Organizações Virtuais podem ser compostas para atender a diversos tipos de trabalhos e podem variar em muitos aspectos, tais como: tamanho, finalidade à qual se destinam, escopo, duração, entre outros [2]. Os participantes de uma Organização Virtual definem regras como: quem pode usar recursos, quais recursos podem usar, quando e como.

No entanto, além de equipamentos voltados para o processamento de alto desempenho (como *clusters* e supercomputadores) muitos computadores pessoais possuem uma grande faixa de processamento ociosa. Esses computadores normalmente

são de uso doméstico ou corporativo e, juntos, podem apresentar um ambiente de grande potência computacional.

Esses computadores podem ser utilizados para realizar o processamento de tarefas que exigem um grande esforço computacional de forma paralela. Para isso, uma arquitetura mestre-trabalhador é adotada, onde os usuários se prontificam a disponibilizar seus computadores para realizar o processamento, de forma voluntária, de um ou mais projetos que o usuário esteja disposto a colaborar [12] [45].

É importante observar que em uma grade computacional *desktop* os participantes "trabalhadores" atuam sempre no papel de provedores de recursos. Sendo assim, o consumo de recursos é realizado de forma unidirecional, sendo sempre a instituição que atua como "mestre" a beneficiada pela grade computacional. QADPZ [45], Boinc [11] e SKTAKI [46] são exemplos de *middlewares* para grades computacionais *desktop*.

O projeto *SETI@home* [12] é um dos exemplos mais clássicos desse modelo de grade computacional. Ele utiliza os ciclos ociosos de computadores espalhados por todo o mundo para analisar dados referentes a sinais de rádio recebidos por um telescópio localizado em Porto Rico, em busca de vida inteligente fora do planeta Terra. Os recursos dos participantes dessa grade computacional formam uma estrutura computacional com uma capacidade coletiva de processamento de dezenas de *Teraflops* [47].

2.5.1 Arquitetura de Grades Computacionais

A arquitetura de uma Grade Computacional fornece mecanismos para o compartilhamento de recursos [2], atendendo aspectos como interoperabilidade e segurança.

Durante uma década de pesquisa nessa área, foi produzido um consenso considerável quanto aos requisitos e à arquitetura de uma Grade Computacional [1]. Muitos protocolos para troca de mensagens entre aplicações foram propostos, assim como diversas APIs com o intuito de oferecer meios mais simples e produtivos para a construção de uma Grade Computacional.

Esses protocolos e APIs podem ser agrupados em uma estrutura de camadas de acordo com o papel que exercem [1]. As camadas internas possuem um número menor

de elementos e, por consequência, a representação da arquitetura através do modelo de camadas possui uma forma parecida com uma ampulheta, conforme pode ser observado na figura 6.

A camada Ambiente acomoda os diversos recursos da grade, que são compartilhados através da utilização de protocolos. Tais recursos podem ser tanto recursos físicos (unidades de disco, processadores, entre outros) como também recursos lógicos (como sistemas de arquivos e licenças de *software*).

Os componentes da camada Ambiente oferecem protocolos e mecanismos que implementam as operações fornecidas pelos recursos. Esses protocolos e mecanismos podem ser fornecidos pelo fabricante do recurso ou até mesmo disponibilizados pelo *Middleware* da grade computacional.

A camada Conectividade e Segurança define um conjunto básico de protocolos para atender aos requisitos de comunicação e autenticação de uma operação da grade computacional. Essa camada é composta por protocolos como, por exemplo, IP, ICMP, TCP, UDP, entre outros [1].

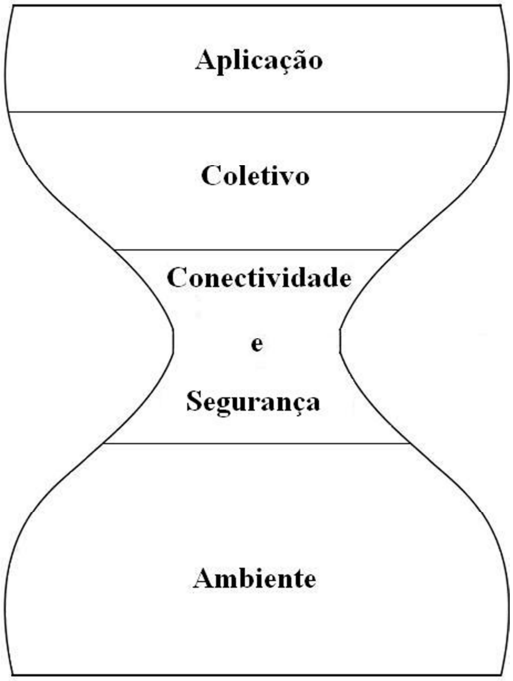

Figura 6: Modelo de Camadas de uma Grade Computacional

Baseado em [1]

Os aspectos de segurança são implementados por mecanismos de autenticação e criptografia. Em uma organização virtual os mecanismos de autenticação devem oferecer, entre outras, as seguintes características:

- *Único Log-on:* Após o usuário efetuar um processo de autenticação para utilizar um determinado recurso da grade computacional, deve ser possível que ele utilize diversos outros recursos sem a necessidade de nova autenticação.
- *Delegação:* É necessário que um usuário possa atribuir a um determinado programa os seus direitos, de maneira que esse atue no sistema em seu nome. Esse programa também deve ser capaz de transferir esses direitos.
- *Integração com várias soluções locais de segurança*: Vários sistemas podem utilizar sistemas locais de segurança e é necessário que os mecanismos da grade sejam capazes de interagir com eles.

A camada Conectividade e Segurança utiliza os mecanismos da camada Ambiente para realizar de forma segura as operações sobre um recurso compartilhado (como, por exemplo, negociação, monitoração, contabilização, entre outros). Os protocolos existentes na camada Conectividade e Segurança incluem:

- *Informação:* Os protocolos encontrados nessa classe são utilizados para obter informações sobre um determinado recurso.
- *Gerenciamento:* Responsáveis pela negociação de utilização de um determinado recurso incluindo aspectos como qualidade de serviço, operação a ser executada, entre outros.

A camada Coletivo possui protocolos e APIs que implementam um conjunto de operações sobre uma coleção de recursos. É possível destacar alguns exemplos:

- *Serviços de diretório:* Armazenam e disponibilizam informações sobre os recursos existentes de forma que os participantes possam consultá-los com a finalidade de localizar e obter dados sobre esses recursos.
- *Serviços de alocação e escalonamento:* Alocam um recurso apropriado para atender uma determinada requisição em função de certos parâmetros.

Na camada Aplicação são encontradas as aplicações propriamente ditas, que operam utilizando os recursos existentes em uma grade computacional.

2.5.2 OGSA, OGSI e WSRF

Em um sistema distribuído a interoperabilidade é uma questão muito importante [3] [5]. Dessa forma, estabelecer um padrão a ser seguido pelas diversas instituições e desenvolvedores que produzem produtos para uma grade computacional contribui para que a interoperabilidade possa ser alcançada entre produtos distintos adotados em uma organização virtual.

A OGSA (*Open Grid Services Architecture*) define que todos os recursos compartilhados devem ser representados por serviços denominados serviços de grade, que são serviços *Web* com interfaces bem definidas para atender a certos requisitos, tais como: a descoberta e a criação dinâmica de serviços, o gerenciamento de tempo de vida, a notificação, entre outros .

A OGSA define ainda requisitos referentes à identificação de instâncias de serviços, pois um mesmo serviço pode possuir diversas instâncias identificadas pelo *Grid Service Handle* (GSH).

O GSH é um nome globalmente único para uma determinada instância e não varia durante o tempo de vida do serviço [5]. No entanto, um GSH não possui as informações necessárias para interagir com uma instância de um serviço de grade (como, por exemplo, protocolo de transporte utilizado e endereço de rede). Sendo assim, tais informações são armazenadas em um *Grid Service Reference* (GSR), que possui um tempo de expiração pré-determinado. Dessa forma, a OGSA define mecanismos para obter novos GSRs partindo de um determinado GSH.

A OGSA não descreve detalhadamente como um serviço de grade deve ser, pois, ela descreve a arquitetura de uma Grade Computacional orientada a serviços de uma forma global, com uma macro visão. A OGSI (*Open Grid Service Infrastructure*) é uma especificação que descreve detalhadamente a arquitetura esboçada pela OGSA, definindo em detalhes como deve ser e se comportar um serviço de grade. A figura 7 ilustra essa relação entre a OGSA, OGSI e serviços de grade, bem como é implementado pelo *Globus Toolkit*.

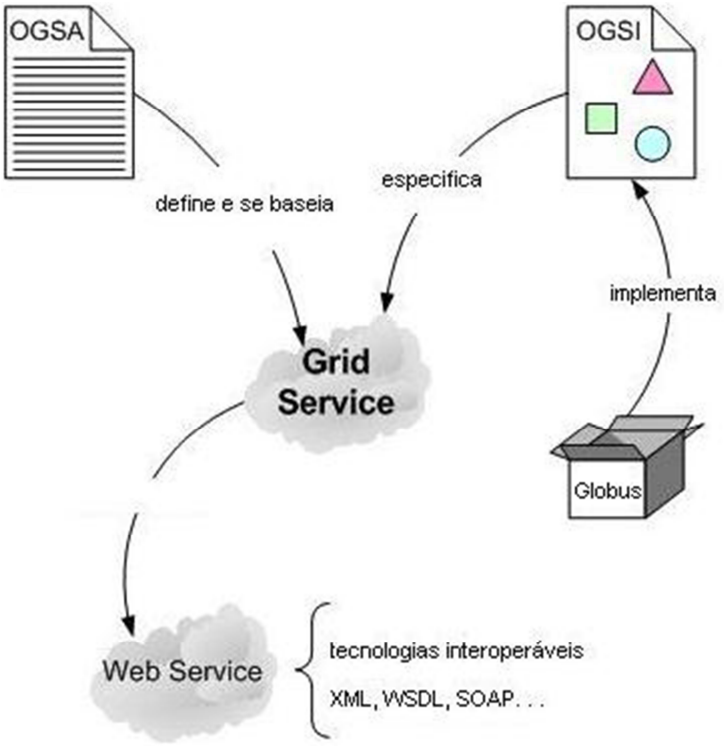

Figura 7: Relação entre OGSA e OGSI

Fonte [48]

De forma a efetuar uma simples analogia, é possível comparar a construção de um serviço de grade com a construção de uma casa. Pois o primeiro passo quando se deseja construir uma casa é procurar um arquiteto de maneira que esse efetue um projeto, dando uma visão ampla de como a casa será depois de pronta. No entanto, esse projeto não traz consigo detalhes de como essa casa deve ser construída. Isso exige que um engenheiro seja consultado, de maneira a criar um projeto mais detalhado definindo como deve ser a estrutura dessa casa, a instalação elétrica e hidráulica, entre outros. Assim, na analogia apresentada a OGSA faz um papel semelhante ao do arquiteto e a OGSI ao do engenheiro [47] [48].

Com a evolução dos serviços *Web*, alguns aspectos da OGSI precisaram ser reanalisados [8]. Assim, surgiu a especificação WSRF (*Web Services Resource Framework)* [49] [50] que refina a OGSI e divide a especificação em outras menores, agrupadas em uma mesma família.

A necessidade de que seja possível que o estado entre diversas invocações de um serviço *Web* seja mantido é um ponto bastante importante e discutido. De maneira a oferecer essa funcionalidade, a WSRF disponibiliza o *WS-Resource,* que é definido como a composição de um serviço *Web* e um recurso capaz de manter seu estado [49]. Tal recurso é expresso como um documento XML, que pode ser criado, endereçado,

acessado, monitorado e destruído, através de mecanismos convencionais existentes nos serviços *Web*.

2.6 Computação em Nuvem

Partindo dos computadores de grande porte até os *tablets* e *smartphones*, a computação é uma ferramenta presente em, praticamente, todas as áreas e segmentos da sociedade moderna. Sua aplicação varia desde um simples usuário doméstico utilizando uma aplicação para ler notícias disponíveis na Internet até sistemas complexos para diagnóstico de doenças.

Com o objetivo de que um usuário, seja ele doméstico ou corporativo, possa fazer uso dos benefícios que a computação e a Internet oferecem, é preciso que ele disponha de recursos que, neste caso, são equipamentos (*hardware*) e aplicativos (*software*).

Com o passar do tempo é normal que novas necessidades requisitem um ambiente computacional mais potente e moderno. Quando isso ocorre é preciso que os recursos sejam ampliados, o que pode envolver um grande custo, pois é necessário investimento para aquisição de equipamentos, adequação de espaço físico, especialistas capacitados e tempo para execução do projeto.

Outras atividades do cotidiano também requerem seus recursos como, por exemplo, água, energia elétrica, gás, entre outros [16]. No entanto, o usuário não deseja ser o responsável pela geração e manutenção desses recursos, pois seria inviável, por exemplo, que um cidadão tivesse que possuir um gerador de energia para poder fazer uso da eletricidade. Ao contrário disso, ele prefere contratar um serviço que é responsável pelo fornecimento desse recurso e pagar por aquilo que for consumido.

A computação em nuvem é um sistema distribuído especializado que tem como objetivo permitir que recursos computacionais também sejam ofertados como serviço, por meio da Internet (o que é uma característica da *Web* 2.0), onde o usuário contrata um provedor e paga por aquilo que utilizar [16] [15].

Nesse tipo de arquitetura, o serviço consumido pode ser de extrema importância para o negócio do contratante, o que determina que problemas em sua oferta podem

causar prejuízos de diferentes formas. Sendo assim, SLAs (*Service-Level Agreements*) são fatores primordiais que estão presentes entre o consumidor e o provedor de recursos, determinando, entre outras, questões de QoS (*Quality of Service*) e penalidades que serão aplicadas em casos de não cumprimento de contratos.

2.6.1 Principais Categorias

Na computação em nuvem diversos são os recursos que podem ser ofertados como serviços a serem utilizados por clientes que pagam pelo volume consumido. As principais formas de oferta de recursos como serviço são [16] [15]:

- *Infrastructure as a Service* (IaaS): Provedores de computação em nuvem ofertam *hardwares* virtualizados (máquinas virtuais, discos, entre outros) que são utilizados pelos consumidores na composição de suas infraestruturas de informática. Com isso, servidores de banco de dados, aplicações, arquivos, entre outros, podem deixar de ser acomodados fisicamente e movidos para a nuvem e consumidos como serviço.
- *Software as a Service* (SaaS): Atualmente é comum que as aplicações utilizadas por um usuário sejam instaladas localmente em seu computador. Esse tipo de utilização do software gera algumas questões como, por exemplo, necessidade de atualização, reinstalação quando o usuário tem seu sistema alterado, cópias de segurança, entre outros. De maneira a solucionar essas questões a oferta de *software* como serviço permite que as aplicações sejam totalmente disponibilizadas e utilizadas por meio da *Web*.
- *Platform as a Service (*PaaS*):* Disponibiliza ferramentas e plataformas para que o usuário possa desenvolver e distribuir aplicações que ficam hospedadas em servidores localizados na nuvem, de onde são executados e utilizados pelos clientes.

2.6.2 Arquitetura em Camadas

Um sistema baseado em computação em nuvem é complexo e exige um grande número de requisitos. Esses requisitos variam desde o gerenciamento de recursos até as aplicações. Na figura 8 é apresentada uma organização em camadas desses requisitos.

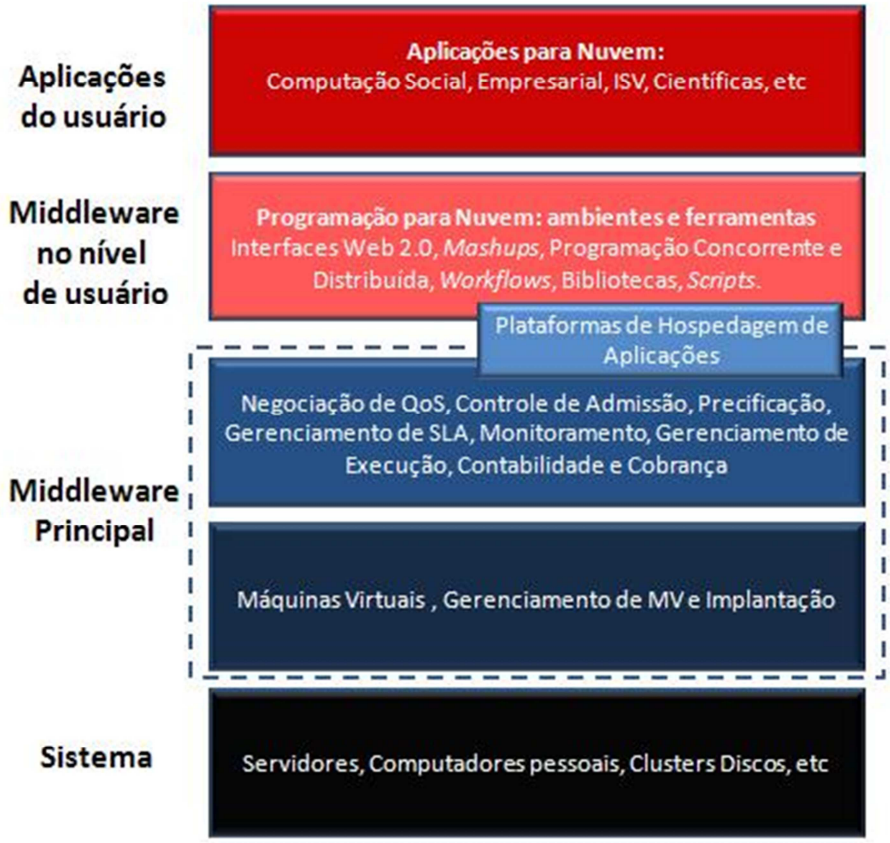

Figura 8: Organização em camadas de um ambiente de computação em nuvem

Fonte [16]

A camada "Sistema" é responsável por acomodar os recursos que são disponibilizados por meio da nuvem como, por exemplo, dispositivos de *hardware*. No entanto, para que esses recursos possam ser utilizados por usuários consumidores é preciso que eles sejam ofertados na forma de serviços e que sejam gerenciados, sendo estas funções atribuídas ao "*middleware* principal".

Para isso, o "*middleware* principal" deve disponibilizar mecanismos que permitam que clientes submetam seus trabalhos para o servidor (*deployment*) e hospedem esses trabalhos, o que comumente é feito por meio de máquinas virtuais.

Além do gerenciamento técnico dos recursos o "*middleware* principal" também se encarrega de realizar os procedimentos que viabilizam o modelo de negócio da computação em nuvem. Sendo assim, entre outros, são necessários:

- *Precificação*: Definição do preço a ser cobrado do cliente pelo uso dos recursos.

- *Negociação de QoS*: Determinar o nível de qualidade de serviço a ser ofertado de maneira a atender aos requisitos do usuário.
- *Gerenciamento de SLA*: Consiste em ferramentas para realizar o monitoramento da oferta de serviços com o objetivo de verificar se o contrato estabelecido com o consumidor está sendo atendido
- *Monitoramento e Gerenciamento de Execução*: Uma vez que aplicações de clientes são executadas no ambiente da nuvem, é preciso monitorar esta execução de maneira a garantir que operações não autorizadas sejam realizadas e que volume de recursos consumidos sejam aqueles definidos anteriormente, entre outros.
- *Contabilização e Cobrança*: Como a utilização dos serviços é paga pelo cliente é preciso determinar o valor a ser cobrado e permitir que o cliente realize o pagamento. Sendo assim, o mecanismo de contabilização realiza a contagem do volume dos recursos gastos e o de cobrança viabiliza seu pagamento pelo cliente.

Os dois níveis apresentados até agora focam nos requisitos existentes para o provedor de serviços. No entanto, muitos são os requisitos que também devem ser atendidos para que usuários possam fazer uso dos serviços existentes na nuvem.

O "*middleware* de nível de usuário" tem a responsabilidade de abstrair os requisitos necessários para permitir o acesso aos serviços ofertados no provedor. Isso inclui ambientes para desenvolvimento de aplicações distribuídas e paralelas, bibliotecas, interfaces para *Web* 2.0, entre outros.

Por fim, na camada "aplicações do usuário" são acomodadas as aplicações que fazem uso de recursos da nuvem para atender às necessidades do usuário.

2.6.3 Provedores de Serviços

Mesmo sendo um paradigma relativamente recente na área de sistemas distribuídos, a computação em nuvem já conta com diversos provedores que ofertam recursos como serviços.

2.6.3.1 ***Amazon EC2***

O Amazon EC2 (*Elastic Cloud Computing*) é um serviço que oferta infraestrutura como serviço (*IaaS*) [16] [51] e cobra pelo volume utilizado. É disponibilizado um conjunto de serviços *Web* que permite que o cliente realize o gerenciamento de suas máquinas virtuais.

Entre as funções de gerenciamento há a característica de poder alterar a capacidade computacional das máquinas virtuais, de acordo com a necessidade do cliente, que caracteriza o serviço como elástico.

2.6.3.2 ***Google AppEngine***

Com foco no fornecimento de plataforma como serviço (PaaS), o Google *AppEngine* oferta um ambiente para desenvolvimento e hospedagem (nos servidores da Google) de aplicações desenvolvidas em Java (JSP, Servlets, etc) e Python [16] [52].

O modelo de negócio do Google *AppEngine* permite que aplicações que consomem até um limite máximo de 500MB de armazenamento e fazem uso de recursos de execução (CPU, memória e largura de banda) para um montante de aproximadamente de 5 milhões de visualizações do aplicativo por mês sejam hospedadas gratuitamente. Quando o desenvolvedor solicita o faturamento da utilização, o volume que ultrapassar o limite gratuito é cobrado.

Os aplicativos são executados em ambientes seguros (*sandbox*) que não permitem que operações que possam comprometer o sistema operacional subjacente sejam executadas. Entre as restrições, destacam-se:

- Os aplicativos são executados em resposta a uma requisição da *Web*, uma fila de tarefas ou uma tarefa agendada.
- Um código de aplicativo não pode ser executado por mais de 30 segundos.
- Aplicativos só recebem pedidos externos de conexão de rede por meio do protocolo HTTP (ou HTTPS) utilizando as portas padrão
- Não é possível realizar gravações no sistema de arquivos. As leituras podem ser executadas somente nos arquivos distribuídos junto com o aplicativo.

2.6.3.3 ***Sales Force***

O *Sales Force* é uma empresa que tem seus negócios focados em sistemas para gestão de empresas de comércio [53]. Esses sistemas são utilizados totalmente pela *Web*, dispensando o cliente de manter infraestrutura de *hardware*, bancos de dados, entre outros, para viabilizar a utilização do sistema. Isto caracteriza o fornecimento de *software* como serviço (SaaS).

No entanto, é possível utilizar o *SalesForce* para desenvolver sistemas personalizados que atendam às demandas do negócio. Para isso, é ofertada uma plataforma para desenvolvimento e hospedagem de aplicativos, totalmente baseada na *Web* (PaaS).

Além disso, são ofertadas APIs que auxiliam o desenvolvedor. São encontradas bibliotecas que abstraem questões comerciais, integração com redes sociais, entre outros.

2.7 Considerações Finais

Neste capítulo foi apresentado um estudo sobre as formas de implementação de serviços *Web*, agentes móveis, paradigmas de computação em grade e em nuvem e a relação entre eles.

Esses estudos sobre serviços *Web* formam a base conceitual necessária para a proposta de mecanismos de transporte de mensagens SOAP, que viabilizam a construção da grade computacional baseada em migração de objetos, onde usuários em diversas situações de conectividade possam atuar como provedores de recursos.

Foram apresentados os fundamentos sobre a computação em grade. No entanto, um melhor estudo sobre as soluções já existentes é muito importante, sendo apresentado no próximo capítulo.

Capítulo

3

Middlewares para Grades Computacionais

3.1 Considerações Iniciais

Uma vez que nesta tese de doutorado é apresentada uma nova proposta de *middleware* para grades computacionais é importante que as soluções existentes atualmente sejam analisadas.

Neste capítulo são apresentadas diversas opções de *middlewares* para grades computacionais para compartilhamento de recursos entre instituições que formam uma organização virtual, grades computacionais *desktop* e grades compostas por receptores de TV.

3.2 Globus Toolkit

O *Globus Toolkit* [6] [43] é uma ferramenta de código aberto, que teve sua primeira versão (1.0) lançada em 1998. Ela permite que recursos de processamento e armazenamento de dados, entre outros, sejam compartilhados de forma segura entre os participantes de uma organização virtual, rompendo as barreiras de localização geográfica sem sacrificar a autonomia local.

Essa ferramenta fornece um conjunto de programas, serviços e bibliotecas que auxiliam e possibilitam o desenvolvimento de grades computacionais, incluindo

mecanismos de segurança, gerenciamento de recursos e dados, detecção de falhas e portabilidade.

Pelo fato de uma grade computacional ser, basicamente, um conjunto de aplicações distribuídas, a interoperabilidade entre elas é de fundamental importância. Dessa forma, o *Globus Toolkit* faz um uso extensivo de serviços *Web* para definir interfaces e estruturar seus componentes. A melhor forma para se garantir a interoperabilidade entre produtos desenvolvidos por entidades diferentes é a adoção de padrões.

Ao observar a Internet, fica claro que uma das questões que a levaram a tal patamar de adoção e sucesso é a padronização, pois, para acessar uma página *Web* não é necessário que o navegador e o servidor *Web* sejam desenvolvidos por uma mesma empresa ou instituição. Dessa mesma forma, o *Globus Toolkit* adota especificações (OGSA e WSRF) que ditam os padrões que devem ser seguidos de maneira a garantir tal interoperabilidade.

O *Globus Toolkit* é composto por um conjunto de módulos agrupados em quatro famílias. Essas famílias são, conforme mostrado na figura 9: Segurança, Gerenciamento de Dados, Gerenciamento de Execução e Módulo de Execução Comum.

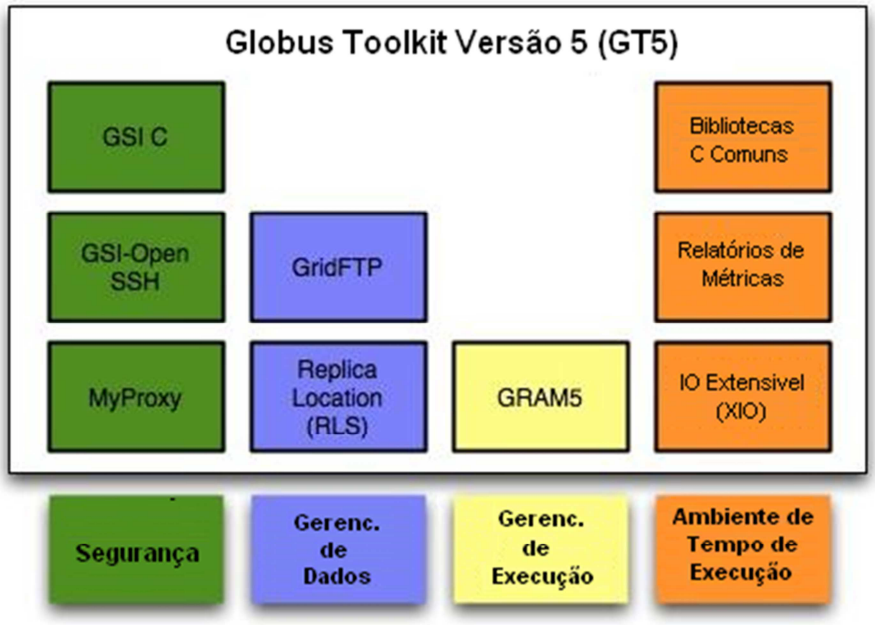

Figura 9: Módulos do *Globus Toolkit* 5

Adaptado de [43]

3.2.1 GRAM

Uma das principais funcionalidades de uma grade computacional é a execução remota de aplicações. Nem sempre a aplicação a ser executada e seus dados se encontram na máquina destino, o que requisita transferência de dados. É necessário também que exista um controle de segurança no local onde a aplicação será executada, pois a aplicação pode ser maliciosa e causar danos ou prejuízos locais.

Quando um cliente do GRAM (*Grid Resource Allocationand Management*) efetua uma submissão de um serviço para execução, esse pode especificar quais os arquivos que devem ser transferidos para o servidor antes do início da execução (podendo ser o próprio executável do serviço e arquivos que serão lidos durante o processamento). O GRAM efetua essa transferência conhecida por *StageIn*. De forma semelhante, o serviço executado pode gerar arquivos contendo os resultados do processamento que são enviados ao cliente (*StageOut)*. O processo de *CleanUp* tem a responsabilidade de excluir os arquivos que foram gerados no servidor durante a execução.

3.2.2 Gerenciamento de Dados

Em uma grade computacional, muitas aplicações envolvem manipulação de dados, transferências de arquivos da aplicação para o servidor (*StageIn*) e transferência de resultados da aplicação (*StageOut)*. Essas funções são realizadas, entre outros, pelo GridFTP e RLS (*Replica Location Service*).

GridFTP é uma extensão do protocolo FTP para transferência de arquivos. Entre os motivos para se utilizar esse protocolo estão: sua grande utilização no mundo para transferência de arquivos, sua especificação aberta e sua arquitetura bem definida [54].

O RLS auxilia na localização de réplicas de arquivos [55]. Esse serviço emprega *Local Replica Catalog* (LRC) que mapeia nomes físicos para nomes lógicos e *Replica Location Index* (RLI). De maneira a fornecer balanceamento de carga e possibilitar que o sistema não tenha um ponto único de falha, o registro no LRC pode ser replicado. As réplicas podem ser localizadas através de consultas ao RLI.

3.2.3 Segurança

As ferramentas *de* segurança preocupam-se em definir quem são os usuários, se realmente eles são quem dizem ser e quais operações eles são autorizados a fazer [7].

A GSI (*Globus Security Infrastructure*) é utilizada no *Globus Toolkit* para implementar os aspectos de segurança. Ela utiliza criptografia por chaves assimétricas como base de sua funcionalidade [24].

O *Globus Toolkit* disponibiliza alguns componentes que possuem papéis bastante importantes em seu modelo de segurança:

- *SimpleCA:* É uma autoridade certificadora capaz de emitir certificados a usuários e serviços de uma Grade Computacional;
- *Delegation*: Disponibiliza uma interface para delegação de credenciais;
- *MyProxy*: É um repositório onde são armazenadas credenciais X.509.

3.2.4 Integração de Agentes Móveis ao Globus

O *Globus Toolkit*, por ser uma implementação da OSGA, abriga as aplicações no formato de *Grid Services*, que são serviços *Web*. O serviço é recebido, escalonado, executado e gerenciado pelo GRAM. Quando concluído seus resultados são enviados ao usuário.

No entanto, pode ser interessante que um serviço em execução em um nó da grade possa ser transferido para outro nó, viabilizando, por exemplo, o balanceamento de carga entre os computadores que compõem o cluster do participante.

O trabalho apresentado por *Aversa, Martino e Venticinque* [23] consiste na integração entre agentes móveis e o *Globus Toolkit*. A solução apresentada faz uso do JADE para prover a infraestrutura necessária para agentes móveis.

De maneira a integrar o ambiente de agentes móveis e o Globus é utilizado um *proxy* instalado no *front-end* do participante da grade. Esse *proxy* é responsável por fazer a ligação entre o *container* WSRF do Globus e o *container* de agentes.

3.3 OurGrid

O *OurGrid* [10] [44] [9], desenvolvido na Universidade Federal de Campina Grande, é uma ferramenta voltada ao compartilhamento de recursos em grades computacionais ponto a ponto, onde cada ponto representa um *site* (como, por exemplo, um conjunto de máquinas em único domínio administrativo). A grade é concebida como uma rede de favores onde um participante, voluntariamente, oferece seus recursos ociosos para que outros possam utilizá-los e também utiliza recursos remotos quando sua capacidade local não é suficiente para uma determinada requisição.

Esse *middeware* possui como uma de suas características principais a execução de BoTs (*Bag of Tasks*), as quais são tarefas que não possuem comunicação entre si e são bastante úteis em mineração de dados, processamento de imagens, quebra de chaves e simulações.

Como a participação na grade é voluntária, para motivar a oferta de recursos nessa rede de favores o *OurGrid* procura:

- Obter uma equidade, de maneira que o volume de recursos que um participante obteve da grade seja proporcional àquele compartilhado por ele.
- Dar uma prioridade maior àqueles participantes que ofereceram mais recursos à rede.

Conforme ilustrado nafigura 10, os principais componentes do *OurGrid* são:

- *MyGrid*: O qual será melhor detalhado a seguir, faz o papel de intermediário utilizado pelo usuário no momento de interagir com a grade computacional.
- *OurGrid Community*: Responsável pela montagem da grade a ser utilizada pelo componente *MyGrid*.
- *SWAN*: Responsável pelo acesso seguro aos recursos.

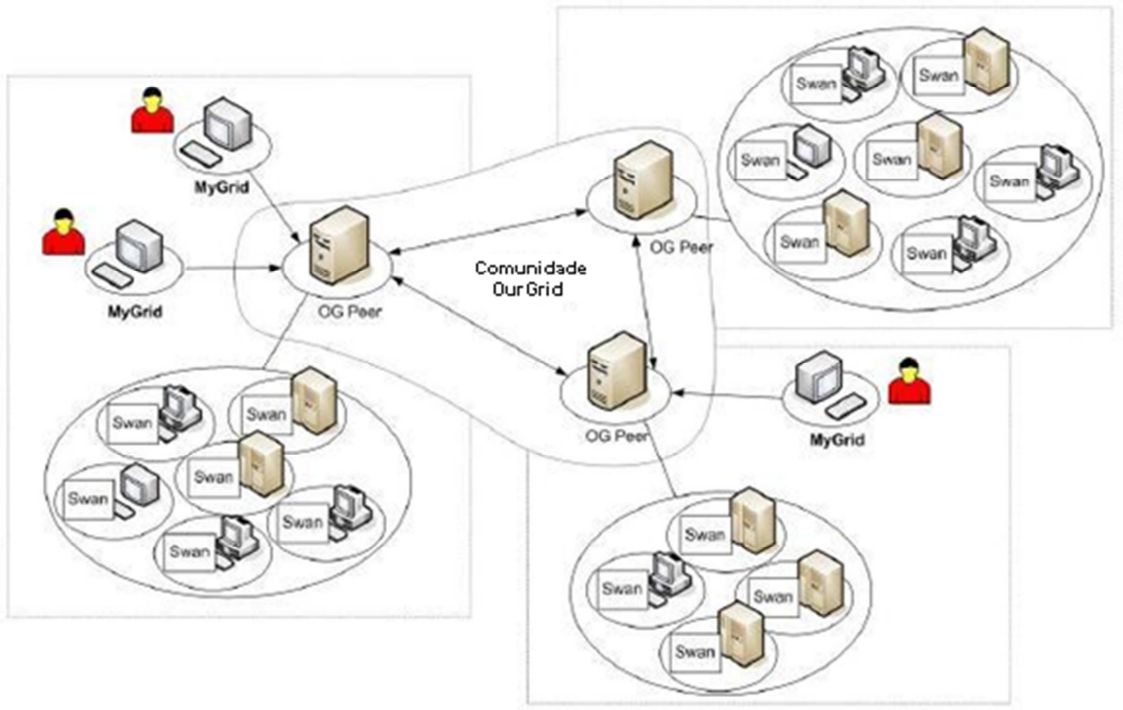

Figura 10: Principais componentes do OurGrid

Fonte [44]

3.3.1 MyGrid

O *MyGrid* [56] é responsável por efetuar a interface entre o usuário e a grade computacional. Quando esse usuário precisa executar uma aplicação é necessário que ele tenha que se envolver o menos possível com detalhes da grade. Sendo assim, um dos objetivos desse componente é simplificar os processos de instalação necessários para que o usuário possa fazer parte desse ambiente distribuído.

Para que o usuário possa executar uma aplicação na grade é necessário que ele possua uma máquina onde seja coordenada tal execução e outra onde a execução realmente ocorra. Essas máquinas não denominadas *home machine* e *grid machine*, respectivamente.

De maneira a obter a interoperabilidade necessária, os recursos disponíveis são representados através de interfaces. A GuM (*Grid Machine Interface*) é uma abstração para uma máquina, enquanto a GuMP (*Grid Machine Provider Interface*) representa um conjunto de máquinas, o qual é gerenciado por terceiros . Essas abstrações são bastante importantes visto que para adicionar novos recursos à grade como, por exemplo, uma máquina, basta implementar tal interface.

Conforme ilustrado na figura 11, quando se deseja utilizar recursos de um conjunto, o componente *escalonador* solicita os recursos ao GuMP, que após ganhar acesso a tais recursos devolve ao escalonador as interfaces (GuMs) para tais máquinas.

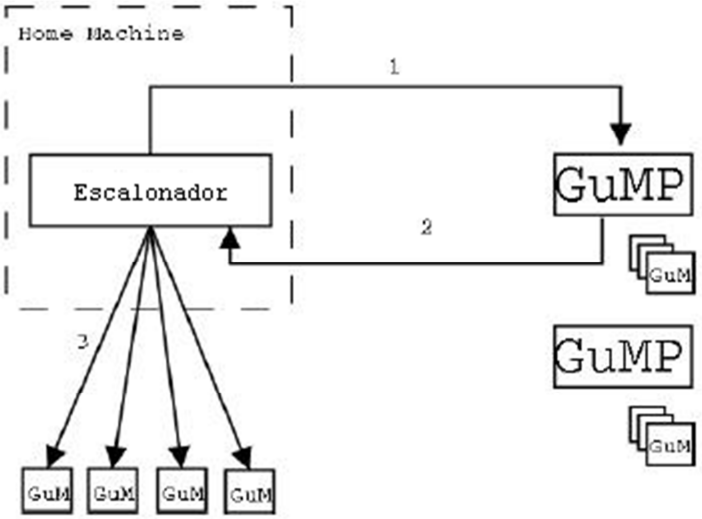

Figura 11: Arquitetura do MyGrid

Adaptado de [56]

O *MyGrid* oferece três implementações nativas para GuM's:

- *UserAgent:* Interface desenvolvida em Java que permite que sejam instaladas aplicações em uma *Grid Machine*.
- *GridScript:* Fornece as mesmas funcionalidades da *UserAgent*, porém na forma de *scripts*.
- *Globus:* Desenvolvido para fornecer interoperabilidade entre o *MyGrid* e o *Globus Toolkit*.

3.4 BOINC

O projeto BOINC (*Berkeley Open Infrastructure for Network Computing*) é mantido pela universidade de Berkeley e viabiliza a construção de grades computacionais *desktop* para aplicações que necessitem de grandes armazenamentos de dados ou potência computacional [11].

Muitos projetos de pesquisa fazem uso desta plataforma para viabilizar seus trabalhos por meio da utilização de computadores pessoais voluntários espalhados por todo o mundo. Exemplos destes projetos são [57]:

- *SETI@home:* busca de vida extraterrestre inteligente por meio do processamento de sinais originados do espaço.
- *Rosetta@home:* determina formas tridimensionais de proteínas
- *LHC@home:* auxilia cientistas na análise de dados referentes à aceleração de partículas realizada pelo CERN.
- *Climaprediction.net:* realiza previsões de tempo e análise de precisão de modelos de previsão.

Dados obtidos por meio do site do projeto [57] no dia 27 de janeiro de 2012 demonstram a potência computacional ofertada por cada país através do compartilhamento dos recursos dos computadores pessoais de seus usuários. Como exemplos, os Estados Unidos apresentam uma colaboração de 2.315.306 *GigaFlops*, o Brasil oferta uma potência computacional na ordem de *27.102 GigaFlops* e a Alemanha 677.759 *GigaFlops*.

A instituição de pesquisa que deseja criar um projeto que faça uso da grade computacional necessita de um ou mais servidores (utilizando Linux, Apache, MySQL, Php e Phyton) para realizar o gerenciamento e distribuição das tarefas.

Por meio de um banco de dados são realizados os gerenciamentos de unidades de trabalho, resultados, contabilizações, etc. Clientes que desejam realizar o processamento de tarefas efetuam a invocação de um serviço *Web* existente no servidor responsável pelo escalonamento que envia ao cliente o código a ser executado. Ao término desta execução, o resultado é retornado. Quando arquivos devem ser retornados os clientes realizam o *upload* por meio de conexões HTTP a um servidor de arquivos.

Um voluntário que deseja colaborar com um projeto precisa simplesmente realizar o *download* e instalação da aplicação no seu computador. A execução pode ocorrer de três formas:

- *Descanso de tela*: quando o computador estiver ocioso, gráficos gerados pelo processamento são apresentados ao usuário, conforme figura 12, que ilustra o exemplo do SETI@home.

- *Serviço do sistema operacional*: a aplicação é executada em *background* e *logs* são armazenados.
- *Aplicação desktop*: o usuário pode verificar os projetos que ele colabora, recursos utilizados, entre outros.

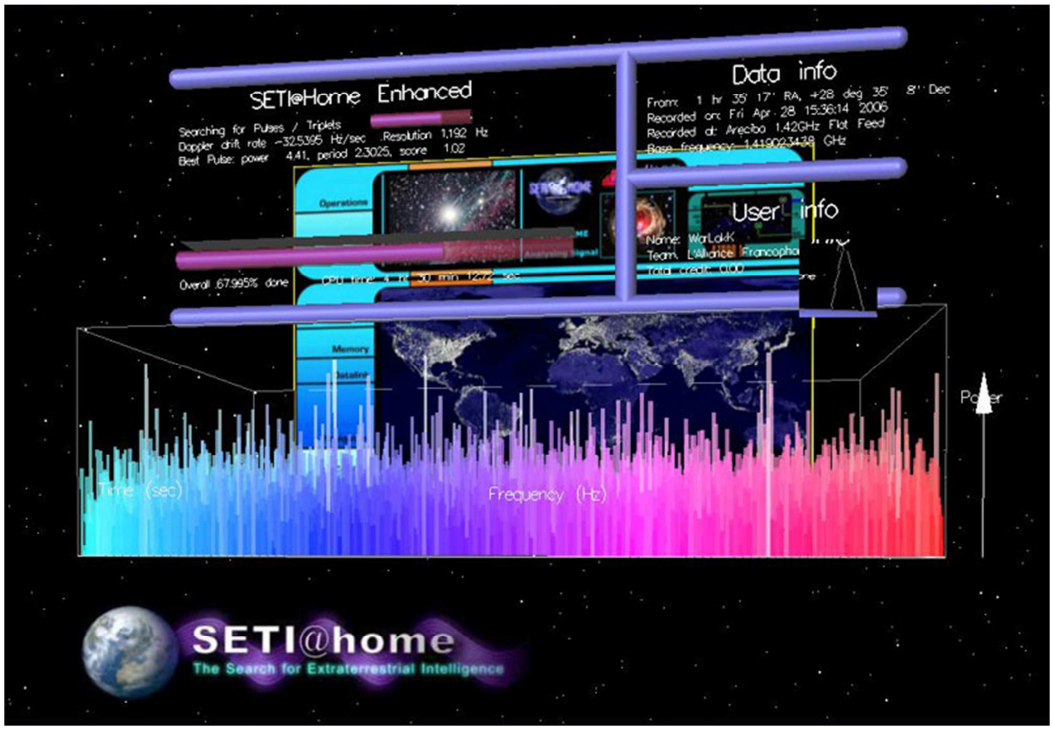

Figura 12: Exemplo de descanso de tela do SETI@home.

Fonte [12]

O BOINC se baseia na ideia de que a execução de aplicações da grade não deve interferir no desempenho dos aplicativos utilizados pelo usuário. Sendo assim, são definidos critérios que determinam quando e como os recursos locais podem ser utilizados por aplicações de terceiros.

No momento em que o cliente realiza a requisição de uma aplicação a ser executada, o servidor envia uma unidade de trabalho que é composta, entre outros, pela aplicação propriamente dita, arquivos de entrada, um descritor de utilização de recursos e um tempo limite para conclusão.

A execução de aplicações em computadores desconhecidos pode apresentar problemas relacionados ao mau funcionamento do sistema voluntário ou comportamento malicioso do usuário que pode modificar o resultado para gerar falsas conclusões. Para evitar esse tipo de problema, o escalonador envia a mesma unidade de trabalho para diferentes clientes e quando os resultados são retornados é feita uma

comparação entre eles. Na existência de um consenso o resultado é considerado verdadeiro. Caso contrário, um novo envio e processamento são realizados.

A experiência que os pesquisadores do BOINC tiveram como o projeto SETI@home demonstra, de maneira geral, a existência de interesse dos voluntários em colaborar com processamentos dessa natureza. De maneira a incentivar a participação são concedidos créditos a cada voluntário em função das características de seu computador (CPU, armazenamento, etc) e o tempo de uso.

Os créditos são contabilizados pelo servidor do projeto a cada resultado correto recebido. Na página do projeto disponível na *Web* é apresentado um *ranking* dos voluntários. Foi observado que muitos usuários se importam com sua colocação, o que pode motivar sua participação cada vez mais efetiva.

3.5 QADPZ

QADPZ é um *framework* para grades computacionais *desktop* proposto na tese de doutorado de Nicolae-Zoran Constantinescu-Fülöp [45]. Esse trabalho tem como foco aplicações científicas e de visualização que podem ser divididas em tarefas menores a serem executadas de forma paralela por computadores existentes em uma rede local ou na Internet.

Baseado no modelo mestre-trabalhador, o QADPZ dá suporte à execução de aplicações desenvolvidas nas linguagens C, C++, Lisp, Python e Java. Além disso, dá também suporte a aplicações que utilizam a biblioteca de troca de mensagens MPI.

O trabalho tem uma forte preocupação com a maneira de realizar a distribuição das unidades de trabalho entre os voluntários. Em grades computacionais *desktop* é comum que a divisão do trabalho em tarefas menores possa ser realizada de forma estática, onde as tarefas são criadas no início da execução do nó mestre e a distribuição é realizada de maneira uniforme entre os voluntários.

No entanto, isso apresenta algumas questões. Entre elas se destacam o fato de que é possível que o número de tarefas seja grande o suficiente para inviabilizar o armazenamento na memória do servidor e a possibilidade de que nem sempre estejam disponíveis todos os dados para criar tais tarefas no momento inicial.

O QADPZ permite que as tarefas possam ser geradas de forma dinâmica. Sendo assim, à medida que as respostas são obtidas dos escravos novas tarefas são criadas e enviadas. Além disso, a distribuição também é realizada e gerenciada de forma dinâmica, pois a determinação do *timeout* de execução de uma tarefa por um trabalhador pode ser alterada durante a execução e as tarefas podem ser reenviadas a outros voluntários.

No trabalho de Constantinescu-Fülöp também é feita uma observação sobre a forma de interação entre mestre e trabalhador. De maneira a aumentar a eficiência, ao invés do trabalhador atuar no papel ativo no recebimento da mensagem (*Pull*) ele trabalha no papel passivo. Sendo assim, uma vez que um voluntário se apresentou ao servidor, este é responsável pelo envio (*Push*) das unidades de trabalho. Isso aumenta a capacidade do mestre em determinar a forma de divisão e envio das mensagens, conforme citado anteriormente.

Com o objetivo de aumentar a eficiência dos trabalhadores, o QADPZ se baseia no fato de que quanto menos o escravo esperar por uma nova tarefa, melhor seu desempenho. Para isso, é feito uso de uma técnica de *pipelining* de tarefas, conforme ilustrado na figura 13.

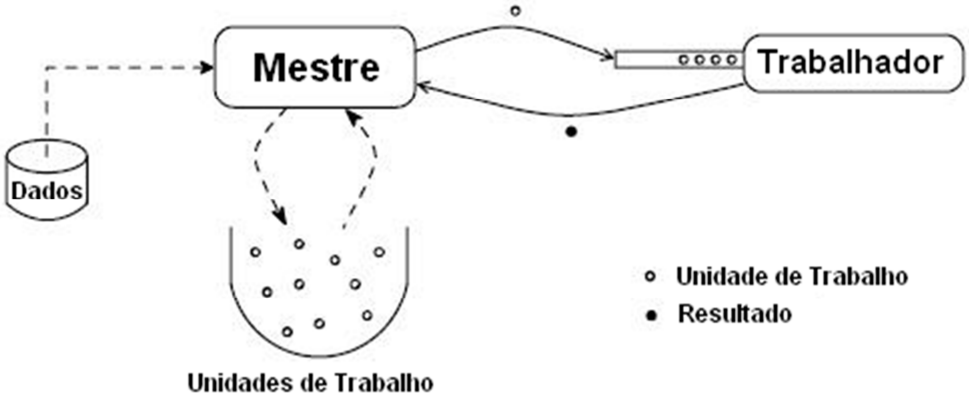

Figura 13: *Pipelining* de tarefas

Fonte [45]

É importante que cada escravo tenha no mínimo uma tarefa adicional àquela em execução no momento. Para isso, ao iniciar sua participação na grade o escravo recebe um conjunto inicial de tarefas e ao enviar o resultado de um processamento uma nova tarefa é enviada.

3.6 SKTAKI

Conforme foi descrito na seção sobre o BOINC, em grades computacionais que utilizam este *middleware*, cada projeto tem seu servidor que é responsável por realizar as tarefas do mestre em uma arquitetura mestre-trabalhador. Com essa arquitetura, cada projeto configura uma grade computacional *desktop* própria e independente.

Um determinado projeto de pesquisa pode requerer uma grade computacional com um número bastante grande de voluntários e este número pode ser obtido por meio da união de grades existentes em outros projetos [46]. O SKTAKI permite que grades computacionais *desktop* providas pelo BOINC sejam organizadas de uma forma hierárquica, que permita que níveis inferiores requisitem a níveis superiores tarefas a serem executadas.

Nesse tipo de abordagem um mesmo projeto pode atuar nos papéis de provedor e consumidor de recursos. Ele atua no papel de provedor quando requisita junto ao servidor superior tarefas a serem distribuídas entre seus “trabalhadores”. O papel de consumidor é determinado quando o servidor envia a seus voluntários ou servidores subordinados as tarefas oriundas de seu próprio projeto.

3.7 TVGrid

TVGrid é uma arquitetura de computação em grade concebida para utilizar recursos ociosos existentes em um sistema de televisão digital [19].

Uma vez que um receptor doméstico (também conhecido por *set-top box*) é um equipamento computacional adaptado para as necessidades relacionadas ao sistema de televisão digital, ele possui unidade de processamento, memória, entre outros, o que possibilita que ele execute aplicações. Dessa forma, uma estação de TV é responsável por enviar aplicações (juntamente com os fluxos de áudio e vídeo) aos receptores, onde são executadas e os resultados são enviados para a estação por meio do canal de retorno.

Essa proposta toma como pressuposto a inexistência de comunicação entre os receptores. Sendo assim, só existe a comunicação entre a estação de TV e o receptor. Por isso, as aplicações são caracterizadas como *Bag-of-Tasks* (BoT).

Na arquitetura são encontrados os seguintes componentes:

- *Task Server*: Responsável por multiplexar as aplicações no fluxo de vídeo a ser transmito aos receptores.
- *Broadcast Network*: Meio de comunicação utilizado para conduzir o fluxo até os receptores.
- *Set-top box*: Receptor responsável pela execução da aplicação.
- *Canal de retorno*: Meio de comunicação utilizado para conduzir a resposta do receptor à estação de TV.

O servidor de tarefas (*Task Server*) pode possuir diversas tarefas a serem enviadas e processadas pelos receptores. Essas tarefas são inseridas no carrossel de dados, sendo o receptor o responsável pela definição de qual tarefa deve ser executada. Para isso, o carrossel realiza o transporte de uma aplicação denominada *Trigger Application*, que é baixada pelo receptor e executada automaticamente. Essa aplicação tem a função de definir de forma aleatória qual tarefa será executada, realizar a cópia e requisição de execução da tarefa.

O comportamento do telespectador pode influenciar diretamente a execução das tarefas existentes em seu receptor, visto que o canal pode ser trocado ou até mesmo o *set-top box* desligado. Para solucionar isso é proposto o uso da redundância, de forma que uma mesma tarefa é executada em mais de um receptor.

Tendo em vista a possibilidade de existência de um número considerável de tarefas a serem executadas, inserir todas elas no carrossel de dados e transmitir via *broadcast* a todos os receptores pode se tornar uma solução inviável.

O *Task Server* possui um repositório com todas as tarefas a serem executadas. Em função da capacidade do carrossel de dados as tarefas são selecionadas para serem enviadas aos receptores. No entanto, uma tarefa só é excluída do repositório quando, no mínimo, um receptor enviou a resposta referente ao seu processamento.

3.8 OddCI

O OddCI (*On-Demand Distributed Computing Infrastructure*) é um projeto de pesquisa que visa prover um ambiente que forneça ao usuário a flexibilidade da computação em grade, juntamente com a elasticidade da computação em nuvem [20].

De maneira a alcançar a escalabilidade requisitada por diversas aplicações, o OddCI deseja fazer uso de todo equipamento conectado à Internet, se destacando entre eles os receptores de TV Digital.

No ambiente computacional provido pelo OddCI não há necessidade de registro dos participantes. Quando necessária, a infraestrutura é construída dinamicamente por meio do canal de difusão (*broadcasting*) do sistema de TV digital. Por isso, dá-se o nome de "*On-Demand*".

Conforme pode ser visto na figura 14, os Agentes de Processamento (tradução de *Processing Node Agents*) são os receptores responsáveis por realizar o processamento de tarefas enviadas via difusão. A comunicação deles com os demais componentes é realizada de duas formas:

- *Canal de Difusão*: utilizado para transmitir dados aos receptores por difusão juntamente com fluxos de áudio de vídeo.
- *Canal Direto*: canal bidirecional que pode ser utilizado pelos receptores para obter dados de entrada e retornar resultados.

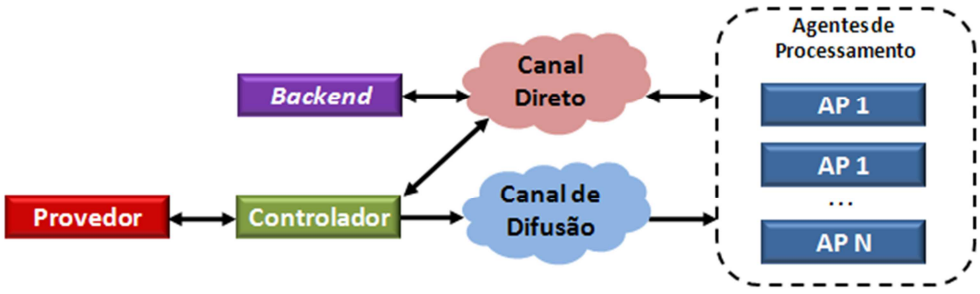

Figura 14: Arquitetura básica do OddCI

Fonte [20]

O componente *Provedor* é responsável por determinar a construção de ambientes de processamento. As instruções são passadas ao *Controlador* que envia dados de controle e a aplicação propriamente dita via difusão aos receptores.

O *Backend* atua diretamente com os receptores. Ele é responsável pelo fornecimento de dados de entrada, bem como seu escalonamento, processamento consolidado dos resultados enviados pelos receptores, entre outros.

Quando um usuário necessita executar uma aplicação nesse ambiente ele envia uma requisição ao *Provedor*. Essa requisição é enviada ao *Controlador,* que realiza a alocação do ambiente computacional junto aos receptores. Isso é feito por meio do envio de uma mensagem de *weakup* que ativa os *set-top boxes* para realizar o processamento.

No momento que um receptor recebe uma solicitação para criação de um ambiente de processamento ele inicia um processo de envio de seu estado para o *Controlador,* que os encaminha ao *Provedor* para que este seja capaz de determinar o estado global do ambiente computacional.

Se um receptor está ocioso e pode realizar o processamento requisitado é criada uma instância em um ambiente de execução para a aplicação do usuário. Essa requisição ocorre dentro de um ambiente seguro de execução (*sandbox*) padrão para grades computacionais.

3.8.1 Desempenho do Receptor

Em [20] foi realizada uma avaliação de desempenho da potência computacional de um receptor digital, equipamento com processador ST7109 e com 32 MB de memória.

As mesmas tarefas que foram submetidas nesse *set-top box* também foram executadas em um PC *Pentium Dual Core*, com processador de 1.6MHz e 1 GB de memória RAM. Com um intervalo de confiança de 90% foi constatado que o receptor apresentou um desempenho 26 vezes pior que o computador.

Este pior resultado era algo esperado, tendo em vista a baixa potência computacional do receptor se comparado ao computador pessoal. No entanto, o grande número de elementos de processamento que pode ser ofertado em um ambiente de TV pode, segundo os autores, tornar o uso dos receptores bastante atrativo.

3.9 Considerações Finais

É comum a utilização de SOA nos *middlewares* para grades computacionais. Além disso, propostas utilizando agentes móveis também são encontradas na literatura. A junção de ambos pode traduzir em uma proposta onde as vantagens podem ser concentradas para produzir resultados interessantes.

Além disso, também foi possível observar a possibilidade do uso dos receptores de TV na construção de uma grade computacional *desktop*. Embora esta tese utilize mecanismos diferentes para realizar o compartilhamento dos recursos dos receptores, esta referência se torna importante na questão da viabilidade do projeto.

Capítulo

4

Televisão Digital

4.1 Considerações Iniciais

A possibilidade de utilizar receptores de TV na composição de uma grade computacional *desktop* é bastante interessante para a área, tendo em vista o potencial grande número de elementos de processamento que podem ser ofertados.

Este capítulo apresenta um estudo sobre os padrões de televisão digital existentes, demonstra as tecnologias envolvidas e dá o embasamento necessário para que possa ser determinada a forma de utilização desses recursos na composição de uma grade computacional.

4.2 Histórico

Em sua primeira transmissão, realizada no ano de 1950 pela extinta TV Tupi, a Televisão dava seu primeiro passo no Brasil, chegando por meio de imagens em preto e branco nas primeiras residências brasileiras [58]. Assis Chateubriand, fundador da emissora, comprou com recursos próprios cerca de 200 televisores e os espalhou pela cidade de São Paulo.

Alguns testes de transmissão a cores foram realizados pela TV Excelsior e pela TV Tupi em 1962 e 1963, respectivamente. Em 1970, alguns poucos telespectadores já puderam acompanhar a Copa do Mundo de Futebol por meio de imagens coloridas, disponibilizadas em algumas salas montadas pela Embratel nas cidades de São Paulo, Rio de Janeiro e Brasília. No entanto, somente em 1972 a TV Globo realizou a primeira transmissão oficial de televisão em cores no país, definindo a segunda etapa histórica dessa tecnologia no Brasil.

No dia 2 de dezembro do ano de 2007, o terceiro marco era consolidado por meio da primeira transmissão de sinais de televisão digital na cidade de São Paulo. Essa nova fase, provida pelo Sistema Brasileiro de Televisão Digital (SBTVD), tem por objetivo levar ao telespectador imagens de alta definição, som com qualidade digital e interatividade.

Segundo o decreto presidencial número 4.901 de 26 de novembro de 2003 [59], o SBTVD foi concebido como uma ferramenta capaz de promover, além dos recursos de entretenimento e cultura oferecidos pelo sistema analógico, diversos outros benefícios à população como, por exemplo, a democratização da informação por meio da inclusão digital e a educação à distância.

A adoção do formato digital na geração, transmissão, recepção e reprodução de programas de TV torna possível a utilização de mecanismos da computação que auxiliam de forma significativa a obtenção de resultados melhores que aqueles oferecidos pelo modelo analógico atual. Como exemplo, pode-se citar a utilização da compressão de dados de maneira que o conteúdo transmitido possa ser reduzido. Nesse caso, conforme Lemos [60], um mesmo canal físico de transmissão terrestre (UHF/VHF), que no modelo analógico comporta apenas uma programação, é capaz de realizar o tráfego de 4 programas no modelo digital, com qualidade superior de som e imagem. Dessa forma, além do aumento da oferta de opções de programação para o telespectador, o aumento da quantidade de informações suportadas pelo canal físico permite ainda a transmissão de dados digitais de qualquer natureza como, por exemplo, programas aplicativos.

A mobilidade é uma questão bastante interessante que o sistema digital de televisão pode oferecer. A tecnologia que começa a ser implantada no Brasil permite que aparelhos portáteis como celulares, PDAs, entre outros, possam ser utilizados na recepção e, por consequência, reprodução dos programas de televisão, proporcionando assim uma maior flexibilidade ao telespectador [18].

No modelo analógico o telespectador possui um papel totalmente passivo, visto que ele tem a possibilidade de apenas assistir o conteúdo transmitido pela emissora. No entanto, a presença da interatividade modifica a forma com a qual a população pode fazer uso do aparelho televisor, tornando possível que por meio de seu controle remoto o usuário envie opiniões sobre o programa assistido, participe de enquetes, votações, entre outros [61].

4.3 Arquitetura Básica de um Sistema de Televisão

O sistema analógico de televisão utilizado atualmente é composto por alguns subsistemas, onde cada um deles tem uma função distinta e bem definida [60]. Conforme ilustrado na figura 15, o primeiro subsistema, a Central de Produções, é responsável pela produção de um programa e consiste, entre outros, na gravação e edição das imagens. Os processos encontrados nesse subsistema podem ser realizados pela própria emissora ou por uma empresa distinta, como uma produtora de vídeo, por exemplo. Uma vez que o programa está pronto é necessária a transmissão entre o estúdio e o subsistema de radiodifusão.

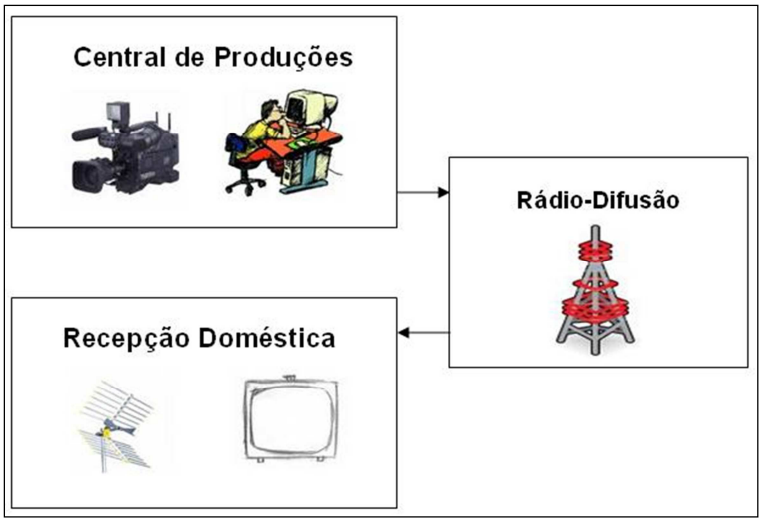

Figura 15: Arquitetura do Sistema Analógico Atual de Televisão
Adaptado de [60]

O subsistema de radiodifusão tem a responsabilidade de levar o sinal televisivo até o equipamento de recepção (antena) do telespectador. Para a transmissão deste sinal pode-se fazer uso de satélites ou antenas espalhadas em pontos altos da cidade.

Por fim, o sistema de Recepção Doméstica recebe o sinal televisivo por meio de uma antena e de um receptor de analógico (que poder ser embutido ou não no aparelho televisor) e o envia para o monitor e autofalantes que apresentam o programa ao telespectador.

Atualmente, alguns processos do sistema de televisão ainda são realizados fazendo uso de tecnologias digitais e analógicas. Como exemplo, em determinados

casos a gravação, edição e armazenamento de imagens encontradas no subsistema da Central de Produções são feitas utilizando a tecnologia digital. No entanto, a radiodifusão e a recepção do sinal continuam fazendo uso do modelo analógico.

Quando um modelo digital é adotado todos esses subsistemas sofrem modificações, conforme apresentado na figura 16. No estúdio, as imagens que são geradas por uma câmera digital são submetidas a um codificador MPEG [62] que é responsável por aplicar técnicas de compressão de dados no sinal de vídeo digital, de maneira que se pode alcançar a taxa de 1 bit comprimido para cada 70 bits existentes na codificação sem compressão [60]. Os fluxos (*streams*) gerados por esse codificador podem ser armazenados em meios físicos como DVDs e discos rígidos que, ao contrário das fitas que têm acesso linear, possuem acesso direto.

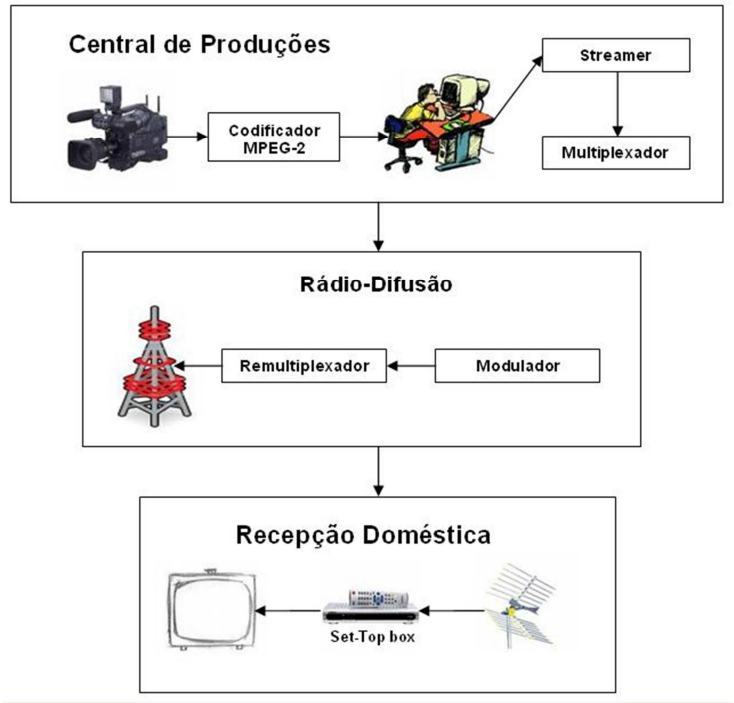

Figura 16: Arquitetura de um Sistema Digital de Televisão

Fonte [60]

Na Central de Produções são adicionados ainda dois novos elementos: o *Streamer* (gerador de fluxo) e o Multiplexador. O primeiro é responsável por gerar fluxos (*streams*) de dados que representarão o áudio e vídeo do programa transmitido. O multiplexador permite que diversos fluxos diferentes de dados possam ser organizados em um único fluxo que será transmitido via *broadcasting*.

Uma vez que os sinais televisivos digitais são compactados, há uma economia da banda disponibilizada por um determinado canal físico de transmissão. Sendo assim, é possível que nesse mesmo canal possam ser transmitidos até 4 programações simultâneas, dependendo da definição de vídeo utilizada. Para isso é necessária a inserção de um remultiplexador que tem a função de multiplexar os fluxos (*streams*) gerados por uma ou mais Centrais de Produção.

Na Recepção Doméstica é necessária a existência de um terminal de acesso que seja capaz receber o sinal digital, decodificá-lo, reproduzir os fluxos de áudio/vídeo e enviá-los ao aparelho televisor. Esse terminal de acesso é denominado *Set-Top Box*.

O *Set-Top Box*, além das funções citadas no parágrafo anterior, pode possuir ainda a capacidade de executar programas. Sendo assim, ele possui características básicas semelhantes àquelas encontradas em um computador pessoal.

4.4 Sistemas MPEG-2

No contexto da televisão digital, um programa consiste em elementos digitais que juntos representam uma entidade de entretenimento transmitida e apresentada ao telespectador. Um programa pode necessitar que vários fluxos de dados (vídeo, áudio e dados propriamente ditos) sejam transmitidos para que, de forma conjunta, sejam decodificados no receptor e apresentados ao telespectador [63].

Cada fluxo de dados necessário a um programa é denominado de Fluxo Elementar (*Elementary Stream* - ES). A especificação MPEG-2 determina como diversos Fluxos Elementares de dados de diversos programas diferentes podem ser multiplexados de maneira a gerar um único fluxo de dados a ser transmitido. Esse fluxo único é denominado de Fluxo de Transporte (*Transport Stream* – TS) [64] [63].

O Fluxo de Transporte consiste em uma sequência de pacotes de transporte de tamanho fixo de 188 bytes. Conforme a figura 17, cada pacote é dividido em cabeçalho, campo de adaptação e dados.

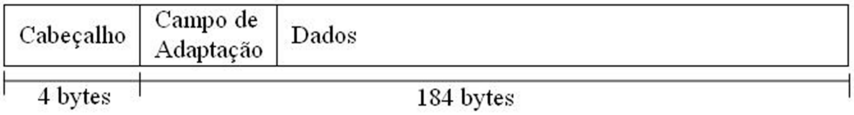

Figura 17: Formato do pacote de transporte

Fonte [65]

Conforme a figura 18, o cabeçalho do pacote de transporte inicia-se com o campo *Sync Byte*, o qual sempre possui o valor 47 em hexadecimal. Esse campo é utilizado pelo decodificador para determinar onde um novo pacote começa dentro do fluxo. Embora seja possível que esse valor se repita no restante do pacote, ele sempre é iniciado a cada 188 bytes, o que possibilita que seja facilmente diferenciado [64] [63] [65]. Como um mesmo Fluxo de Transporte pode conter diversos Fluxos Elementares, existe o campo PID que tem como objetivo identificar a qual Fluxo Elementar o pacote pertence.

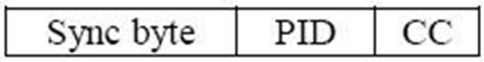

Figura 18:Cabeçalho do pacote de transporte

Fonte [65]

O campo CC (*Continuity Counter Field*) é utilizado para dar um número sequencial para cada pacote de transporte gerado. Isso é útil, por exemplo, para identificar a perda de um pacote.

4.4.1 PSI

No momento em que um decodificador é iniciado ele não tem nenhum conhecimento prévio quanto ao conteúdo do Fluxo de Transporte que será recebido [64]. Sendo assim, é preciso que seja definida a função de cada um dos Fluxos Elementares existentes (representados pelo PID do cabeçalho do pacote de transporte), quais fluxos de áudio pertencem a um determinado vídeo, etc. Essas informações são fornecidas pelo PSI (*Program Specific Information*), exemplificado pela figura 19.

Uma tabela denominada PAT (sigla do inglês *Program Association Table*) é transportada (em intervalos regulares) em um fluxo com PID igual a zero. Sendo assim, o decodificador deve primeiramente localizar e montar essa tabela para que as informações sobre os programas transmitidos sejam obtidas. Cada entrada dessa tabela armazena o nome de um programa e o PID do fluxo que transporta seu PMT (*Program Map Table*).

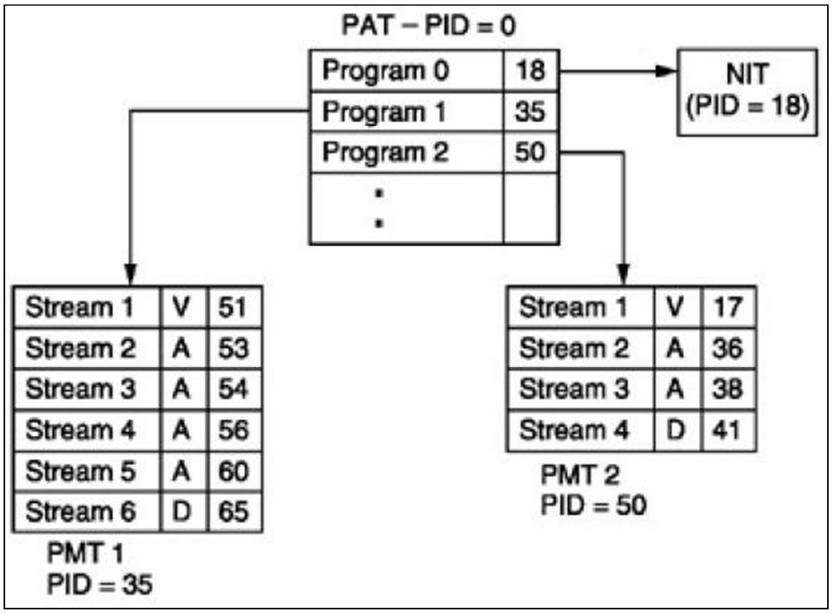

Figura 19: Exemplificação de um PSI

Fonte [64]

A PMT, por sua vez, tem a função de informar os PIDs dos Fluxos Elementares pertencentes a um determinado programa, bem como especificar o conteúdo de cada um desses fluxos, que pode ser áudio (A), vídeo (V) ou dados (D).

4.4.2 DSM-CC

Durante a transmissão de um programa, muitas vezes é importante que, além de áudio e vídeo, dados propriamente ditos e aplicativos sejam transmitidos. Essa transmissão pode ser comparada com o processo de *download* existente na Internet. No entanto, quando uma transmissão por *broadcast* é utilizada, no momento que um receptor seleciona um determinado programa ele realiza a recepção do Fluxo de Transporte partindo do ponto atual. Sendo assim, partes dos dados a serem recebidos

(uma aplicação, por exemplo) podem ter suas transmissões realizadas em momentos anteriores ao inicio da recepção do Fluxo de Transporte, o que ocasionaria problemas em função dos dados perdidos.

O DSM-CC (*Digital Storage Media – Commandand Control*) permite a implementação de um Carrossel de Dados [21]. Essa especificação permite que os dados a serem transmitidos sejam organizados em módulos que são acomodados no Fluxo de Transporte sob um determinado PID. Esses módulos são regularmente repetidos de maneira que quando os dados transportados são requeridos, caso a transmissão já esteja em andamento, a repetição dos módulos iniciais é esperada para que a recepção possa ser realizada.

4.5 Padrões de Televisão Digital Interativa

Um sistema de televisão digital interativa envolve inúmeros processos até que uma programação possa ser recebida e apresentada ao telespectador [60]. Se os processos forem organizados no formato de uma arquitetura de camadas, para cada uma dessas camadas é possível que sejam encontradas mais de uma solução tecnológica capaz de realizar o trabalho necessário. A figura 20 apresenta a organização em camadas de um sistema de televisão digital interativa.

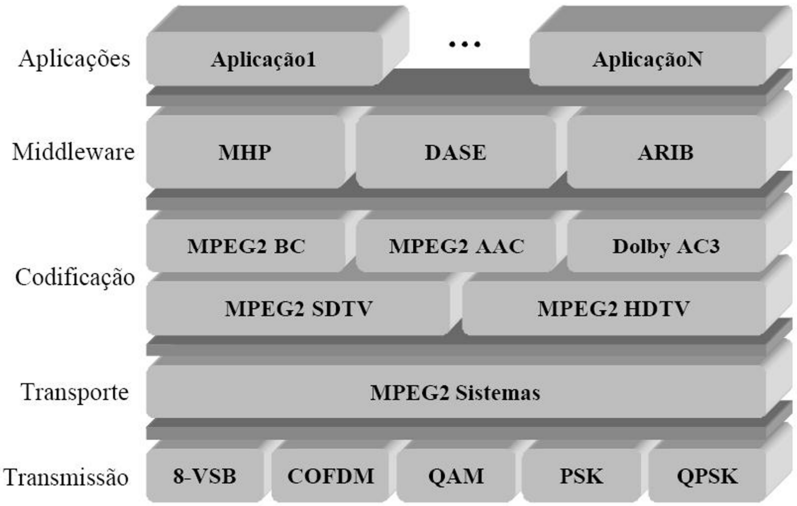

Figura 20: Organização em camadas de um sistema de televisão digital interativa

Fonte [60]

As aplicações consistem nos programas enviados via *broadcasting* aos receptores, onde deverão ser executados. Uma vez que há a possibilidade da existência de heterogeneidade de *hardware* e sistema operacional do terminal de acesso (*Set-Top Box)* do telespectador, uma camada de software que apresente APIs aos aplicativos se faz necessária. Essa camada é denominada de *middleware*.

A codificação tem a responsabilidade de representar áudio e vídeo em formato digital, realizando funções como, por exemplo, a compactação dos dados para que possam ser transportados até o aparelho receptor.

A camada de transporte tem a função de multiplexar os diversos fluxos de áudio vídeo e dados de maneira a construir um único fluxo a ser transmitido. Por fim, a camada de transmissão é responsável pelo processo de radiodifusão do sinal televisivo, envolvendo inclusive a recepção pelo equipamento do telespectador.

Quando um sistema de televisão digital interativa é implantado em um país, é necessário que todos os fabricantes de equipamentos envolvidos no sistema (desde a emissora até o telespectador) produzam seus produtos de maneira que sejam capazes de interagir entre si. Para isso, a adoção de padrões é de uma importância bastante grande, pois ela tem como objetivo a geração de documentos que especifiquem e orientem os fabricantes quanto às tecnologias que devem ser adotadas em cada processo. Atualmente existem três padrões mundiais de televisão digital interativa que são apresentados a seguir.

4.5.1 ATSC

O padrão ATSC (*Advanced Television Standard Committee*), também conhecido como padrão americano, pode ser utilizado para transmissões terrestres, via satélite e via cabo [60]. Além dos Estados Unidos, outros países utilizam esse padrão como, por exemplo, Canadá e Coréia do Sul.

A transmissão do sinal pode ser terrestre, via satélite ou via cabo utilizando as modulações 8-VSB, QPSK ou QAM, respectivamente [60]. A modulação adotada pela transmissão terrestre inviabiliza a recepção do sinal por meio de dispositivos móveis.

A recomendação MPEG-2 Sistemas é utilizada na camada de transporte para multiplexar os sinais de áudio, vídeo e dados para que possam ser transmitidos em um único canal físico e posteriormente demultiplexados pelo receptor do telespectador.

Embora o sistema ATSC defina formatos de vídeo para HDTV e SDTV [66], nos Estados Unidos, em função de seu modelo de negócios a adoção do padrão foi voltada para a utilização do HDTV [60]. Desta forma, na camada de codificação de vídeo é utilizado o MPEG-2 HDTV. Para a codificação de áudio utiliza-se o Dolby AC-3.

A figura 21 apresenta as tecnologias empregadas em cada uma das camadas no sistema ATSC.

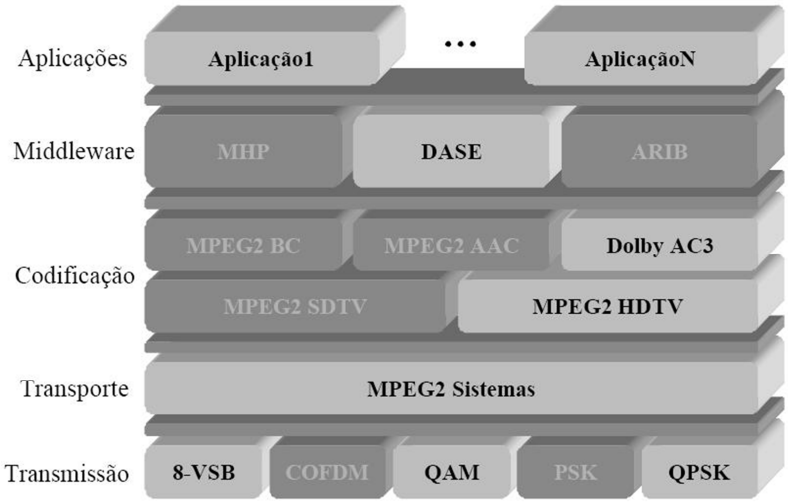

Figura 21: Organização em camadas do padrão ATSC

Fonte [60]

4.5.2 DVB

O padrão DVB (*Digital Vídeo Broadcasting*), também conhecido como padrão europeu de televisão digital, é adotado nos países da união europeia e em outros, tais como Austrália, Nova Zelândia e Malásia [60].

Na Europa, esse sistema é utilizado também para oferecer acesso à Internet [67]. Para isso, fazem uso de equipamentos convencionais para acesso a essa rede como, por exemplo, *modems*, e utilizam o sinal recebido por *broadcast* para prover um recurso adicional de *downstream*.

A figura 22 apresenta as tecnologias empregadas por esse padrão em cada uma das camadas do sistema de televisão digital. Para a codificação de áudio e vídeo são utilizadas as recomendações MPEG-2BC e MPEG2-SDTV, respectivamente [60].

Embora o padrão permita a utilização de alta resolução, na Europa foi adotada, em um primeiro momento, a resolução padrão.

Para a camada de transporte, de forma semelhante ao ATSC, o DVB faz uso da recomendação MPEG2 Sistemas.

A transmissão pode ser terrestre (DVB-T) utilizando modulação CDFDM, via Cabo (DVB-C) com modulação QAM, via satélite (DVB-S) fazendo uso da modulação QPSK, entre outras que fazem uso de micro-ondas.

Em função do tamanho do canal, até seis programas podem ser transmitidos simultaneamente em formado SDTV [60]. Isso se torna interessante em um continente como o Europeu onde a quantidade de países pequenos e próximos é maior, pois um mesmo programa pode ser transmitido ao mesmo tempo em vários idiomas utilizando um mesmo canal físico.

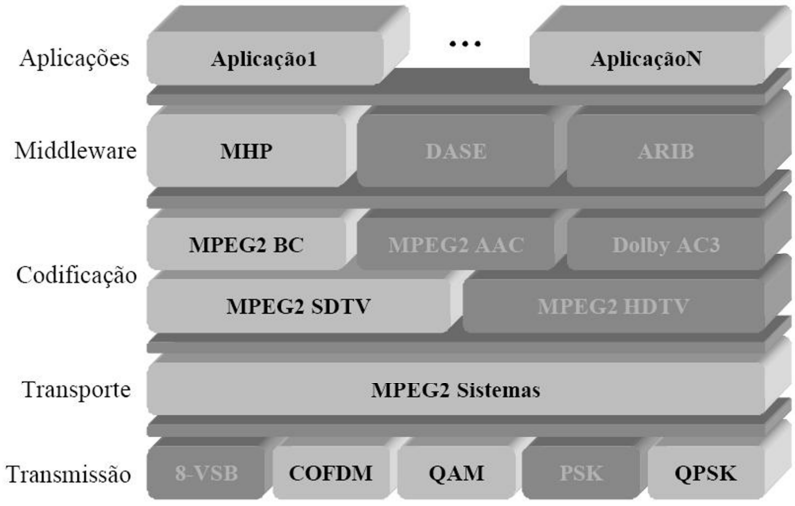

Figura 22: Organização em camadas do padrão DVB

Fonte **[60]**

4.5.3 ISDB

O ISDB (*Integrated System Digital Broadcast*), também conhecido como padrão japonês de televisão digital, teve o início de suas pesquisas em 1983 pela NHK tendo como objetivo um novo conceito de transmissão por *broadcasting* que oferecesse flexibilidade, expansibilidade e comodidade [68]. Com a especificação proposta em 1999, esse padrão é reconhecido como aquele que reúne o maior conjunto de

facilidades, pois permite alta definição de som e imagem (HDTV) e recepção móvel e portátil [60] [69].

Para a transmissão esse modelo adota três esquemas de modulação: COFDM, QAM e PSK, para transmissão terrestre, via cabo e via satélite, respectivamente [60]. Para a transmissão terrestre o padrão ISDB pode operar com canais de 6, 7 ou 8 MHz conseguindo taxas de transmissão que variam entre 3.65 a 23.23 Mbps.

A adoção de um sistema hierárquico de transmissão, que é conseguido por meio da divisão da banda dentro de um canal, permite a recepção da programação por equipamentos fixos e móveis [60] [69]. Dessa forma, o ISDB torna possível que o telespectador receba o sinal televisivo em equipamentos que variam desde um aparelho televisor fixo em sua casa até um *handheld* utilizado dentro de um ônibus.

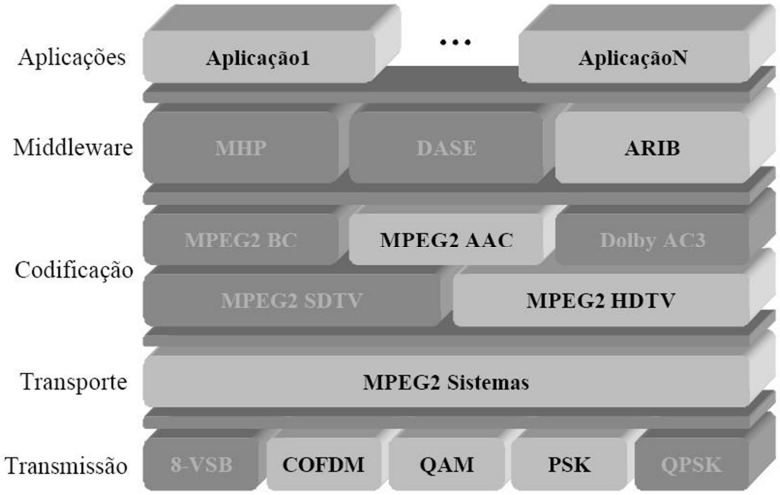

Figura 23: Organização em camadas do padrão ISDB

Fonte [60]

Conforme apresentado pela figura 23, na camada de transporte o ISDB faz uso das recomendações MPEG2-Sistemas para realizar a multiplexação e demultiplexação dos fluxos de áudio e vídeo.

Para realizar a codificação este padrão adota os as recomendações MPEG-2 AAC e MPEG-2, para áudio e vídeo, respectivamente. De maneira a permitir que cenários diferentes de utilização do padrão possam ser implementados, a codificação de vídeo pode ser realizada utilizando diferentes níveis de resolução.

4.5.4 ISDTV

No ano de 2003 foi iniciado o projeto SBTVD (Sistema Brasileiro de Televisão Digital) que tinha como objetivos reunir instituições de pesquisa que apresentariam propostas e soluções para compor um novo sistema de televisão digital [70]. Em meados de 2005 o governo federal optou pela adoção do ISDB (padrão Japonês de televisão digital) como base tecnológica para o padrão a ser implantado no Brasil.

Uma vez que bons resultados foram obtidos em trabalhos realizados por pesquisadores nacionais e as necessidades do Brasil diferem-se daquelas existentes em outros países, uma solução híbrida foi determinada. Essa solução adota os mecanismos de transmissão e recepção do ISDB e incorpora a camada de software desenvolvida pelo Brasil. Essa união de tecnologias culminou no padrão brasileiro de televisão digital denominado ISDTV (*International Standard for Digital Television*).

Na camada de codificação foi adotado o H.264 para vídeo. Este é o mais recente padrão desenvolvido em conjunto pelo MPEG (*Motion Pictures Experts Group*) e pela VCEG (*Vídeo Coding Experts Group*) [71]. Ele apresenta o maior avanço em compressão de vídeo e será utilizado no padrão brasileiro tanto para vídeos de alta definição quanto para àqueles de definição padrão. Para codificação de áudio, assim como é realizado no padrão japonês, será de feito o uso do MPEG-2 AAC para transmissões em *Stereo* e 5.1.

O *middleware* é um dos componentes que sofreu maior alteração quando comparado o ISDTV e o ISDB. De maneira a atender às necessidades encontradas no modelo brasileiro de televisão digital, foi adotada a solução brasileira denominada Ginga [72] [73] [74].

As camadas de transporte e transmissão não sofrem alterações quanto aos padrões adotados, mantendo os mesmos utilizados pelo ISDB.

A disposição das camadas no padrão brasileiro de televisão digital é apresenta na figura 24.

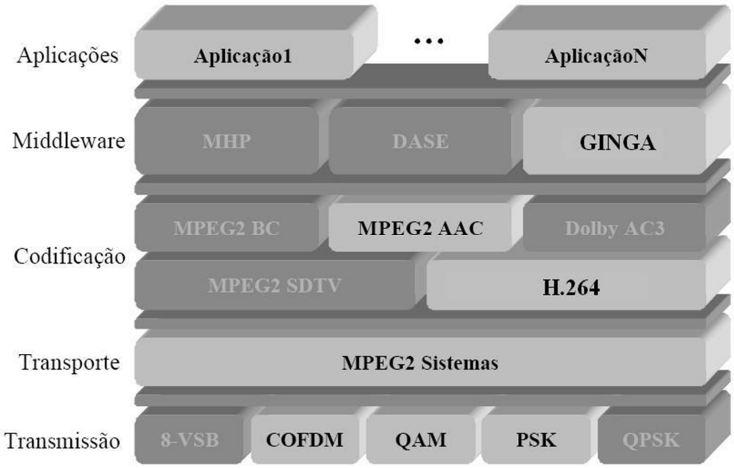

Figura 24: Organização em camadas do padrão ISDTV

Adaptado de [60]

4.6 Set-Top Box

Para que um aparelho de televisão seja capaz de realizar todos os procedimentos necessários para apresentar uma programação (processamento do Fluxo de Transporte, decodificação, apresentação de áudio e vídeo, execução de programas, comunicação com emissoras de televisão, entre outros) é preciso que o aparelho televisor tenha todos os recursos necessários incorporados.

Durante o período de transição muitos aparelhos analógicos continuarão em uso. Sendo assim, para que esses aparelhos possam ser utilizados no sistema de televisão digital, é necessária a existência de um terminal de acesso capaz de realizar as funções citadas no parágrafo anterior. Esse terminal de acesso é denominado *Set-Top Box* [60].

Uma vez que o *Set*-Top *Box* pode receber códigos executáveis, ele possui um sistema operacional e um ambiente de execução dos programas. Tendo em vista a possibilidade da existência de uma heterogeneidade dos componentes de *hardware* e *software* dos diferentes modelos de terminais de acesso desenvolvidos por fabricantes distintos, para que uma mesma aplicação enviada via *broadcasting* pela emissora possa ser executada em todos os receptores existentes, faz-se uso de um *middleware* (que é uma camada de *software* entre as aplicações e o sistema operacional) que, no caso do sistema brasileiro, é o Ginga [72] [73] [74].

4.6.1 Canal de Retorno

O canal de retorno é o meio de comunicação que permite que informações possam ser requisitadas e/ou enviadas, pelo *Set-Top Box*, à emissora ou a um provedor de serviços [60] [71].

Embora nenhuma tecnologia de enlace tenha sido definida pelo padrão ISDTV (sistema brasileiro de televisão digital), muitas soluções podem ser adotadas para tal função como, por exemplo, conexão a cabo, WiMax, Wi-Fi, ADSL, GSM, entre outros [71].

A figura 25 apresenta o emprego do canal de retorno em um sistema de televisão digital interativa. Uma vez que uma aplicação é recebida via *broadcasting* e executada pelo *Set-Top Box*, se durante sua execução se faz necessário o envio de informações para a Internet, isso pode ser realizado utilizando esse canal. É possível ainda que o canal de retorno seja utilizado como um meio auxiliar para a recepção de dados da emissora.

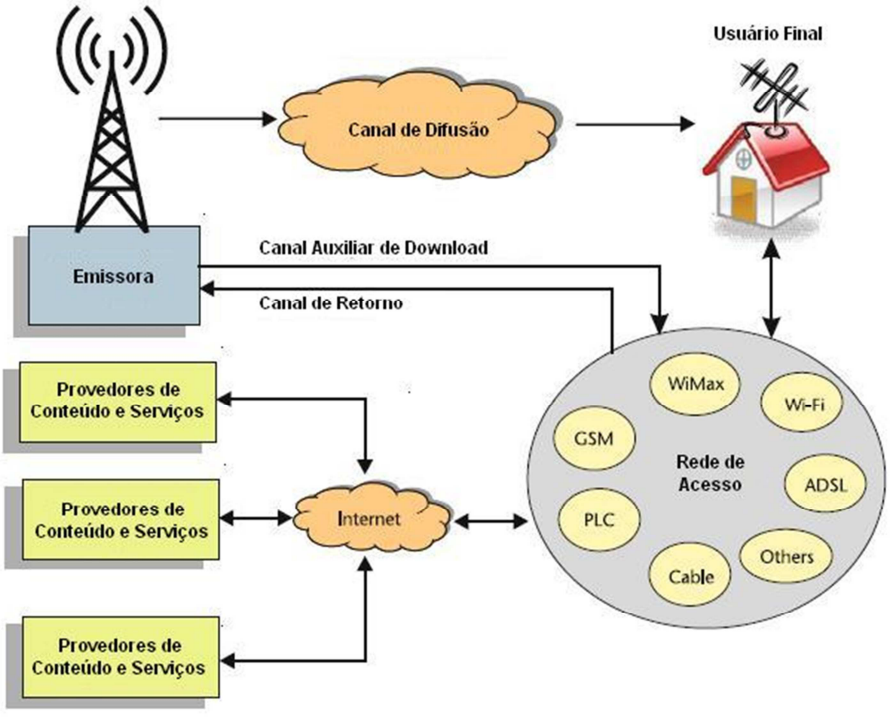

Figura 25: Emprego do canal de retorno em um sistema de televisão digital interativa

Fonte [71]

4.7 Middlewares

O *Set-Top Box*, conforme citado na seção anterior, pode possuir ainda a capacidade de executar programas aplicativos recebidos da emissora via *broadcast* [75]. Sendo assim, ele possui características básicas semelhantes a um computador pessoal.

Diferentes fabricantes de *Set-Top Box* podem adotar arquiteturas de hardware e *software* diferentes [60]. Sendo assim, a distribuição de um mesmo aplicativo implicaria na necessidade do desenvolvimento de várias versões, uma para cada modelo de receptor, o que inviabilizaria o sistema.

De maneira a solucionar esse problema, optou-se pela adição de uma camada de *software* que é localizada entre o sistema operacional e o aplicativo, conforme ilustrado pela figura 26. Essa camada tem como objetivo oferecer uma API genérica que é utilizada pela aplicação de maneira que detalhes referentes ao sistema operacional e o *hardware* sejam transparentes à aplicação.

Figura 26: Inserção de uma camada de software entre a aplicação e o sistema operacional
Fonte [60]

Essa API Genérica compõe o *middleware* de um *Set-Top Box*. Atualmente, algumas opções de *middleware* são oferecidas, conforme apresentado a seguir.

4.7.1 MHP

O MHP (*Multimedia Home Platform*) teve o início de seu desenvolvimento no ano de 1997 pelo projeto DVB, o padrão europeu de televisão digital [75]. No ano 2000 a primeira versão do *middleware* foi disponibilizada e após isso sofreu alterações em

função de observações realizadas por desenvolvedores de aplicações, fabricantes de *set-top boxes*, entre outros.

Uma vez que aplicações de natureza e propósitos diferentes podem ser executadas em receptores diferentes, ambos são classificados em três perfis [75]:

- *Enhanced Broadcasting*: Aplicações e receptores que oferecem somente interatividade local. Dessa forma, a única interface de rede necessária é aquela utilizada na recepção do sinal via *broadcast*.
- *Broadcasting Interativo*: Aplicações e receptores que oferecem interatividade que necessitam de comunicação bidirecional. Desta forma, um canal de retorno se faz necessário.
- *Acesso à Internet:* Aplicações e receptores que necessitam acessar serviços e conteúdo da internet.

O MHP define o ciclo de vida de uma aplicação que é executada no receptor. Quando uma programação de uma determinada emissora é selecionada pelo usuário, se um programa é transportado junto com os fluxos de áudio e vídeo este é automaticamente iniciado, permanecendo em execução até que seu término seja alcançado ou outra programação seja selecionada pelo usuário.

Esse *middleware* permite que um emissor que possua mais de um canal associe uma mesma aplicação ao seu conjunto de canais. Dessa forma, se o telespectador realiza a seleção de outro canal do mesmo conjunto, a aplicação que foi iniciada anteriormente permanece em execução.

A partir da versão 1.1 as aplicações podem ser carregadas, via HTTP, de um servidor *Web*. Sendo assim, é feito uso do canal de retorno para realizar a requisição e o *download* do aplicativo.

O MHP possibilita a execução de aplicativos escritos em linguagem procedural ou declarativa [60]. Para o primeiro caso, é feito uso da Java TV constituindo as aplicações DVB-J. Tecnologias baseadas em HTML são utilizadas no desenvolvimento de aplicações baseadas em linguagens declarativas, sendo essas aplicações classificadas como DVB-HTML.

*4.7.1.1 **GEM***

De maneira a permitir que o MHP pudesse ser adotado em outros padrões de televisão digital, foram retiradas algumas características que eram específicas para o padrão europeu (DVB) e criada a especificação GEM (*Globally Executable MHP*) [75].

Também adotado pelo ATSC e DVB, aplicações que são desenvolvidas para a especificação GEM podem ser executadas em qualquer um desses ambientes.

4.7.2 DASE

O middleware DASE (*DTV Application Software Environment*) é utilizado no padrão americano de televisão digital [60] [76].

As aplicações executadas por um receptor DASE podem ser escritas de forma procedural (utilizando Java TV) ou declarativa (utilizando uma extensão da linguagem HTML). Essas aplicações são executadas dentro do AEE (*Application Execution Engine*) que se recomenda que seja implementado utilizando uma máquina virtual Java [76].

4.7.3 ARIB

ARIB (*Association of Radio Industries and Businesses*) é uma organização japonesa que padroniza a camada de *middleware* para um sistema de televisão digital [60] [77].

Segundo Oliveira [77], a especificação ARIB-STD tem como objetivo permitir a oferta de serviços interativos como, por exemplo, informações adicionais de programas, diversos ângulos para visualização de um programa de televisão, interatividade, entre outros.

O desenvolvimento de aplicações multimídia é realizado por meio de uma linguagem declarativa denominada BML (*Broadcast Markup Language*), baseada em XML, que faz uso de linguagens como XHTML, CSS, *ECMAScript*, entre outros.

4.7.4 Ginga

Durante a concepção do Sistema Brasileiro de Televisão Digital o governo federal determinou que tal iniciativa não devesse ter como objetivo somente a tecnologia empregada, mas também contribuir para a promoção da inclusão social no

país [73]. Dessa forma, alguns requisitos foram determinados de maneira que os objetivos possam ser alcançados.

Algumas peculiaridades encontradas nos requisitos para o sistema nacional de televisão digital não eram de simples solução utilizando-se as opções de *middlewares* disponíveis no mercado. Dessa forma, optou-se pelo desenvolvimento de uma plataforma nacional de *software* denominada Ginga [72] [73] [74].

Esse *middleware* permite a execução de aplicações declarativas, utilizando a linguagem NCL [72] e de aplicações procedurais que fazem uso da linguagem Java em sua implementação [73]. A plataforma possibilita ainda a execução de aplicações híbridas que possuem partes procedurais e partes declarativas.

Para que ambas as naturezas de aplicações possam ser executadas, conforme ilustrado na figura 27, o Ginga disponibiliza os componentes *Motor de Execução* (composto por uma máquina virtual Java e obedecendo a especificação JavaDTV [78]) e *Motor de Apresentação* que são responsáveis por executar aplicações procedurais e apresentar as declarativas, respectivamente.

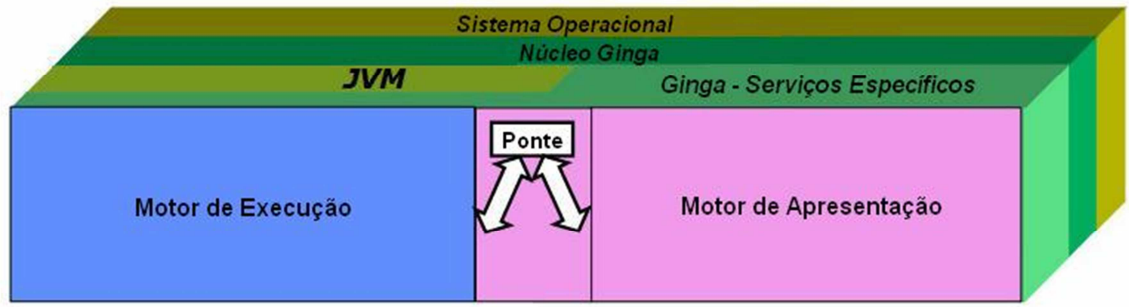

Figura 27: Arquitetura em alto nível do Ginga

Fonte [73]

Uma vez que uma aplicação de uma natureza pode utilizar recursos de aplicações de outra natureza, a *Bridge* é disponibiliza para viabilizar essa interface entre os dois componentes.

O *Núcleo do Ginga* tem, entre outras responsabilidades, a função de receber os conteúdos transportados por meio dos fluxos MPEG-2 e do canal de retorno.

O ambiente de execução disponibilizado pelo Ginga é apresentado na figura 28. Aplicações nativas hospedadas em um equipamento dotado do Ginga podem ser implementadas independentemente dos padrões definidos pelo *middleware*, porém podem fazer uso das APIs por ele disponibilizadas.

Uma aplicação enviada pela emissora via *broadcasting* (também conhecida como XLET) deve, necessariamente, ser executada utilizando a API disponibilizada, pois é isso que garante que todos os receptores, independente do sistema operacional e do *hardware* adotados pelo fabricante, serão capazes de executar a aplicação.

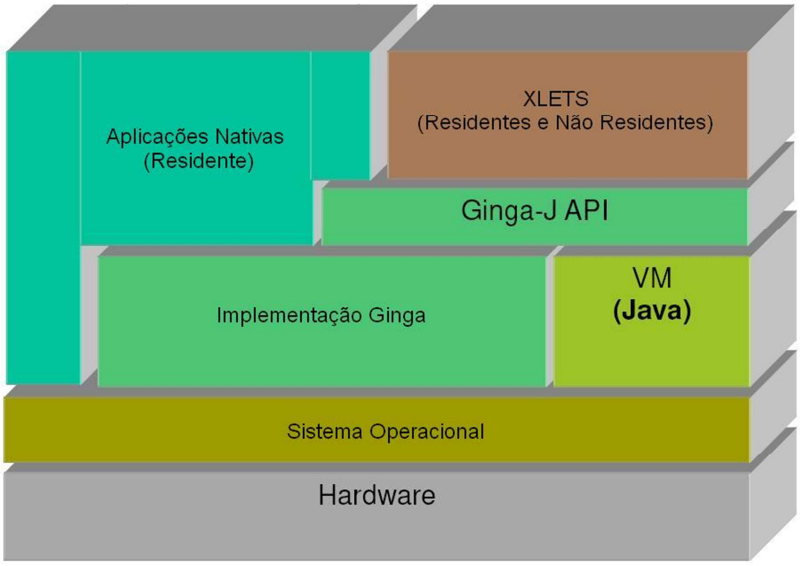

Figura 28: Arquitetura e ambiente de execução

Fonte [73]

Conforme citado anteriormente, alguns requisitos do Sistema Brasileiro de Televisão Digital não são atendidos pelos *middlewares* pré-existentes. Sendo assim, um conjunto de APIs foi criado para atender exclusivamente essas necessidades como, por exemplo, envio assíncrono de mensagens via canal de retorno e comunicação do receptor com outros equipamentos utilizando interfaces padrão como, por exemplo, *Bluetooth*.

Na implementação de referência do Ginga (que faz uso do modelo de componentes) existe um componente denominado *Interaction Channel* que oferece interfaces que permitem que os demais objetos da arquitetura se comuniquem com aplicações remotas utilizando um canal bidirecional de comunicação.

4.8 Interatividade

A televisão digital interativa permite ao usuário deixar de realizar um papel totalmente passivo, onde ele apenas assiste ao conteúdo de áudio e vídeo transmitidos

pela emissora, podendo realizar ações como, por exemplo, interagir com e emissora participando de uma votação por meio do controle remoto de sua TV, enviar e receber *e-mails*, acessar a *Web*, entre outros [60] [71].

Para que a interatividade seja possível, é preciso, entre outros, a presença de um gerador de carrossel que é responsável pelo envio (pela emissora) da aplicação a ser executada e a existência de um *Set-Top Box* capaz de executá-la [60]. Os sistemas de televisão digital podem ser divididos em função da forma de interatividade oferecida:

- *Sistemas pseudointerativos*: São aqueles onde a interatividade é limitada ao sistema interno do *Set-Top Box*. Dessa forma, o usuário pode receber uma aplicação da emissora (*via broadcasting*), interagir com essa aplicação, mas sem retornar informações à emissora.
- *Sistemas interativos*: Além das características existentes nos sistemas pseudointerativos, os sistemas interativos permitem que o usuário envie dados à emissora. Para isso, é necessária a existência de um canal de comunicação para o retorno de dados.

4.9 Considerações Finais

A implantação do Sistema Brasileiro de Televisão Digital irá prover diversas vantagens aos telespectadores como, por exemplo, melhor qualidade de áudio e vídeo, interatividade, entre outras.

Os *set-top boxes* apresentam algumas características similares àquelas encontradas em um computador convencional, permitindo a execução de aplicações.

É possível que os recursos computacionais existentes nos *set-top boxes* encontrem-se ociosos em determinados períodos. Sendo assim, torna-se interessante a possibilidade de compartilhar tais recursos para que possam ser empregados em tarefas que exijam uma grande potência computacional, caracterizando uma grade computacional *desktop* composta por *set-top boxes.*

Capítulo

5

Grid Anywhere

5.1 Considerações Iniciais

Inicialmente as grades computacionais tinham como foco as instituições que formavam organizações virtuais com o objetivo de compartilhar recursos entre elas. Nesse tipo de abordagem é comum o compartilhamento de ambientes computacionais de alto desempenho como *clusters* e supercomputadores, sendo que cada participante pode atuar como provedor ou consumidor de recursos.

Outra abordagem para esse paradigma permite que computadores pessoais espalhados por toda a Internet possam ser disponibilizados de forma voluntária para compor uma grade computacional *desktop*. Nesse modelo, os voluntários sempre atuam no papel de provedores de recursos.

Observando esses possíveis cenários, pode ser observado que as grades computacionais estão na grande maioria dos casos relacionadas ao processamento de alto desempenho, que normalmente é requisitado por grandes empresas e instituições de pesquisa.

Diversos *middlewares* já permitem que essas arquiteturas de grades computacionais sejam implementadas e utilizadas com sucesso. No entanto, extensões podem ser projetadas para permitir que novas aplicações e novos tipos de provedores e consumidores possam ser inseridos neste paradigma de computação distribuída.

Com o objetivo de permitir essas extensões, foi desenvolvido nesta tese de doutorado o *Grid Anywhere*, que se trata de uma especificação e um núcleo de um *middleware* para grades computacionais de processamento com módulos acopláveis.

O *Grid Anywhere* é totalmente baseado na migração (ou criação remota) de objetos, que podem ser invocados por meio do uso do protocolo SOAP. No entanto, uma especificação arquitetural permite que novas formas de transporte das mensagens SOAP sejam desenvolvidas e acopladas de maneira muito simples ao *middleware*, permitindo que as extensões citadas anteriormente possam ser realizadas.

Assim, nesta tese de doutorado são apresentados modelos, propostas e implementações que além de compor o núcleo do *middleware*, viabilizam a ampliação do paradigma, sendo apresentadas duas extensões. A primeira considera que as grades computacionais possam ser utilizadas para permitir que usuários convencionais (domésticos ou corporativos) possam fazer uso de recursos remotos ociosos existentes em computadores de outros usuários, da mesma categoria, para aumentar a sua potência computacional para execução de aplicativos.

Como exemplo disso, é possível imaginar um representante comercial que utiliza seu computador portátil para atender seus clientes. Se uma nova versão da aplicação exigir uma potência computacional maior que aquela provida por seu microcomputador, seria preciso adquirir um novo equipamento. Nesse tipo de situação, torna-se interessante que o computador do usuário possa ser utilizado apenas para executar a interface homem-máquina e que o restante seja executado remotamente, o que desonera o equipamento local de grandes demandas de processamento.

Quando se observa a oferta de plataforma como serviço (PaaS) do paradigma de computação em nuvem, é possível notar algo semelhante com esta abordagem. É possível que aplicações sejam executadas em provedores que ofertam seus recursos como serviços e depois cobram por aquilo que o usuário consumiu.

No entanto, muitos computadores pessoais estão ociosos e já são utilizados por aplicações científicas, como é o caso do SETI@Home. Por isso, torna-se bastante interessante permitir que esses computadores pessoais ociosos possam ser utilizados também para aplicações convencionais de maneira a prover plataforma como serviço de forma colaborativa.

Essa abordagem caracteriza uma sobreposição dos conceitos de computação em nuvem e computação em grade. A viabilidade dessa sobreposição foi demonstrada teoricamente por Ian Foster, como pode ser visto na figura 29.

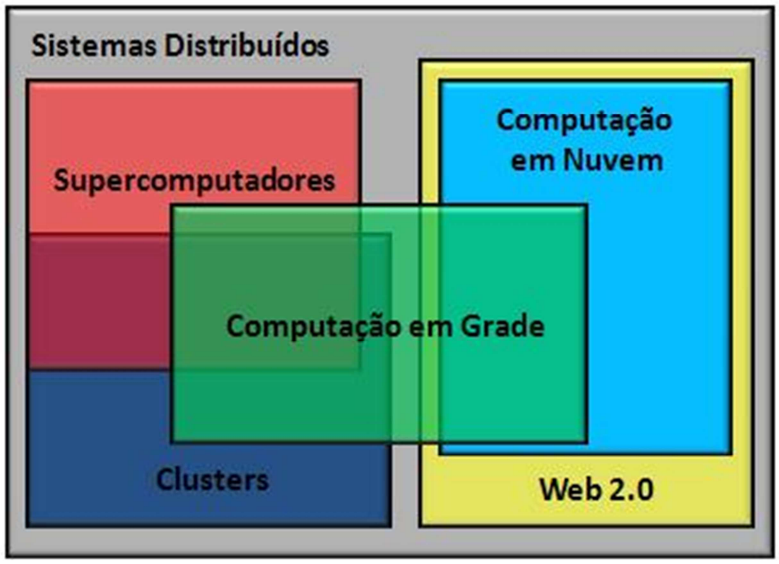

Figura 29: Relação entre sistemas distribuídos

Baseado em [15]

A segunda extensão tem como objetivo permitir que diferentes tipos de equipamentos pudessem compartilhar seus recursos para ampliar a potência computacional de grades para processamento paralelo. Para isso, o *Grid Anywhere* permite que objetos remotos sejam enviados por uma emissora de TV via difusão (*broadcasting*) e hospedados nos receptores de televisão digital. Para realizar a execução dos métodos, o *middleware* permite que as mensagens SOAP sejam também transportadas por meio de difusão, caracterizando um ambiente de processamento paralelo composto pelos receptores.

A criação dessas extensões do paradigma cria muitos novos problemas que precisam de soluções que são apresentadas nas próximas seções deste capítulo da tese.

5.2 Sam Dog: Ambiente de Execução Segura de Aplicações

O processo de compartilhamento de recursos em uma grade computacional exige um alto nível de segurança, tendo em vista que aplicações de terceiros são executadas no sistema local. Para isso, um provedor precisa ser capaz de determinar “o que” pode ser feito, por “quem” e “quando”.

As arquiteturas de grades computacionais podem apresentar diferentes requisitos no gerenciamento de segurança. Em uma organização virtual, por exemplo, o número de

consumidores é restrito aos participantes que a compõem. Já nas grades computacionais *desktop* o modelo mestre-trabalhador determina que as aplicações que são executadas em um voluntário são originadas sempre de um mesmo consumidor.

É possível que um provedor de recursos tenha necessidade de determinar políticas de segurança diferentes para grupos ou usuários distintos. Quando isso é observado, nota-se que as duas configurações de grades citadas no parágrafo anterior possuem uma quantidade de consumidores onde é viável esse tipo de gerenciamento.

No entanto, quando uma grade computacional é utilizada como infraestrutura de um ambiente para ofertar plataforma como serviço (PaaS) por meio do compartilhamento de recursos, o número de consumidores pode ser muito grande, tornando o gerenciamento mais complexo.

Uma alternativa simples para um cenário como esse seria possuir uma única configuração que determina de forma global o que poderia ser realizado por aplicativos de terceiros no sistema local. Mas, a possibilidade de diferenciação dos direitos dos usuários torna a grade computacional muito mais flexível e eficiente. Como exemplo, um usuário que participa de uma grade computacional permite que apenas um amigo estabeleça conexões de rede e escreva dados no sistema de arquivos, os demais consumidores da grade podem apenas utilizar um determinado percentual do processador.

Além das dificuldades de configuração de políticas de segurança em função do volume e da heterogeneidade dos consumidores, a pouca ou nenhuma experiência que o usuário pode possuir nesse tipo de atividade pode inviabilizar todo o processo, pois muitas são as atividades que uma aplicação externa pode executar no sistema local, sendo preciso abranger todas elas.

Diversas são as soluções para garantir a execução segura em grades computacionais *desktop*. O uso de virtualização é proposto para essa finalidade [79]. No entanto, quando equipamentos com uma menor capacidade computacional são utilizados para compor a grade (como os receptores de televisão digital, por exemplo), a execução da máquina virtual pode consumir uma quantidade de recursos que inviabilizaria a utilização desse tipo de dispositivo como provedor.

O uso de intercepção de chamadas de sistema também é uma alternativa [80]. Um módulo chamado SBLSM que é baseado no LSM (*Linux Security Module*) é

utilizado pelo *XtremWeb* para realizar esse gerenciamento. No entanto, isso sacrifica a portabilidade.

De forma parecida com o *XtremWeb*, o Entropia [81] também faz uso de intercepção de chamadas de sistema para gerenciar a execução de aplicações, mas suportando sistemas Windows. A mesma questão da interoperabilidade é encontrada.

Essas soluções apresentam bons resultados. No entanto, não são aplicáveis aos requisitos do *Grid Anywhere* em função da necessidade de portabilidade.

Uma solução chamada Jalapa [82] apresenta um mecanismo totalmente baseado em Java para a implementação de *sandboxing*. Ela se baseia na análise do histórico de chamada de métodos para definir um autômato que é utilizado para estabelecer as regras locais, onde é possível adotar sentenças lógicas utilizando, inclusive, os parâmetros das chamadas.

Essa solução é bastante interessante tendo em vista sua portabilidade. No entanto, na arquitetura do *Grid Anywhere*, a existência de um número grande de consumidores exige mecanismos mais flexíveis de gerenciamento. Além disso, se faz necessária a possibilidade de utilizar mais informações além dos parâmetros de chamadas de métodos para estabelecer regras mais abrangentes.

O *Sam Dog* é um ambiente seguro de execução de aplicações que tem como objetivo prover mecanismos de gerenciamento de políticas de segurança com portabilidade, flexibilidade e simplicidade, de maneira que seja viável abrigar o maior número possível de usuários em uma grade computacional, independentemente de sua familiaridade com a computação.

Em cenários como os propostos nesta tese de doutorado, é bastante interessante que os usuários possam delegar as funções de gerenciamento a entidades com maior nível de conhecimento no assunto. Mas, características pessoais podem requisitar que modificações sejam realizadas de maneira a refinar as regras para atender às peculiaridades locais.

Para viabilizar esses requisitos, o *Sam Dog* adota um modelo de gerenciamento de políticas em cascata. Nesse modelo as regras são armazenadas em diferentes camadas independentes que são sobrepostas de acordo com o perfil usuário. Ao determinar uma permissão sempre é observada a regra que está visível no momento.

Na figura 30 são demonstradas três políticas diferentes de gerenciamento. A política “A” define que todos os usuários podem gravar dados no disco e acessar a rede. Na política “B” as regras determinam que somente os usuários de uma determinada universidade podem ter acesso ao disco e que ninguém pode fazer acesso à rede. Por último, a política “C” é configurada para permitir que somente Pedro tenha acesso à rede, sem se preocupar com o acesso a disco.

Figura 30: Políticas de Segurança

Em função da forma de sobreposição das camadas são determinadas as regras de acesso ao recurso. Na figura 31 são exemplificadas essas sobreposições.

Figura 31: Sobreposições das Políticas de Segurança

Na primeira organização de políticas, todas as regras definidas em “A” foram sobrepostas por “B”, prevalecendo essas últimas. O segundo exemplo faz referência a um caso onde um usuário deseja utilizar todas as regras estipuladas por “A” com exceção do gerenciamento de rede. Por fim, a última configuração demonstra uma situação onde o usuário deseja fazer uso das regras impostas por “A”, mas aplicando as modificações feitas tanto em “B” quanto em “C”.

O *Sam Dog* considera a existência de uma organização hierárquica onde cada elemento possui seu próprio conjunto de regras (camada). A sobreposição dessas regras é feita automaticamente, sendo que as camadas de nível inferior herdam as configurações do nível superior que são sobrepostas por aquelas definidas no nível atual.

Cada nível pode ser composto por um gerenciador de segurança de domínio, um grupo ou uma entidade, que são detalhadas a seguir.

5.2.1 Gerenciador de Segurança de Domínio

O *Grid Anywhere* considera que usuários podem não ser capazes de definir suas próprias configurações de segurança. Mas, eles podem possuir instituições confiáveis de onde essas configurações podem ser herdadas.

Os Gerenciadores de Segurança de Domínio (GSD) são responsáveis por configurar e disponibilizar políticas de segurança aos usuários que confiam nessa instituição. Eles podem ser compostos por universidades, empresas, emissoras de televisão, entre outros.

Esses gerenciadores são organizados de maneira hierárquica, o que permite que as regras sejam herdadas. Sendo assim, quando um novo domínio de segurança é criado o administrador pode escolher um gerenciador já existente que possui uma política de segurança semelhante àquela que ele deseja empregar para se subordinar. Após isso, apenas as exceções precisam ser configuradas no domínio local.

Um exemplo de composição de gerenciadores de domínio é apresentado na figura 32.

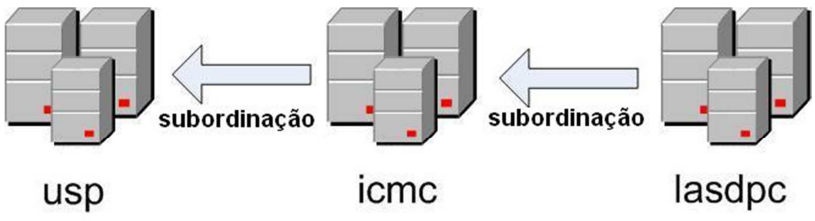

Figura 32: Exemplos de domínios de segurança

Nesta configuração o domínio USP possui um conjunto de regras que é herdado por ICMC que, por sua vez, cria suas próprias regras para sobrepor àquelas herdadas. O

mesmo acontece com LASDPC, sendo que este último herda as políticas dos dois primeiros.

É feita a utilização de um espaço de nomes para denotar a hierarquia dos gerenciadores de domínio. Sendo assim, cada gerenciador possui como identificador sua composição genealógica completa separada por pontos. Portanto, os exemplos da figura 32 teriam, respectivamente, os identificadores: *usp, usp.icmc* e *usp.icmc.lasdpc*.

Os gerenciadores de segurança e o modelo de políticas em cascata são os elementos chave para viabilizar o controle de execução em uma grade computacional composta por muitos participantes que compartilham recursos entre si. Os GSDs permitem que usuários leigos em computação tenham um ambiente de execução seguro por meio da utilização de políticas pré-estabelecidas e o modelo em cascata fornece os mecanismos necessários para que aqueles participantes com maior nível de conhecimento possam personalizar suas configurações.

5.2.2 Entidades e Grupos

Conforme citado anteriormente, o processo de segurança em uma grade computacional necessita saber “o que” pode ser feito e por “quem”. As entidades são responsáveis por determinar o “quem” desse processo.

Um domínio é gerenciado por um Gerenciador de Segurança de Domínio e pode possuir diversos subdomínios, grupos e entidades. Uma entidade é tudo aquilo que possa atuar no papel de consumidor de recursos (um computador, *tablet*, *smartphone* ou um usuário propriamente dito) e um grupo é um conjunto de entidades, sendo ambos subordinados a um GSD.

Um grupo sempre é associado a um domínio e uma entidade pode ser associada diretamente ao domínio ou a um grupo. A identificação dos grupos e entidades é realizada de maneira semelhante àquela utilizada nos domínios.

Se considerar uma estrutura de domínios, grupos e usuários conforme a figura 33, os identificadores das entidades seriam: *usp.icmc.lasdpc.alunos.joao* e *usp.icmc.lasdpc.professores.daniel*.

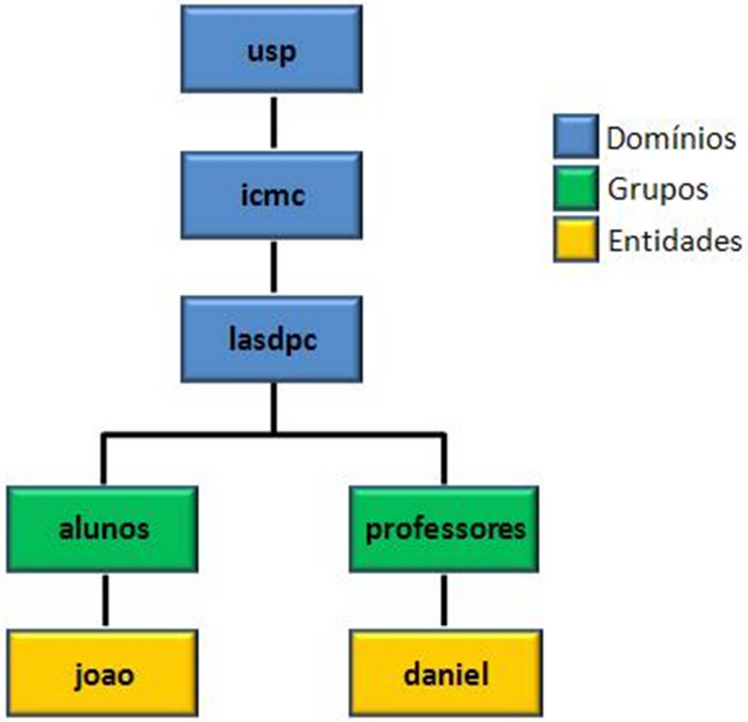

Figura 33: Exemplo de composição de entidades, grupos e de domínios

Com o objetivo de eliminar problemas de ambiguidade nos processos existentes, não é permitido que um gerenciador de domínio de segurança possua mais de um subordinado direto com o mesmo nome.

Embora não exista a figura explícita de uma organização virtual no *Grid Anywhere*, só é possível realizar o compartilhamento de recursos entre entidades que são conexas considerando a árvore formada pela hierarquia dos GSDs, grupos e entidades. Esse requisito é estabelecido em função das questões de autenticação que são apresentadas na próxima subseção.

A organização em árvore dos GSDs, grupos e entidades são um ponto fundamental para a flexibilidade do mecanismo de gerenciamento de políticas de segurança. Permissões e restrições de acesso podem ser inseridas em qualquer ponto dessa árvore, sendo que uma entidade sempre adota aquela mais próxima de si.

Utilizando como exemplo a árvore apresentada na figura 33, é possível que um administrador queira que somente o *Prof. Daniel* tenha permissão para um determinado acesso. Para que não seja necessário realizar configurações em todos os outros usuários,

basta que uma restrição seja atribuída para o domínio *LASDPC* e uma permissão seja inserida diretamente para a entidade *Daniel*.

5.2.3 Certificação Digital e Autenticação

Todo o processo de autenticação utilizado pelo *Sam Dog* é baseado no uso de chaves assimétricas. Para isso, cada gerenciador de segurança de domínio emite certificados para seus subdomínios e entidades.

Cada participante possui um repositório com os certificados de GSDs válidos processados anteriormente. Esse repositório deve possuir no mínimo os certificados de todos os gerenciadores de domínios aos quais o participante é subordinado.

Quando uma nova autenticação se faz necessária e não existe o certificado para aquela entidade no repositório, é requisitado a ela seu certificado. Ao receber esse certificado é feita uma validação de caminho (*validation path*) considerando apenas os certificados emitidos por GSDs. Essa validação ocorre até que um GSD comum ao requisitante e ao requisitado seja encontrado.

No entanto, o identificador não possui informações suficientes para determinar o que são grupos e o que são GSDs, devido ao fato da possibilidade de existência de subgrupos. Para resolver isso, o algoritmo apresentado a seguir realiza a validação dos certificados, processa o identificador do consumidor para verificar os GSDs comuns entre eles, solicitar os certificados dos demais GSDs e realizar a validação propriamente dita. São as primitivas desse algoritmo:

- *Rn*: identificador do requisitante.
- *Rd*: identificador do requisitado.
- *Rn(x)*: identificador consistindo do nível 0 ao *x* de *Rn.*
- *Rd(x)*: identificador consistindo do nível 0 ao *x* de *Rd.*
- *S(x)*: Número de níveis existentes no identificador *x.*
- *Cd(x)*: Certificado digital do identificado *x.*
- *Pk(x)*: Chave pública do identificador *x.*
- *Endereco(x,y)*: Solicita ao GSD *x* o endereço do GSD *y*.
- *BuscarCertificado(x)*: Realiza a busca do certificado no endereço *x.*

- *Repositorio(x)*: Verifica se o GSD identificado por *x* possui certificado no repositório
- *Validacao(x,y)*: Realiza a validação do certificado x com a chave pública *y*.
- *Remover(x)*: Remove o certificado *x* do repositório
- *ValidaEntidade(x,y)*: Valida a entidade *x* com a chave pública *y*.

Algoritmo 1: Identificação e validação dos certificados

```
//Identifica o GSD comum de maior nível (E)
E=-1
N=0
Enquanto N < S(Rn), faça:
  Se Rd(N) é GSD E (N=0 OU Endereco(Rn(N-1),Rn(N))!=null) E Rn(N)=Rd(N), então
    E=N
  Senão:
    N=S(Rn)
  FimSe
  N++
FimEnquanto
Se E=-1, então: //Não há GSD comum
  Rejeitar
FimSe
//Determina o ultimo GSD existente no identificador (UGSD) e
//carrega os certificados dos GSDs não comuns e os armazena
UGSD=E
N=E+1
Enquanto N < S(Rn), faça:
  UGSD=N
  Se !Repositorio(Rn(N)), então:  //Não existe no repositório
    End=Endereco(Rn(N-1),Rn(N))
    Se End=Null, então: //Não é um GSD
      UGSD=N-1
      N=S(Rn)
    Senão:
      BuscarCertificado(End)
    FimSe
  FimSe
  N=N+1
FimEnquanto
// Validação dos certificados. Do ultimo GSD até o primeiro comum
Correto=true
N=UGSD
Enquanto N > E, faça:
  Se !Validacao(Cd(Rn(N)),Pk(Rn(N-1))), então:
    Correto=false
    Remover(Cd(Rn(N))
```

```
    N=0
  FimSe
  N=N-1
FimEnquanto
Se !ValidaEntidade(Rn, Pk(Rn(UGSD))), então:
  //GSD não reconhece a entidade requisitante
  Rejeitar
FimSe
```

5.2.4 Tarefas

As tarefas são responsáveis por determinar “o que” pode ser feito por uma entidade, grupo de entidades ou um domínio de segurança. É comum que em ambientes mais complexos uma grande quantidade de possíveis tarefas seja encontrada.

Semelhantemente com as entidades, o grande volume de tarefas a serem administradas (permitidas ou negadas) pode dificultar o processo de gestão de segurança do usuário provedor. Por isso, as tarefas também são organizadas de forma hierárquica para flexibilizar o gerenciamento.

As permissões e restrições podem ser aplicadas em qualquer ponto da árvore, sempre sendo observada aquela mais próxima do nó utilizado na requisição (que sempre é uma “folha” da árvore). Na figura 34 é apresentado um exemplo de organização das tarefas. Nesse caso, com exceção das requisições de conexões de rede, todas as operações de IO são bloqueadas.

Cada tarefa existente tem um identificador que é composto por todo o caminho entre o nó raiz da árvore até a tarefa propriamente dita. Utilizando o exemplo da figura 34, a tarefa “Aceitar Conexões” é identificada por *tarefas.io.network.aceitarconexoes*.

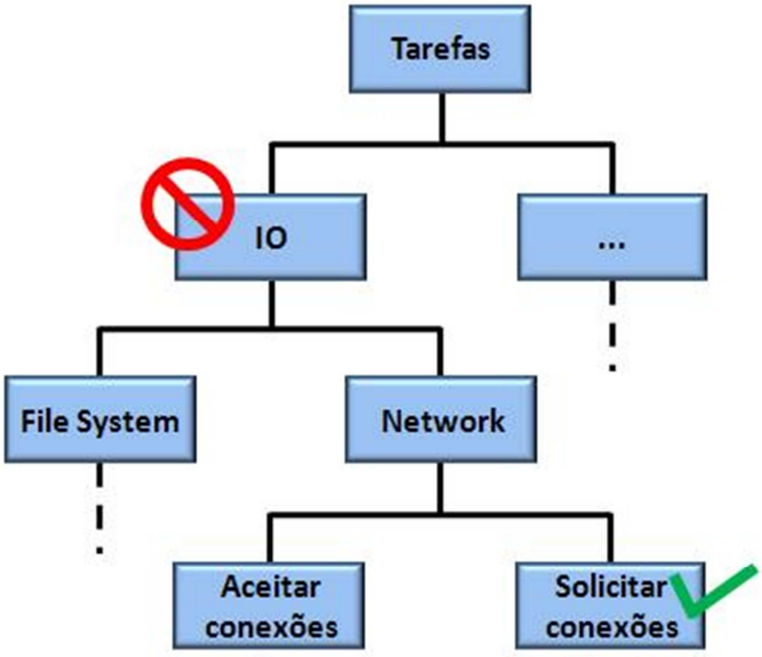

Figura 34: Organização em árvore das tarefas

5.2.5 Composição de Tarefas e Entidades

Até este ponto foram discutidas separadamente as entidades e as tarefas que, respectivamente, determinam “quem” e “o que”. No entanto, elas precisam ser compostas para que seja possível realizar a sentença que determina “quem pode fazer o que”. Para isso, é realizada uma composição entre as árvores de entidades e de tarefas.

A composição permite que qualquer nível da árvore de tarefas seja utilizado para determinar a regra de utilização para qualquer nível da hierarquia de entidades. Sendo assim, para árvores com *T* tarefas e *E* entidades, é possível estabelecer *T*E* diferentes regras.

Cada regra é composta por uma tripla que é responsável por determinar a composição entre tarefas e entidades e a ação determinada por esta regra. Sendo assim, a notação abaixo é utilizada para formalizar essa tripla:

regra(ação, entidade, tarefa)

A ação consiste em dois possíveis valores: aceitar ou rejeitar. Utilizando as entidades da figura 33 e as tarefas da figura 34 é possível descrever um cenário fictício de uso da composição: *Todos os usuários do domínio LASDPC podem realizar*

qualquer tipo de operação de IO, com exceção do aluno João que não pode estabelecer conexões de rede.

Para isso, considerando o modelo de composição das árvores, apenas duas regras são necessárias:

a) *regra(aceitar,usp.icmc.lasdpc,tarefas.io);*

b) *regra(rejeitar,usp.icmc.lasdpc.alunos.joao,tarefas.io.network.solicitarconexoes);*

É possível observar que a regra “A” foi responsável por determinar o cenário genérico e a regra “B” a exceção. Esse tipo de especificação facilita o gerenciamento de cenários complexos que necessitem de uma política mais flexível de estabelecimento de regras.

O motor de processamento de regras segue um algoritmo que viabiliza que a composição das árvores permita configurações específicas que são sobrepostas às genéricas. Para isso, é verificada a existência de regras para cada possível composição das árvores.

Quando uma requisição de acesso é realizada são informados os identificadores da entidade requisitante e da tarefa requisitada (ambos serão sempre folhas das respectivas árvores). Partindo desses identificadores o algoritmo inicia o processamento de todos os níveis de entidades e tarefas, partindo da mais específica e realizando generalizações à medida que nenhuma regra é encontrada.

Conforme pode ser visto pelo algoritmo a seguir, onde:

- *E*: Identificador da entidade requisitante.
- *T*: Identificador da entidade requisitada.
- *I(x,y)*: Trecho do identificador *x* partindo do nível 0 até o nível *y*.
- *ProcureRegra(x,y)*: Realiza a procura por uma regra estabelecida para os níveis de entidade e tarefas identificadas respectivamente por *x* e *y*.

Algoritmo 2: Composição de tarefas e entidades

```
a=S(E)-1
Enquanto a >= 0, faça:
  b=S(T)-1
  Enquanto b >= 0, faça:
    ProcureRegra(I(E,a), I(T,b))
    Se a regra for encontrada, então:
      Execute a ação especificada
      Fim
    FimSe
    b=b-1
  FimEnquanto
  a=a-1
FimEnquanto
```

Assim, para cada possível nível das entidades são testados todos os níveis de tarefas. Isso garante que sempre será processada a regra mais próxima (específica) à entidade requisitante.

5.2.6 Utilização de Contexto para Definição de Regras

Dessa forma, já foram discutidas as entidades e as tarefas que determinam, respectivamente, “quem” pode fazer “o que”. No entanto, é possível que as requisições de tarefas possam ser liberadas ou bloqueadas para uma determinada entidade em função do contexto em que a requisição está sendo realizada. Sendo assim, passa-se a determinar “quem” pode fazer “o que” e “quando”.

Para isso, no momento da requisição de uma tarefa é fornecido um conjunto de variáveis que determina o contexto no qual a execução está ocorrendo. Em função desse contexto são estabelecidas as regras para determinar se a requisição deve ser aceita ou rejeitada.

A definição da regra utiliza o formato padrão para imposição de uma condição: *IF/ELSE*. Portanto, o administrador cria uma regra por meio de uma proposição que faz uso de operadores relacionais e lógicos para determinar a ação a ser tomada em função das variáveis de contexto.

5.2.7 Sobreposição dos Conjuntos de Regras

A possibilidade de realizar a herança de regras estabelecidas por um GSD é um ponto chave para permitir que usuários não experientes possam compor a grade computacional aqui proposta.

O algoritmo apresentado na subseção 5.2.5 tem como objetivo a composição das árvores de entidades e tarefas. Esse algoritmo seria suficiente para um conjunto de regras único, mas não permite a sobreposição de políticas se segurança.

Já foi discutido que diferentes políticas se segurança são definidas em cada GSD e também no próprio provedor de recursos. Tais políticas são herdadas, partindo do GSD raiz e seguindo até o provedor. Para determinar esse comportamento é necessária uma extensão do algoritmo de busca de regras, resultado no algoritmo apresentado a seguir, onde:

- *G*: Vetor contendo as políticas de segurança de todos os GSD que o provedor é subordinado. O GSD raiz tem sua política armazenada na posição 0.
- *ProcureRegraGSD(x,y,z)*: Realiza a procura, no GSD *x*, por uma regra estabelecida para os níveis de entidade e tarefas identificadas respectivamente por *y* e *z*.

Algoritmo 3: Composição de GSDs, entidades e tarefas

```
a=S(G)-1
Enquanto a>=0, faça:
  b=S(E)-1
  Enquanto b>=0, faça:
    c=S(T)-1
    Enquanto c>=0, faça:
      ProcureRegraGSD(G[a], I(E,b), I(T,c))
      Se a regra for encontrada, então:
        Execute a ação especificada
        Fim
      FimSe
      c=c-1
    FimEnquanto
    b=b-1
  FimEnquanto
  a=a-1
FimEnquanto
```

Esse algoritmo é o responsável pela característica de aplicação de regras em cascata. Como pode ser observado, são sempre processadas primeiramente as regras que estão localizadas localmente no provedor e aquelas dos GSDs mais próximos. Sendo assim, a herança de regras ocorre somente quando não há definições mais específicas.

A função de complexidade desse algoritmo é *O(S(G)*S(E)*S(T))*. Para as situações onde as regras estão definidas nos níveis mais altos das árvores de GSDs, entidades e tarefas, o algoritmo precisa percorrer toda a estrutura de dados.

É possível que uma mesma requisição seja realizada por um mesmo aplicativo inúmeras vezes. Se em cada requisição a pesquisa for realizada completamente o tempo de busca da regra pode prejudicar o desempenho da aplicação. Para isso, é utilizada uma *cache* que é responsável por armazenar as consultas já realizadas. Sendo assim, quando uma nova busca de regra é feita, antes de percorrer a árvore de composição de GSDs x Entidades x Tarefas uma checagem é feita para tentar localizá-la na *cache*.

5.2.8 Implementação

O modelo proposto nessa seção foi totalmente implementado, utilizando a plataforma Java, em dois módulos independentes:

- *Sam Manager*: Permite ao usuário o gerenciamento das políticas.
- *Sam Controller*: Ambiente seguro de execução de aplicações Java que faz uso das políticas estabelecidas pelo módulo gerenciador.

Essa independência entre os módulos foi realizada porque a forma de gerenciamento das políticas de segurança seguindo o modelo proposto pode ser aplicada em diversas outras finalidades além do ambiente de execução de aplicações (*sandbox*). Como exemplo, o *Sam Dog* pode ser responsável pelo gerenciamento de permissões de acesso de usuários em um sistema ERP (*Enterprise Resource Planning*).

Para isso, são ofertadas APIs que provêm ao desenvolvedor total transparência dos recursos de cascateamento de políticas de segurança, processamento de regras, entre outros.

5.2.8.1 ***Implementação e Comunicação com GSDs***

Os Gerenciadores de Segurança de Domínio são disponibilizados utilizando SOA. Sendo assim, a utilização de serviços *Web* viabiliza a comunicação entre provedores e GSDs.

No momento da inicialização de um provedor de recursos é feita a recuperação das políticas de segurança que deverão ser sobrepostas pelo algoritmo apresentado anteriormente. Para realizar a busca dessas políticas o provedor não precisa ter os endereços de todos os GSDs, mas somente do seu superior direto. Por isso, cada GSD existente na hierarquia é responsável por fornecer o endereço do GSD superior para que a construção possa continuar até encontrar o gerenciador raiz.

A comunicação entre provedor de serviços e gerenciadores ocorre somente uma vez. Dessa forma, o tempo de construção das políticas não influencia o desempenho das operações requisitadas pela aplicação do consumidor de recursos.

5.2.8.2 ***Pacote de Tarefas***

O gerenciador de políticas de segurança permite que o *Sam Dog* seja utilizado para diversas aplicações que necessitem realizar controle de acesso a recursos. Aplicações distintas normalmente apresentam tarefas diferentes.

Um pacote permite ao usuário descrever, por meio de um arquivo XML, uma hierarquia de tarefas que serão gerenciadas pelo *Sam Dog*. Uma vez que este arquivo é inserido na pasta *tasks* do diretório de instalação do *Sam Dog*, automaticamente elas passam a compor a hierarquia principal que é gerada por meio de todas as hierarquias encontradas nos arquivos existentes nesta pasta.

Como exemplo, a estrutura de tarefas apresentada na figura 34 seria composta pelo documento XML apresentado a seguir.

É importante observar que a tarefa *aceitarconexoes* possui três variáveis de contexto: endereço IP de origem, porta de destino da conexão e hora atual. Essas variáveis devem ser acomodadas, pelo sistema gerenciado, em uma tabela *hash*, onde o nome da variável é a chave e seu conteúdo o valor de uma entrada tabela.

Listagem 1: Documento de definição de pacote de tarefas

```
<package>
  <id>tarefas</id>
  <version>1.0</version>
  <groups>
    <group>
      <id>io</id>
    </group>
    <group>
      <id>filesystem</id>
      <superGroup>io</superGroup>
    </group>
    <group>
      <id>network</id>
      <superGroup>io</superGroup>
    </group>
  </groups>
  <tasks>
    <task>
      <id>aceitarconexoes</id>
      <group>network</group>
      <context>
        <variables>
          <variable>
            <name>@ipOrigem</name>
            <type>String</type>
          </variable>
          <variable>
            <name>@porta</name>
            <type>int</type>
          </variable>
          <variable>
            <name>@hora</name>
            <type>int</type>
          </variable>
        </variables>
      </context>
    </task>
  </tasks>
</package>
```

*5.2.8.3 **Descrição das Regras***

As regras existentes em uma política de segurança também são descritas por um documento XML. Nesse documento são descritas as proposições que são analisadas pelo motor de processamento do *Sam Dog* para determinar a ação a ser tomada.

A primeira versão da implementação do Sam Dog dá suporte à utilização de variáveis e constantes na construção de uma proposição. As variáveis são sempre precedidas pelo símbolo @ (arroba) e as constantes de texto são informadas entre aspas. Constantes numéricas são escritas normalmente.

Para a construção da proposição são adotados operadores relacionais (=, !=, <, <=, > e >=) e operadores lógicos (*and/or*). Além disso, é permitida a utilização de parênteses para determinar a precedência lógica da operação.

O documento a seguir ilustra o estabelecimento de uma regra que permite que aplicações executadas em um ambiente de execução gerenciado pelo *Sam Dog* aceite conexões somente no período da madrugada na porta padrão para HTTP requisitadas pelo professor Daniel.

Listagem 2: Documento de representação de regras

```
<sam>
  <rule>
    <entity>usp.icmc.lasdpc.professores.daniel</entity>
    <task>tarefas.io.network.aceitarconexoes</task>
    <if>
      <expression>
        @porta=8080 and @hora>=0 and @hora<7)
      </expression>
      <then>
        <command>accept()</command>
      </then>
      <else>
        <command>reject()</command>
      </else>
    </if>
  </rule>
</sam>
```

De maneira a simplificar o processo de definição de regras, uma interface gráfica foi desenvolvida para permitir que a construção de políticas possa ser feita de forma intuitiva. A figura 35 é utilizada para demonstrar essa interface.

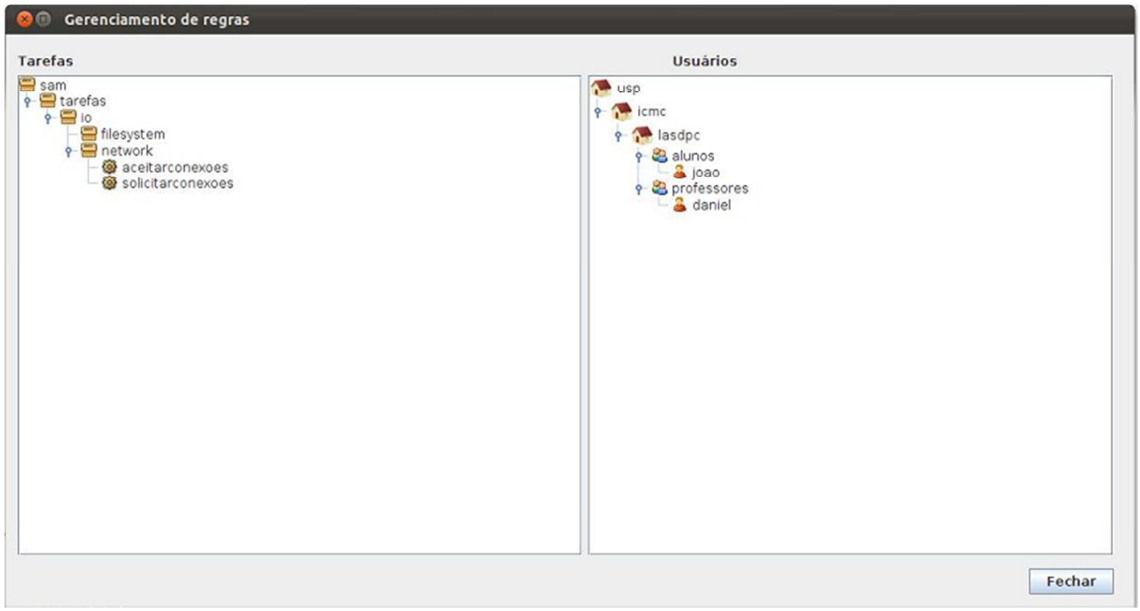

Figura 35: Interface de navegação

É possível observar que as hierarquias de GSDs, entidades e grupos são construídas automaticamente e demonstradas no lado direto da interface, enquanto as tarefas são disponibilizadas no lado esquerdo.

Para definir uma nova regra, basta o usuário selecionar os níveis de tarefas e entidades desejadas e solicitar uma nova regra, que tem sua composição realizada na interface vista na figura 36.

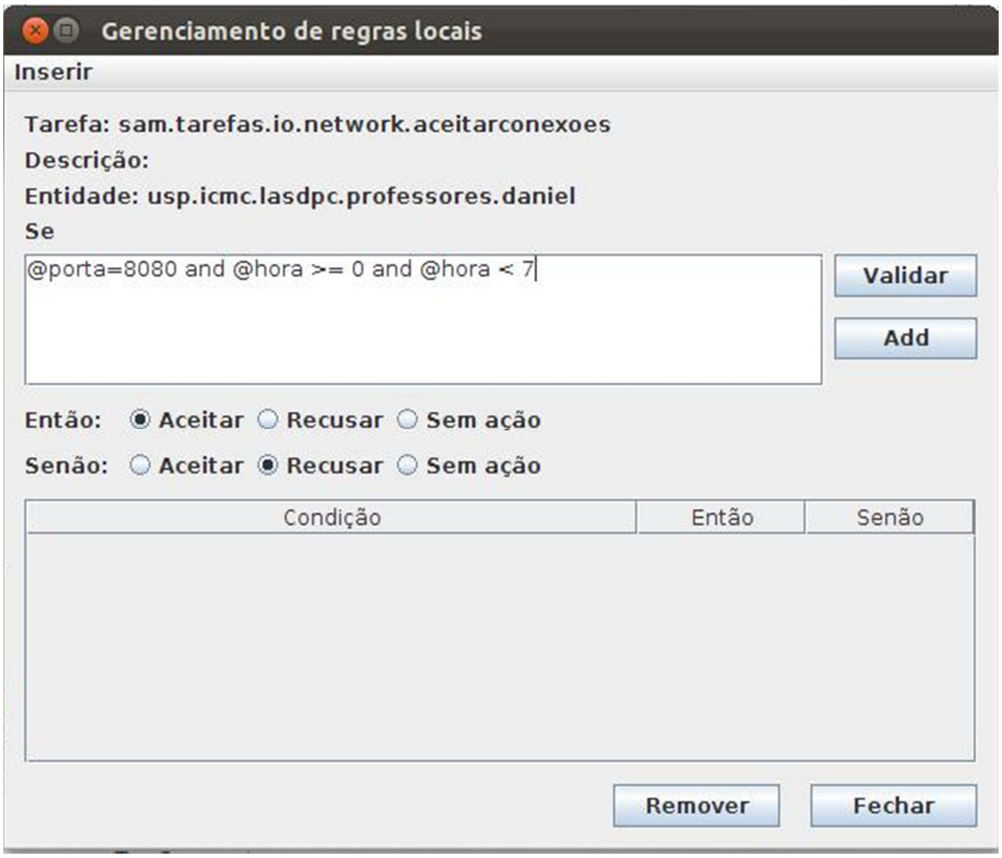

Figura 36: Interface de definição de regras

*5.2.8.4 **Sandbox para Execução de Aplicações Java***

A plataforma Java tem um *sandbox* que é responsável pelo gerenciamento da execução de aplicações na máquina virtual [83]. Esse ambiente de execução possui, conforme ilustrado na figura 37, um componente chamado *SecurityManager* que tem a função de interceptar as operações realizadas pela aplicação.

Para determinar se a requisição deve ser aceita ou rejeitada (quando uma exceção é lançada), o *SecurityManager* realiza uma consulta junto ao *AccessController* que, em função da política de segurança estabelecida, determina a ação a ser tomada.

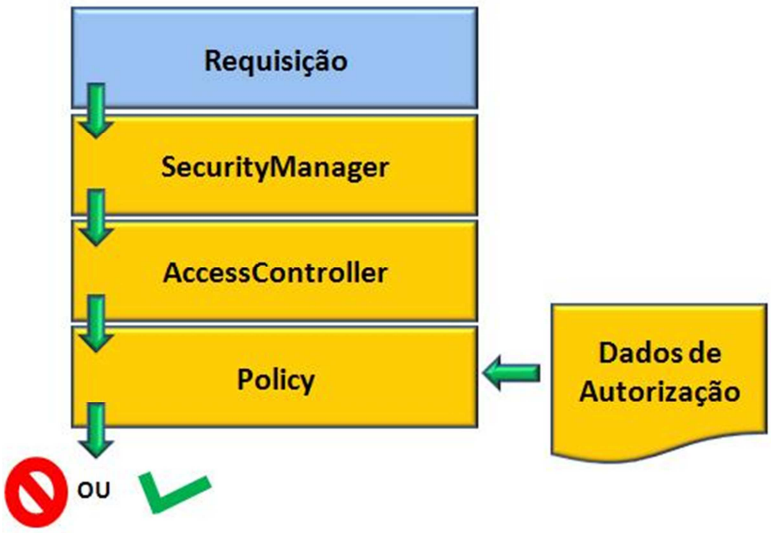

Figura 37: Controle padrão de recursos da plataforma Java

Traduzido de [83]

Na implementação padrão da plataforma Java cada regra é descrita por meio da associação de um tipo de operação à ação a ser tomada. Esse tipo de abordagem se apresenta pouco flexível quando um ambiente de grade computacional com muitos consumidores é utilizado.

Para permitir um melhor gerenciamento dos recursos compartilhados, o *Sam Dog* é utilizado para substituir esses componentes. Conforme demonstrado na figura 38, uma sobrescrita do *SecurityManager* é utilizada para enviar as requisições para o *SamController* que realiza a consulta junto ao *SamManager* que implementa todo o modelo de gerenciamento de pesquisa de regras demonstrados anteriormente.

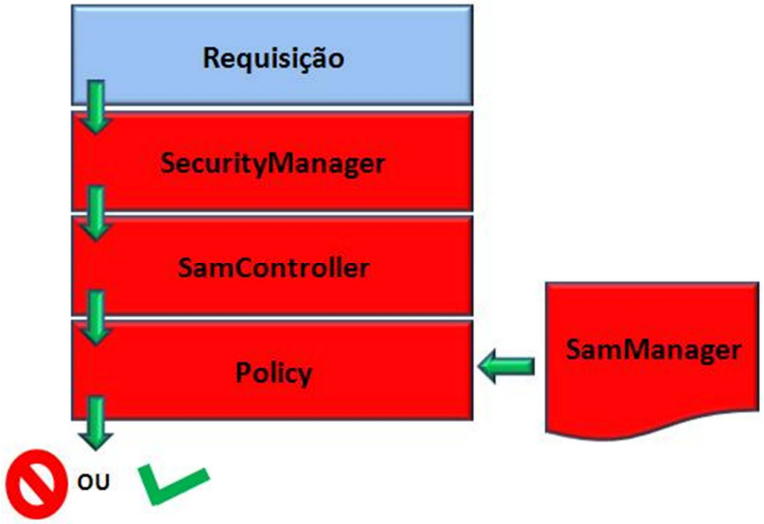

Figura 38: Controle de recursos realizado pelo Sam Dog

*5.2.8.5 **Avaliação de Desempenho***

De maneira a determinar o tempo adicional que será inserido pela utilização do *Sam Dog* em uma aplicação, foi realizada uma avaliação de desempenho da implementação citada anteriormente.

Conforme apresentado pela tabela 4, o planejamento do experimento foi realizado utilizando 5 fatores contendo 2 níveis cada. É importante observar que a palavra “nível” é apresentada nesta subseção em dois contextos: a) níveis das árvores de GSDs, entidades e tarefas; b) níveis de configuração dos fatores.

Tabela 4: Planejamento de Experimento

Fator	**Níveis**
Número de GSDs	2 e 3
Número de níveis na identificação de identidade da regra	5 e 10
Número de níveis na identificação de tarefa da regra	5 e 10
Número de regras existentes na política de segurança	5 e 10
Taxa de acerto de *cache* (%)	30 e 50

O ambiente de execução do experimento foi composto por 4 computadores. Três deles utilizados como gerenciadores de domínio de segurança (organizados em uma hierarquia) e um computador onde o ambiente de execução controlado pelo *Sam Dog* foi executado.

Uma vez que a comunicação com os gerenciadores é estabelecida somente no momento da inicialização do provedor, o custo de rede não foi considerado. Sendo assim, ao realizar os experimentos todas as camadas de políticas de segurança já estavam localizadas no provedor de recursos com as árvores montadas.

Como o objetivo principal da avaliação é determinar o tempo gasto para realizar a busca pelas regras, todas as consultas foram estipuladas utilizando-se identificadores de entidades e tarefas com 10 níveis. No entanto, os possíveis cenários foram criados inserindo as regras nos níveis 5 e 10 para ambos os identificadores.

A localização da política também foi considerada, realizando experimentos com as regras estabelecidas nas camadas dos gerenciadores de nível 2 e 3.

É possível que diversas regras sejam encontradas para um mesmo nível. Sendo assim, foram testadas situações com 5 e 10 regras para determinar a influência desse fator no tempo total. Por último, foi considerada a taxa de acertos de buscas na *cache* de regras.

A execução do planejamento foi realizada considerando um fatorial completo [84]. Sendo assim, todas as possíveis combinações dos fatores foram mensuradas utilizando como amostra um número de 100 execuções e um intervalo de confiança de 95%.

Os gráficos das figuras 39 e 40 são utilizados, respectivamente, para demonstrar os tempos de execução em experimentos com, respectivamente, 30 e 50 por cento de acerto de busca na *cache* de regras.

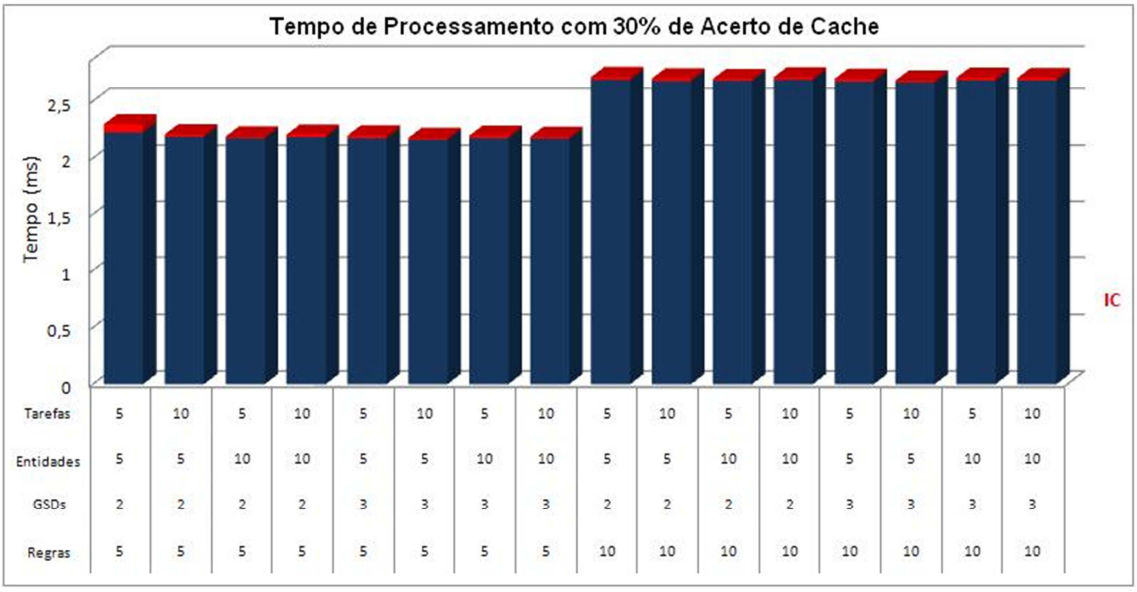

Figura 39: Tempos de processamento do com taxa de acerto de 30%

Como era esperado, o melhor caso, para uma *cache* com 30% de acerto, ocorreu quando as regras foram inseridas em um conjunto menor (5 regras) localizado no gerenciador mais próximo do provedor (nível 3) e utilizando identificadores de entidades e tarefas mais específicos possíveis (nível 10). Neste caso de teste o tempo médio gasto foi de 2,179 milissegundos.

O pior caso, também esperado, ocorreu quando as regras foram encontradas em um grupo maior (com 10 elementos) e em políticas mais genéricas, localizadas em um GSD mais distante e com identificadores de domínios e tarefas utilizando 5 níveis. Para essa busca o tempo médio foi de 2,705 milissegundos.

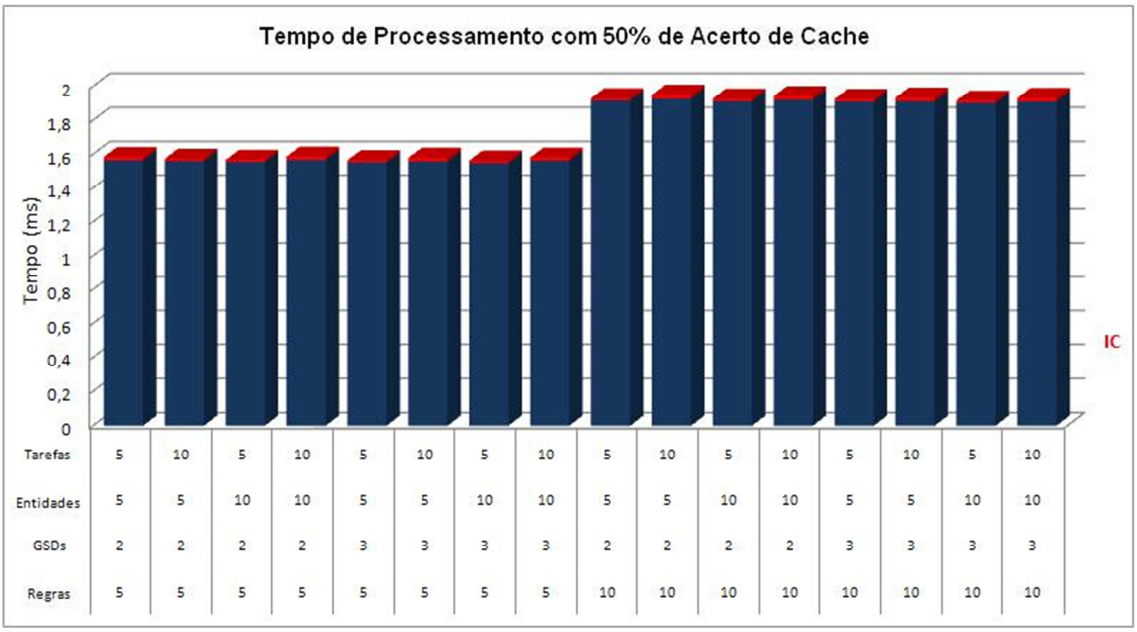

Figura 40: Tempos de processamento do com taxa de acerto de 50%

A utilização de uma *cache* com maior índice de acertos diminuiu o tempo em todos os cenários. O melhor caso teve o tempo diminuído para 1,571 milissegundos enquanto o pior caso gastou em média 1,917 milissegundos.

Conforme pode ser visto no gráfico da figura 41, o fator que mais apresenta influência no desempenho é a taxa de acerto da *cache* (mais de 97%) seguido pelo número de regras em uma mesma política (1,018%). A combinação de ambos forma a terceira maior influência.

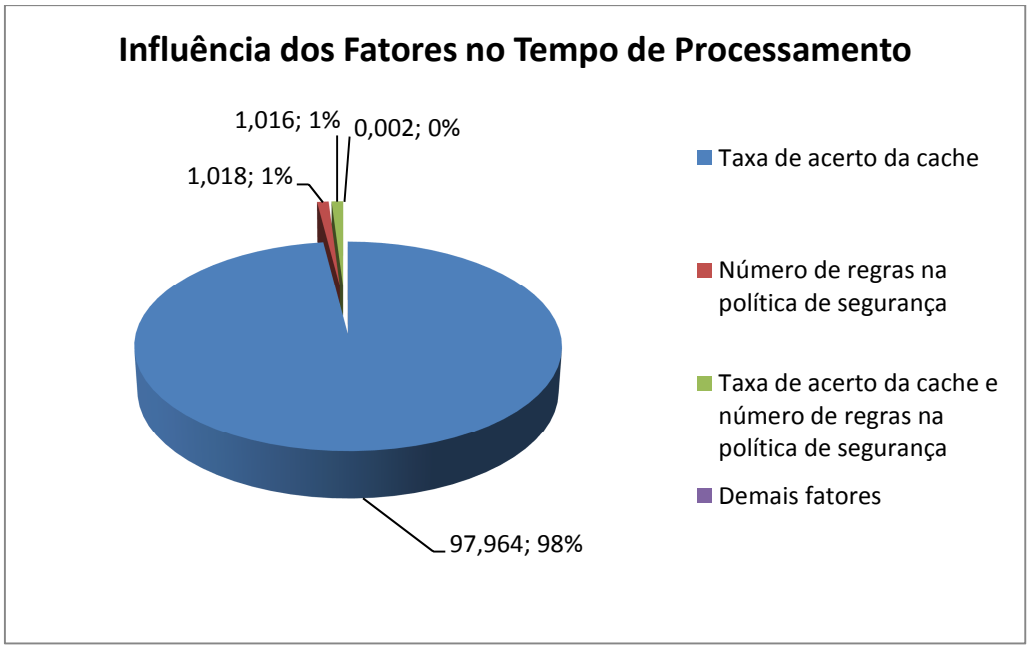

Figura 41: Influência dos fatores

5.3 WSBCL: Web Services Based Classloader

Quando um sistema distribuído é construído, é comum a utilização de componentes remotos que são invocados por meio da rede de computadores. Diversas são as tecnologias que permitem a implementação de aplicações como essa, destacando entre elas: RPC (*Remote Procedure Call*), RMI (*Remote Method Invocation*), CORBA (*Common Object Request BrokerArchitecture*) e serviços *Web*.

Para que uma aplicação possa ser executada remotamente é preciso que seu código executável esteja, de alguma forma, disponível no sistema. É comum, em sistemas com arquitetura estática, que a aplicação servidora seja instalada (ou implantada) no sistema de arquivos do equipamento que a hospedará.

Na plataforma Java, os *bytecodes* (que formam o código executável de uma aplicação) são carregados pela máquina virtual sob demanda. Sendo assim, ao executar uma aplicação a classe que contém o método principal é transferida para a máquina virtual e as demais são carregadas à medida que as instâncias de seus objetos acontecem. Todo esse trabalho é realizado por um componente da plataforma chamado *classloader* [85].

O *classloader* padrão da plataforma Java SE realiza a busca das classes requisitadas no sistema local de arquivos. Isso é suficiente quando as aplicações servidoras são totalmente instaladas no sistema que as executará.

Em uma grade computacional o dinamismo das aplicações que são executadas remotamente é muito grande. Para utilizar o carregador padrão é preciso que a cada nova execução, cópias de classes sejam realizadas para o sistema de arquivos remoto. Uma vez que não é possível prever quais classes serão instanciadas remotamente, é preciso que todas as classes sejam transferidas.

O *Grid Anywhere* cria um ambiente de execução de aplicações Java que tem como objetivo agregar os participantes da grade computacional como se fossem uma grande máquina virtual Java e as conexões de rede são utilizadas como barramentos dessa máquina. Sendo desta forma, é importante que uma aplicação que tenha objetos distribuídos pela grade seja capaz de transportar somente as classes necessárias. Para isso, é preciso um *classloader* que quando requisitado faça o carregamento sob demanda dos *bytecodes* por meio da rede de computadores.

O processo de carga de classes pela rede de computadores é realizado por diversas soluções. Entretanto, na arquitetura do *Grid Anywhere* um participante (provedor ou consumidor) da grade pode estar localizado em ambientes com restrições de conexão como, por exemplo, *firewalls*, NATs ou *proxies* que inviabilizam o recebimento de requisições de conexão.

O *URLClassLoader* permite que classes Java sejam hospedadas em um servidor HTTP que é requisitado por um *classloader* [86]. Observando o cenário da grade computacional proposta, é possível notar a não existência de uma arquitetura centralizada de fornecimento de classes. Isso implica que um participante que atua no papel de consumidor de recursos deveria ter um servidor HTTP em operação para que seja possível fornecer o executável ao provedor. Isso é inviável em função do tipo dos equipamentos que podem atuar como consumidor e em função do perfil dos usuários.

A opção de utilização do JXTA é uma alternativa atrativa para a carga de classes de forma descentralizada, sendo resolvidos, inclusive, os problemas de conectividade apresentados anteriormente [87]. No entanto, é preciso que as aplicações sejam desenvolvidas utilizando essa plataforma.

A interoperabilidade entre os participantes de uma grade computacional é de uma importância muito grande. É necessário que um dispositivo seja capaz, independentemente de sua plataforma, de fornecer classes a um provedor de recursos. Sendo assim, a utilização de serviços *Web* apresenta-se como uma excelente solução tendo em vista sua independência de linguagem de programação e sistema operacional.

O WSBCL (*Web Services BasedClassLoader*) é um modelo de carregamento de classes totalmente baseado em serviços *Web* que tem como objetivo permitir a transmissão de *bytecodes* entre participantes heterogêneos e com restrição de conectividade.

5.3.1 Arquitetura

Para que um *middleware* possa fazer uso do WSBCL como carregador de classes é interessante que ele não tenha necessidade de se preocupar com as questões que são resolvidas por este *classloader*.

Sendo assim, o WSBCL provê uma transparência de arquitetura ao ambiente que necessita realizar processos de carga de classes de maneira que esse sistema considere uma arquitetura cliente-servidora convencional, conforme ilustrado pela figura 42.

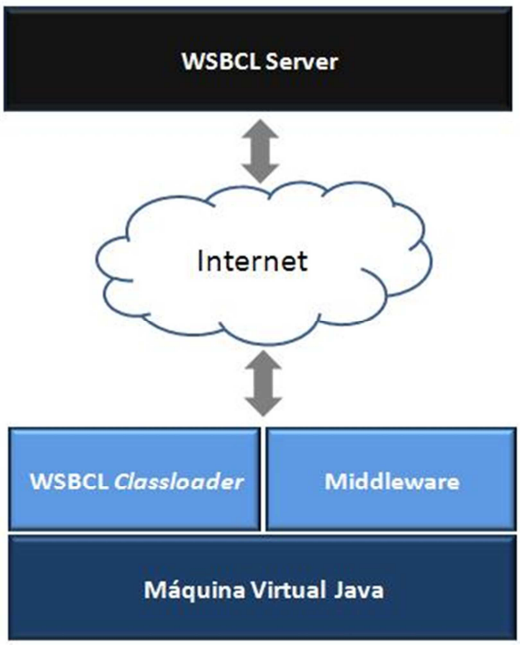

Figura 42: Arquitetura ciente-servidor interpretada pelo provedor

O provedor de serviços tem uma visão de que seu carregador de classes (que atua como cliente) se conecta a um servidor na Internet que é responsável pelo fornecimento dos *bytecodes*. No entanto, conforme citado anteriormente, esse servidor pode estar localizado em situações onde não seja possível estabelecer conexões com ele.

Além disso, o consumidor de recursos da grade computacional (que nesse caso hospeda o servidor de classes) pode ser um equipamento de baixa potência computacional que inviabiliza a execução de um servidor HTTP para a disponibilização do serviço *Web* responsável por fornecer as classes.

Para o cliente (*classloader*) essas questões devem ser transparentes. Por isso, o servidor do WSBCL é representado utilizando-se uma caixa preta. No entanto, na realidade, o servidor de classes é composto de maneira que ele não seja um agente passivo no procedimento de estabelecimento de conexões, mas sim um elemento ativo.

Para viabilizar isso, é feito uso de um componente intermediário denominado *Class Relay* que é responsável por ser, obrigatoriamente, o único componente passivo na arquitetura. Por isso, é preciso que ele esteja localizado em um ambiente onde requisições externas possam ser realizadas sem restrições. Na figura 43 é ilustrada a composição real da arquitetura implementada pelo WSBCL.

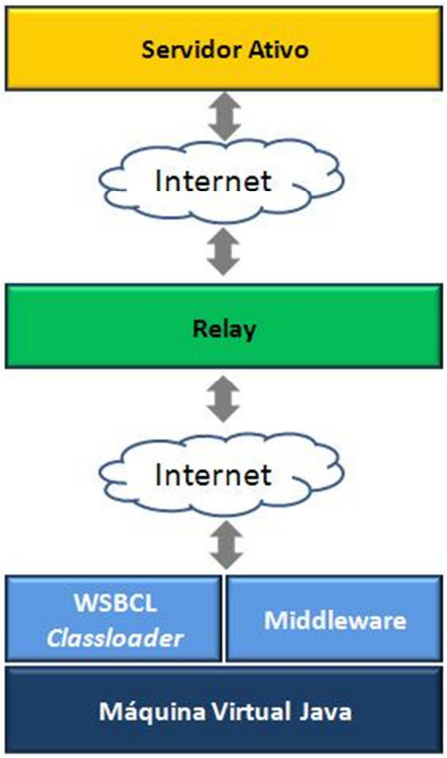

Figura 43: Arquitetura implementada pelo WSBCL

Uma vez que nesse modelo somente o *relay* atua como elemento passivo, o servidor de classes, que é executado no consumidor de recursos da grade, atua como um agente ativo realizando chamadas ao serviço *Web* localizado no *relay* para encontrar solicitações de classes e atendê-las.

A arquitetura demonstrada na figura 43 considera que tanto o provedor quanto o consumidor não podem receber requisições externas, por isso o *relay* é disponibilizado por um terceiro computador. Mas, é possível que qualquer um desses dois participantes possua condições de conectividade que os permitam atuar como elementos passivos. Por isso, o *relay* pode também ser hospedado localmente, conforme ilustrado na figura 44.

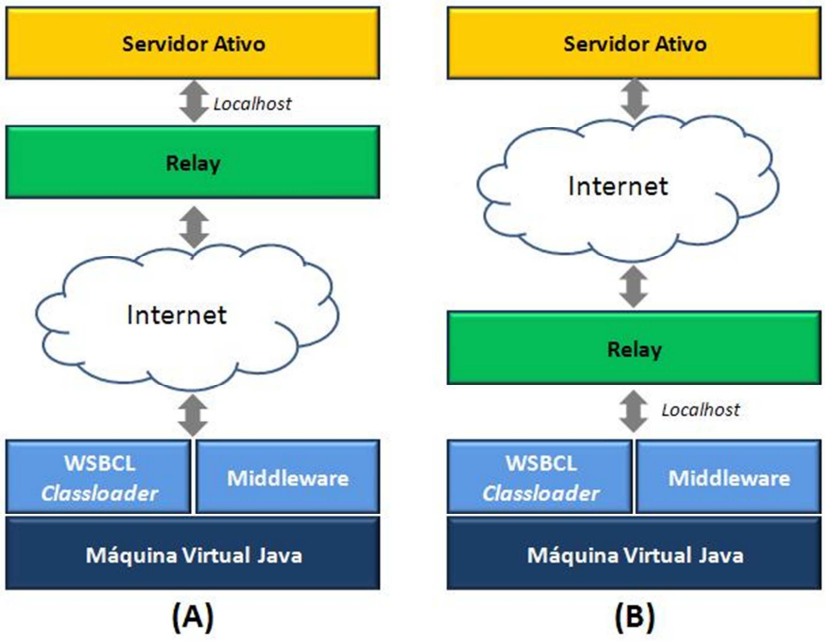

Figura 44: Configurações com *class relay* local

Na opção “A” é demonstrada a situação onde o consumidor de recursos oferece um *relay* próprio para que os provedores que estão o atendendo possam carregar as classes necessárias. O cenário “B” constitui um provedor de recursos que disponibiliza localmente um *relay* para seus consumidores de recursos.

5.3.2 Comunicação entre Carregador de Classes e Servidor Ativo

Uma vez que o *middleware* apresentado nesta tese tem como objetivo permitir que diferentes classes de dispositivos possam compor a grade computacional, é preciso que o servidor ativo possa ser executado em diferentes plataformas.

Determinadas plataformas podem suportar linguagens de programação específicas. Por isso, o *Grid Anywhere* faz uso de um sistema de carregamento de classes baseado em serviços *Web*.

O *relay*, que é responsável pelo encaminhamento das requisições e das classes entre o servidor e o carregador de classes, apresenta operações que permitem que o carregador de classes (localizado no provedor de serviços) realize suas requisições e que o servidor ativo (localizado no consumidor) faça o envio dos *bytecodes*. No entanto, uma vez no papel ativo, o servidor realiza um processo de busca (*pulling*) por requisições que quando encontradas são atendidas ou em caso contrário são notificadas como "classe não encontrada".

Antes de iniciar o processamento, o servidor e o carregador de classes (*classloader*) estabelecem uma sessão. Isso é feito de maneira que possa haver um gerenciamento das requisições e classes trocadas entre ambos para que seja possível realizar controle de sequência (para evitar ataques de replicação de mensagens) e assinatura das requisições e classes para evitar que falsas mensagens sejam inseridas. A figura 45 ilustra todo este processo.

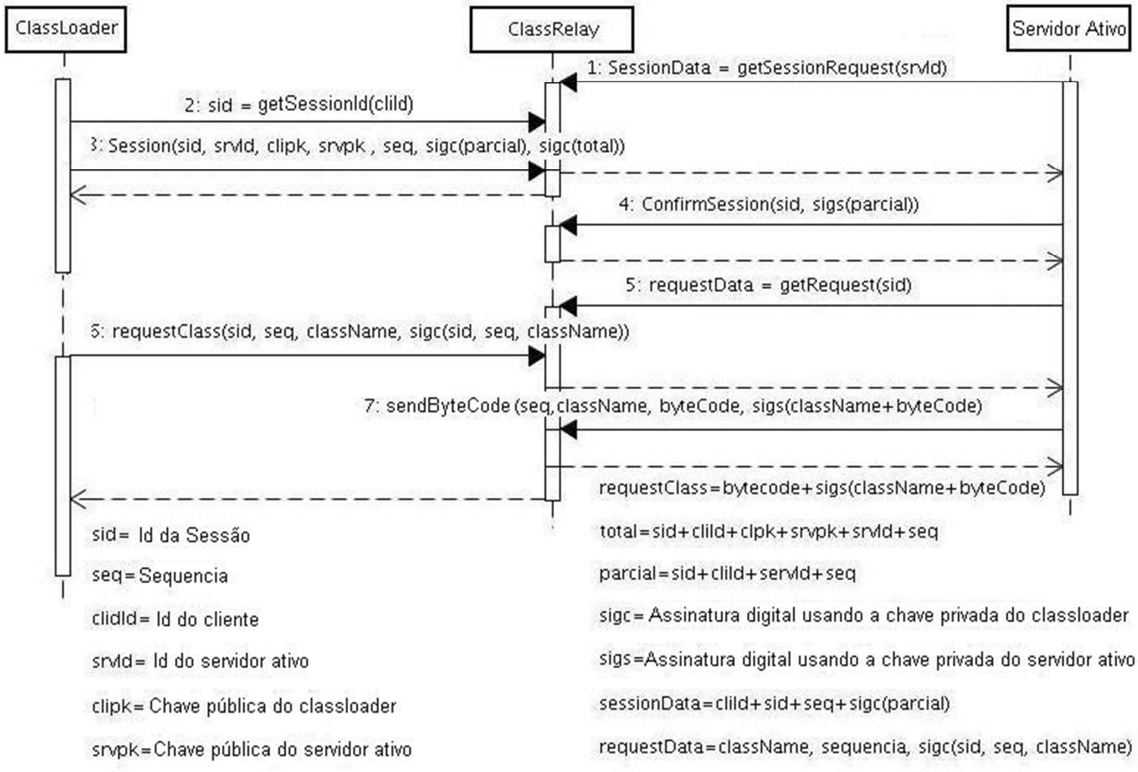

Figura 45: Diagrama de sequência do relacionamento entre *classloader*, *relay* e servidor ativo.

5.3.3 Implementação

O modelo de carregamento de classes apresentado anteriormente foi totalmente implementado utilizando-se a plataforma Java. O serviço *Web* foi disponibilizado por meio do JAX-RPC.

O fato de um serviço *Web* não possuir, originalmente, a capacidade de armazenar o estado da aplicação implica que sua simples utilização não é suficiente para implementar o *relay*, pois este precisa armazenar requisições e classes que são trocadas entre o carregador e o servidor ativo.

De maneira a resolver essa situação, o *relay* foi implementado conforme a figura 46. O serviço *Web* é utilizado para realizar a interface de comunicação com o carregador e o servidor ativo. Ao receber as mensagens o serviço as encaminha para o núcleo do *relay* que é uma aplicação Java SE responsável pelo gerenciamento das requisições e classes trafegadas. Este núcleo fica em execução contínua.

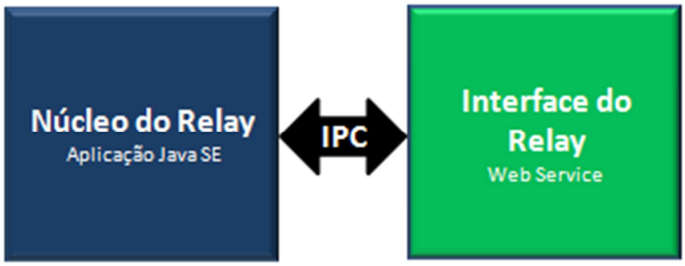

Figura 46: Arquitetura do *Relay* de Classes

O núcleo do *relay* tem sua arquitetura demonstrada na figura 47. Uma vez que esse componente atua como um intermediário totalmente passivo entre o carregador de classes e o servidor, os processos de requisição e envio de classes são realizados utilizando o princípio de *produtor/consumidor*. Sendo assim, para cada possível tipo de mensagens trocadas há uma fila.

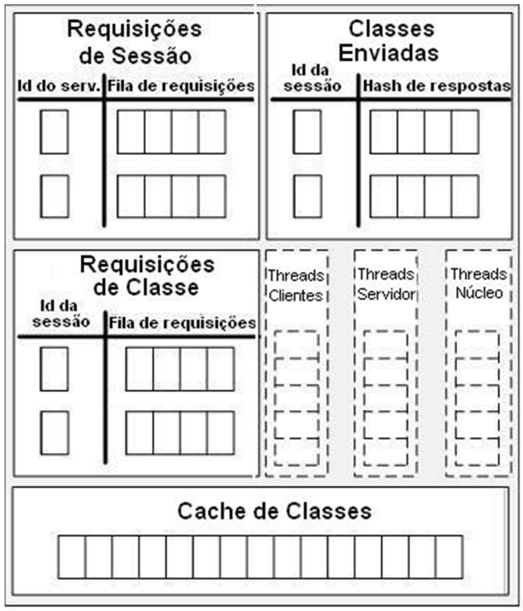

Figura 47: Arquitetura do Núcleo do *Relay*

A fila de sessões é responsável por armazenar as solicitações de estabelecimento de conexões recebidas dos carregadores até o momento em que o servidor requisitado busque por um novo pedido endereçado a ele. Ao estabelecer uma sessão, duas novas estruturas são criadas:

a) *Requisições de classes*: fila contendo os nomes das classes requisitadas pelo carregador.

b) *Classes enviadas*: tabela *hash* contendo os *bytecodes* enviados pelo servidor. Cada classe é indexada por meio de seu nome.

O algoritmo produtor/consumidor é implementado por meio das *threads clientes* e *servidoras*. No momento da requisição de classes ou estabelecimento de sessões a *thread* cliente atua como *produtora* e a *thread* servidora como *consumidora*. No momento do envio dos *bytecodes* os papéis são invertidos.

5.3.3.1 ***Avaliação de Desempenho***

A implementação do WSBCL foi submetida a uma avaliação de desempenho que teve como objetivo analisar o tempo gasto no carregamento de classes em diferentes cenários de utilização e a influência dos fatores que compõem cada cenário.

Na tabela 5 são apresentados os fatores avaliados nos experimentos. O fator "Servidor e *Relay* Juntos*"* é utilizado para compor cenários onde o servidor ativo possui

potência computacional suficiente para fornecer seu próprio *relay* (Sim) e situações onde é utilizado um *relay* localizado em um terceiro componente (Não).

Tabela 5: Fatores analisados na avaliação de desempenho

Fator	**Descrição**	**Níveis**
Link	Velocidade do link Ethernet utilizado	10/100mbps
Servidor e Relay Juntos	Determina se o relay de classes está na mesma máquina que o servidor ativo	Sim/Não
Tamanho da Classe	Tamanho da classe carregada	25k/50k
Autenticado	Utilização de autenticação	Sim/Não
Estabelecimento de Sessão	Estabelecimento de sessão antes da carga da classe	Sim/Não

O fator “Autenticado” determina se foi feito uso de assinaturas digitais para autenticar as requisições e as classes. Por último, o “Estabelecimento de Sessão” tem como objetivo determinar o custo do estabelecimento de sessões antes da transmissão das classes. Para isso, foram feitas aferições considerando o tempo de criação da sessão e da transmissão da classe (Sim) e somente da transmissão da classe (Não).

Os experimentos foram organizados por meio de um fatorial completo e cada possível composição dos fatores foi executada 100 vezes com o objetivo de determinar o tempo médio da carga de classes considerando um intervalo de confiança de 95%.

O gráfico da figura 48 é utilizado para representar os tempos de carga de classes em um *link* de comunicação de 10KBPS. O melhor caso para o carregamento de classes é aquele onde o servidor e *relay* estão na mesma máquina (diminuindo um enlace de dados), sem a necessidade de estabelecer sessão (pois isso já foi realizado anteriormente), sem a utilização de autenticação das mensagens e com classes de 25KB. Para este cenário, o processo de carga dos *bytecodes* demorou 176,545 milissegundos, em média.

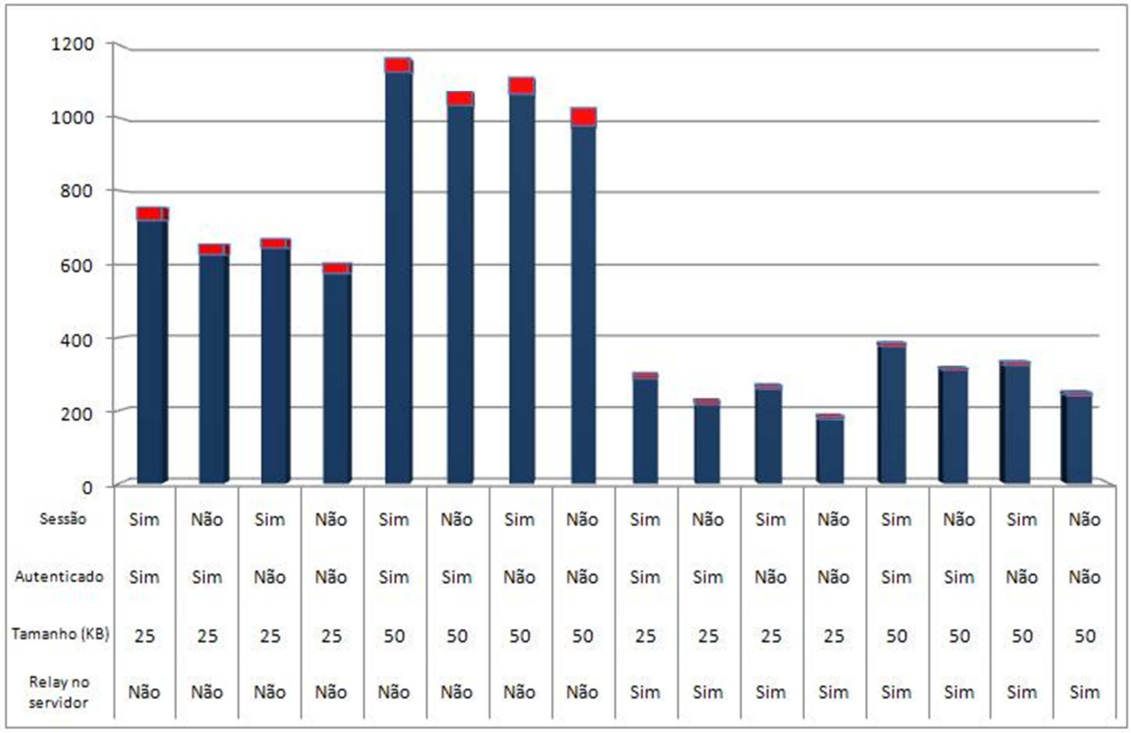

Figura 48: Tempos de carga de classes com link de 10 MBPS

O pior caso foi, conforme era possível ser previsto, o carregamento de classes com 50KB utilizando uma arquitetura com o *relay* instalado em uma máquina exclusiva, com estabelecimento de sessão e autenticação. O tempo médio foi de 1,125 segundos.

Por meio da figura 49 são representados os tempos para carregamento de classes utilizando um *link* de dados de 100MBPS. Assim como ocorreu com 50MBPS de velocidade de transmissão, o melhor caso ocorreu no cenário com sessão já estabelecida, sem autenticação e com servidor ativo e *relay* em execução no mesmo computador. Nessa configuração o tempo do processo de carga de classes de 25KB foi de 122,285 milissegundos.

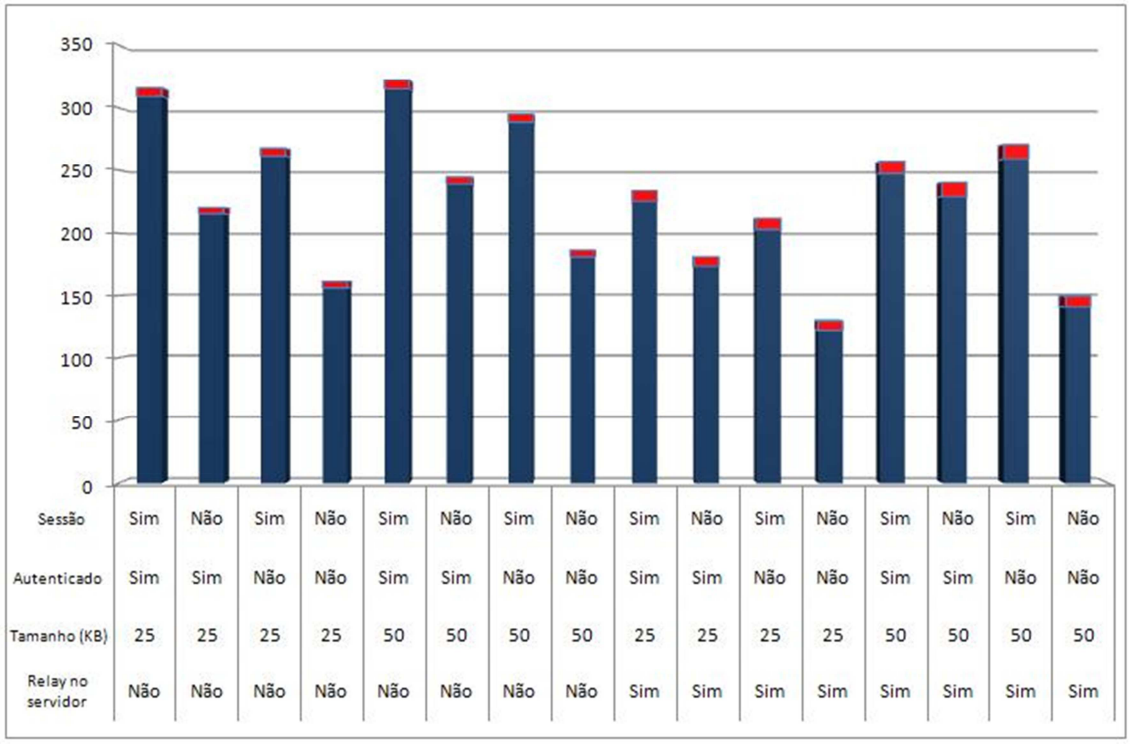

Figura 49: Tempos de carga de classes com link de 100 MBPS

O pior caso foi encontrado quando o WSBCL fez a carga de classes com tamanho de 50KB, com *relay* e servidor ativo em máquinas distintas e com uso de autenticação e estabelecimento de sessão, consumindo para isso 314,922 milissegundos.

Utilizando-se o método de regressão [84] para determinar a influência de cada fator (que é apresentada na figura 50), ou da combinação deles, foi possível confirmar a importância da qualidade do enlace de comunicação de dados. Pois este fator representou mais de 34% da influência no tempo total de carregamento de classes.

A análise dos fatores também comprovou a eficácia da possibilidade de modificar a localização do *relay* em função das características de conectividade dos participantes. Essa localização tem uma influência de mais de 28% no tempo, indicando que consumidores de serviço que possuem condições de hospedar o *relay* devem fazê-lo.

A combinação desses dois fatores (velocidade do *link* e localização do *relay*) gera a terceira maior influência no desempenho do WSBCL, representando 21,5%.

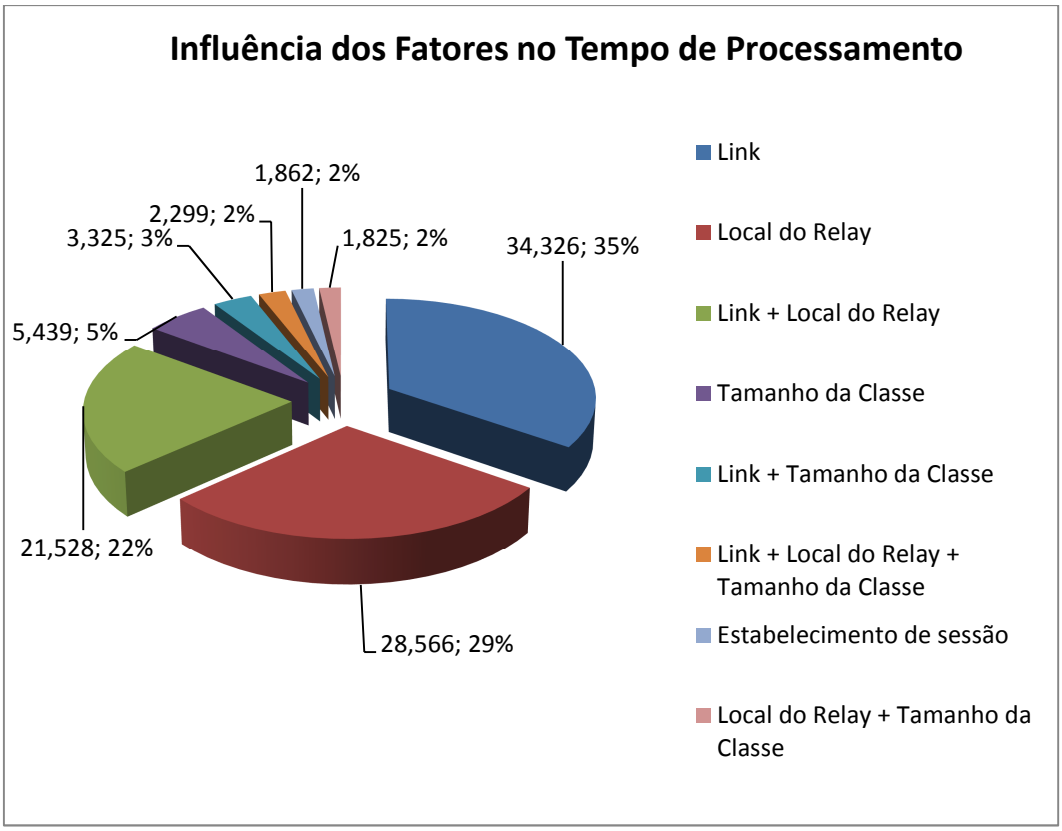

Figura 50: Influência dos fatores no processo de carga de classes

5.4 Sesiom: Ambiente de Hospedagem de Objetos Distribuídos

Aplicações podem ser desenvolvidas para terem seus componentes distribuídos pela rede de computadores de forma que um cliente possa realizar a invocação de procedimentos ou métodos localizados remotamente.

RPC (*Remote Procedure Call*), RMI (*Remote Method Invocation*) e CORBA (*Common Object Request Broker*) são opções que há muitos anos vem permitindo a construção de aplicações distribuídas [22]. No entanto, a necessidade de garantir a interoperabilidade entre aplicações desenvolvidas em linguagens diferentes e executadas em plataformas distintas motivou o desenvolvimento dos serviços *Web* que são fortemente baseados em XML.

A utilização de serviços *Web* é bastante adotada na construção de sistemas distribuídos e já é uma das bases para alguns *middlewares* para computação em grade ou em nuvem.

Porém, quando grades computacionais *desktop* são idealizadas, algumas questões dificultam a adoção de serviços *Web* na implementação:

- *Dinamismo das aplicações*: Nas grades computacionais aqui tratadas, um provedor de recursos pode, a cada momento, estar executando uma aplicação diferente. A utilização de serviços *Web* para essa finalidade requer uma quantidade considerável de implantações de aplicações. Dessa forma, são necessários mecanismos para implantação automática de serviços, o que pode gerar custos computacionais adicionais.
- *Segurança*: Os serviços devem ser monitorados para que não causem danos ao sistema local. O usuário precisa ter conhecimento para ser capaz de manter um ambiente seguro, o que pode não ser comum nos ambiente da grade computacional propostos nesta tese.
- *Servidores de aplicações*: Dependendo do perfil do provedor de recursos, a disponibilização de um servidor de aplicações pode inviabilizar sua participação.
- *Conectividade*: Os provedores podem estar sob condições de conectividade que não permitam que requisições externas possam ser realizadas, o que é comum quando se utiliza o HTTP como protocolo de transporte para SOAP.
- *Não persistência de estado*: A característica natural dos serviços *Web* de não manterem o estado da aplicação entre as chamadas é suficiente para aplicações do tipo *Bag of Tasks* em grades computacionais. No entanto, quando aplicações de outras naturezas são demandadas, isso se torna um fator limitador.

Esses fatores dificultam a utilização dos serviços *Web* na arquitetura do sistema distribuído proposto nesta tese, pois, considera-se a utilização de uma arquitetura de grade computacional como fonte de recursos para um ambiente responsável por abrigar serviços que ofertam plataformas de execução de aplicações de usuários convencionais.

5.4.1 Arquitetura

O Sesiom é uma plataforma para sistemas distribuídos que gera uma arquitetura que apresenta algumas características de agentes móveis unidas aos serviços *Web*.

Considerando uma aplicação de um usuário convencional (doméstico ou corporativo), objetos mais complexos que exigem uma maior potência computacional são migrados (ou criados) para um provedor de serviços e seus métodos são invocados utilizando o protocolo SOAP.

Nesse tipo de abordagem, uma aplicação, conforme apresentado na figura 51, faz uso da máquina virtual local para executar objetos que são simples o suficiente para permanecerem na máquina cliente ou que não possam ser migrados (por exemplo, a interface homem-máquina, IO, etc).

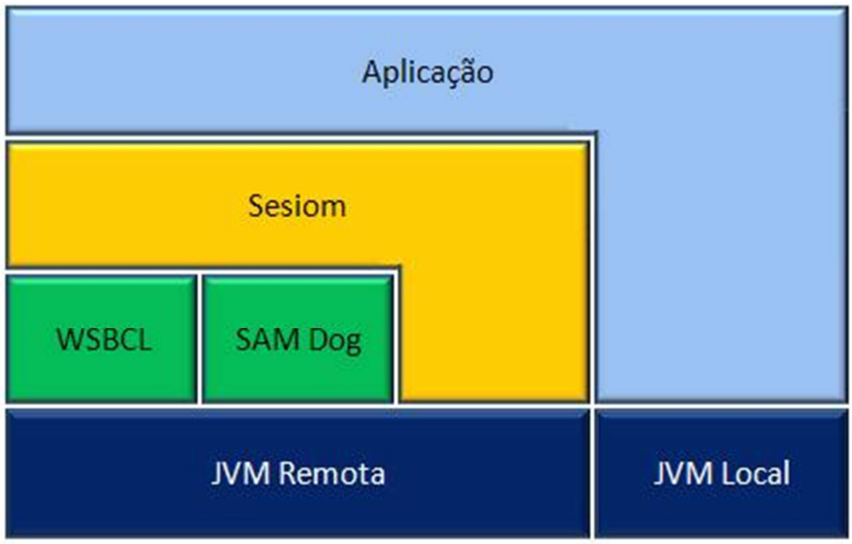

Figura 51: Organização em camadas de uma aplicação que utiliza o Sesiom

Objetos mais complexos são executados por máquinas virtuais remotas. Para isso, o Sesiom fornece uma camada de abstração para tornar simples todos os procedimentos de criação remota de objetos ou migração, chamada de métodos, entre outros.

Para gerenciar os processos de carregamento de classes e execução segura de aplicações é feito uso do WSBCL e do *SAM Dog*, que foram detalhados nas subseções anteriores.

Quando se observa uma arquitetura de sistema distribuído onde os requisitos podem ter uma heterogeneidade grande, como é o caso do *Grid Anywhere*, um modelo estático de *middleware* pode se tornar insuficiente para atender às demandas. É possível que a utilização de determinados dispositivos na grade computacional requeira recursos

que não são necessários a outros. Por isso, é preciso um servidor de aplicações que seja altamente adaptável ao ambiente.

O Sesiom adota um modelo de servidor de aplicações onde núcleos estáticos são executados em conjunto com módulos dinâmicos que são desenvolvidos e executados de acordo com os requisitos do sistema construído. Essa arquitetura é demonstrada na figura 52, sendo os módulos:

- *Gerenciador de Mensagens*: Realiza o gerenciamento das mensagens para migração, criação e recuperação de objetos e chamada remota de métodos. Esse módulo é de grande importância porque ele define como as mensagens SOAP serão transportadas entre o cliente e o servidor, permitindo que novos cenários de comunicação possam ser atendidos por meio da implementação de um novo módulo.
- *Núcleo do Controle de Admissão*: Responsável aceitar ou recusar uma nova requisição de estabelecimento de sessão.
- *Adaptador de middleware*: Permite que o Sesiom integre-se ao *middleware* da grade computacional que o utiliza como ambiente de execução de objetos remotos. No controle de admissão ele é responsável por determinar a ação a ser tomada, sendo que para isso ele se comunica com o *middleware* da grade computacional para verificar a decisão do escalonador sobre o compartilhamento dos recursos com o consumidor em questão. O *container* utiliza o adaptador para poder obter o descritor que define a política de notificações e enviar as mensagens.
- *SOAP Engine*: Realiza a chamada dos métodos dos objetos hospedados. A chamada é descrita pela mensagem SOAP recebida pelo Gerenciador de Mensagens.
- *Monitor*: Para cada sessão estabelecida um monitor é executado para realizar o acompanhamento dos recursos utilizados pelo consumidor. Esse monitoramento varia desde o gerenciamento da potência computacional disponibilizada pelo provedor até a contabilização do volume consumido.

- *Comunicador*: Módulo que permite que um objeto hospedado possa se comunicar com objetos localizados em provedores remotos.
- *Container Engine*: Realiza a hospedagem e gerenciamento dos objetos.
- *Replicador*: Responsável por gerenciar réplicas dos objetos remotos para questões de confiabilidade. Para cada ação junto a um objeto remoto, o evento *OnObjectInteration* deste módulo é invocado informando a mensagem enviada para o *container*.

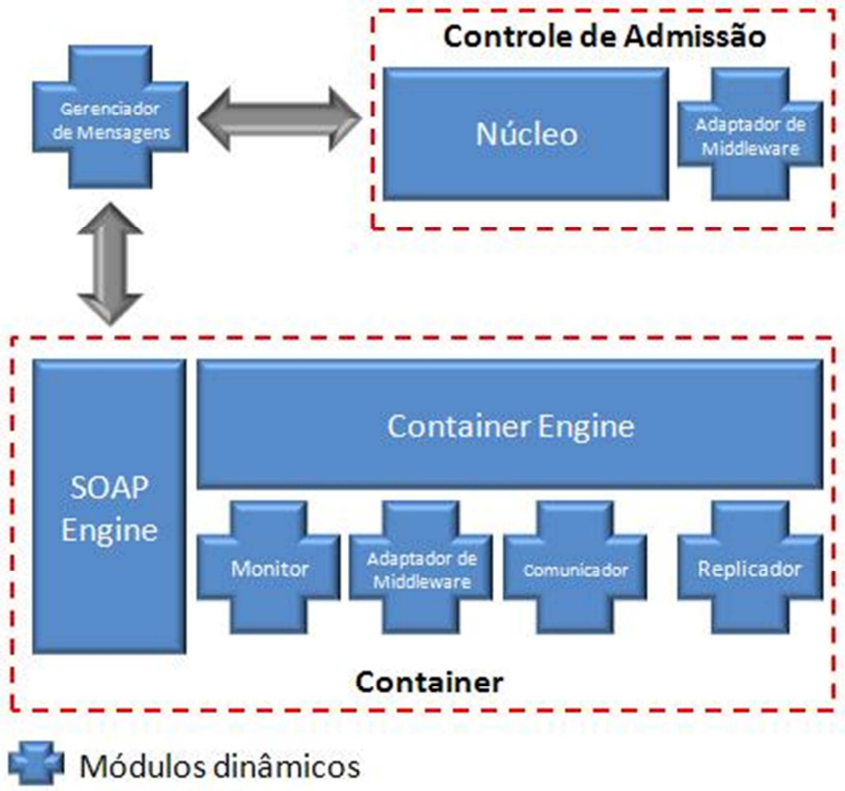

Figura 52: Arquitetura do container de aplicações do Sesiom

Quando uma aplicação utiliza o Sesiom para gerenciar seus objetos remotos é preciso que o cliente seja capaz de referenciar, de maneira simples, os componentes que se encontram hospedados no servidor. Conforme pode ser visto na figura “53-B”, o provedor hospeda os objetos e os relaciona por meio de uma tabela denominada HOT (*Hosted Objects Table*). Essa tabela tem a finalidade de realizar uma associação entre a referência da memória *heap* onde o objeto se encontra e um endereço artificial (referência remota) que é utilizado pelo cliente para se referir ao objeto remoto. Além

disso, essa tabela mantém informações sobre o momento do último acesso realizado pelo cliente.

Para que uma aplicação possa fazer uso do Sesiom para sua execução, é estabelecida uma sessão entre ela e o servidor. Quando uma nova sessão é criada, um novo *container* é ativado no servidor.

Para interagir com esse *container* o cliente recebe um *gateway* que abstrai todas as operações que podem ser realizadas remotamente como, por exemplo, criação de um novo objeto, migração de um objeto já existente, solicitar o retorno de um objeto, entre outros.

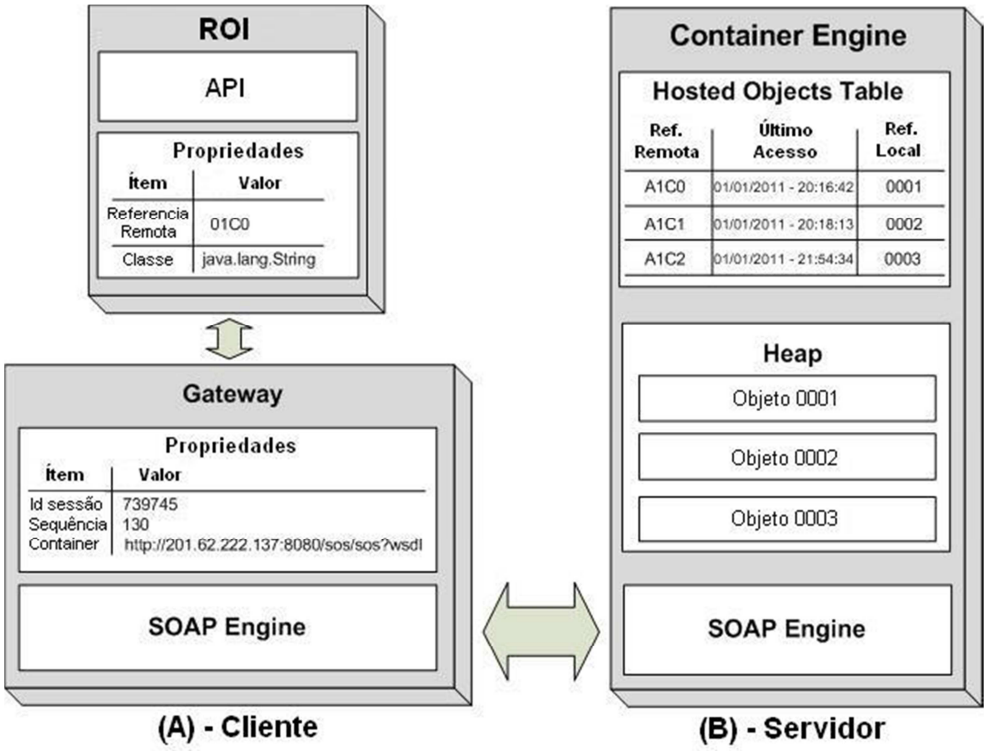

Figura 53: Ilustração de um cliente e um servidor gerenciados pelo Sesiom

Um objeto que é hospedado remotamente também necessita de um canal que permita a interação entre ele e a aplicação cliente. Para isso, toda vez que o *gateway* realiza a hospedagem de um objeto remoto ele cria um elemento local chamado ROI (*Remote Object Interface*) que é responsável por viabilizar esta interação. É mantida nesse elemento a referência remota do objeto (que está alocada na tabela HOT) e a classe ao qual ele pertence.

Os elementos *SOAP Engine* existentes tanto no cliente quanto no consumidor são responsáveis, respectivamente, pela geração e pelo consumo de mensagens SOAP utilizadas para realizar as chamadas aos métodos existentes no objeto remoto.

5.4.2 Invocação de Métodos Remotos

Em um serviço *Web* convencional a mensagem SOAP é endereçada a uma operação específica do serviço endereçado pelo protocolo de transporte. No entanto, no Sesiom a mensagem SOAP é endereçada ao container alocado para o cliente e esse container pode possuir diversos objetos hospedados.

Uma vez que o transporte é totalmente flexível no Sesiom, para que o *SOAP Engine* possa ser capaz de determinar qual é o objeto que deve ter um de seus métodos invocados, é feito uso do cabeçalho do protocolo SOAP para inserir a referência remota do objeto. Essa referência é definida no atributo "*id*" do elemento "*oid*".

Em função da potência computacional do cliente é possível escolher como deve ser manipulado o retorno da chamada de um método. São duas as possíveis alternativas:

- *Retornar objeto*: Nesse caso, o objeto a ser retornado é serializado e inserido na mensagem SOAP de retorno. No cliente, o objeto é reconstruído e disponibilizado para a aplicação.
- *Retornar referência remota*: Em alguns casos o objeto retornado pelo método pode ser complexo o bastante de forma que se torna inviável reconstruí-lo na aplicação cliente. Para isso, o Sesiom permite que o objeto a ser retornado seja hospedado no próprio servidor e apenas sua referência é retornada ao cliente por meio da mensagem SOAP. Quando isso ocorre, o *gateway* do cliente automaticamente cria um novo ROI para endereçar o objeto.

Para que seja possível o aplicativo cliente sinalizar junto ao servidor o tipo de retorno desejado, mais uma informação foi criada no cabeçalho do SOAP: o atributo "*remoteReference*" do elemento "*returnType*". Quando esse atributo possui o valor *true*, o objeto é hospedado no *container* e apenas a referência remota é retornada. O objeto só é serializado e retornado quando o valor do atributo é *false*.

É importante observar que quando um método é invocado por meio de uma mensagem SOAP o Sesiom realiza essa execução no objeto já hospedado. Sendo assim,

se essa invocação causar alterações no estado deste objeto, as modificações persistem durante todo o seu ciclo de vida.

5.4.3 SesiomLet

O *container* do Sesiom permite que objetos de qualquer classe sejam hospedados e invocados conforme descrito anteriormente. No entanto, uma categoria distinta de objetos chamada de *SesiomLets* desfruta de vantagens adicionais.

Um *SesiomLet* é um objeto que implementa uma interface especificada pelo Sesiom que permite uma interação entre *container* e objeto. A especificação determina que um SesiomLet tenha métodos para:

- *Definir uma referência para o adaptador de middleware*: O container define junto ao objeto hospedado uma referência para o adaptador. Isso permite que o objeto faça uso de APIs para, entre outros, envio de notificações, mensagens, obtenção de dados do ambiente e outras funcionalidades definidos pelo adaptador.
- *Receber mensagens*: Objetos remotos podem trocar mensagens desde que sejam *SesiomLets*.
- *Determinar ação a ser realizada no momento de sua criação remota*: Permite que o programador defina um código a ser executado quando um objeto é instanciado remotamente.
- *Determinar ação a ser realizada no momento da migração para outro container*: Método que é executado quando um objeto hospedado em um *container* é migrado para outro servidor do Sesiom.
- *Determinar ação a ser realizada no momento da chegada a um container*: Ação a ser realizada quando um objeto migrado é hospedado em um *container*.
- *Determinar ação a ser realizada no momento da destruição do objeto*: Define o código a ser executado quando o cliente requisita a destruição do objeto remoto.

5.4.4 Implementação

O modelo descrito nesta subseção da tese foi construído utilizando-se a plataforma Java. Para disponibilizar os núcleos do servidor de maneira que os módulos possam ser acoplados dinamicamente foi feito uso de injeção de dependência. Um arquivo XML descreve a classe que é responsável por implementar cada módulo.

Uma vez que um modelo simples de programação era um dos objetivos, foi disponibilizada uma API para realizar o desenvolvimento da aplicação cliente. Utilizando essa API o programador realiza a interface com o container e o objeto remoto utilizando, respectivamente, as classes *Gateway* e *RemoteObject*.

5.4.4.1 Características Básicas da API

Para realizar a criação do Gateway o cliente deve possuir apenas o endereço do servidor, o endereço do *relay* para realizar a transferência de classes e o ID do provedor, conforme o exemplo abaixo:

```
Gateway gw = new Gateway(<containerAddress>,<wsdlWSBCL>,<containerId>);
```

A hospedagem do objeto pode ocorrer de duas formas: migração ou instanciação remota. Em ambas as formas, uma referência local da classe *RemoteObject* (ROI) é criada para prover o acesso direto e transparente ao objeto remoto criado no container.

Na migração, um objeto instanciado no cliente é transportado para o provedor, sendo preciso que todas as classes envolvidas implementem a interface *serializable*. Para isso, basta solicitar a migração junto ao *gateway*:

```
<RemoteObject>ref=gw.migrateObject(<Object>);
```

Quando a instanciação remota é solicitada, o cliente envia ao provedor o pedido de criação de um objeto e todo o processo ocorre remotamente. Nesse processo não é preciso nenhum tratamento diferente nas classes envolvidas.

Para realizar a criação deste tipo de objeto basta solicitar a ação junto ao gateway informando a classe a ser utilizada.

```
<RemoteObject>ref=gw.createObject(<ClassName>);
```

É possível ainda realizar a criação direta do objeto sem haver o *container*. Quando isso ocorre, o container é criado automaticamente.

```
<RemoteObject>ref = new RemoteObject(<ClassName>,<containerAddress>,→
<wsdlWSBCL>,<containerID>);
```

Por meio do *RemoteObject* o usuário realiza as chamadas remotas ao objeto que está hospedado no container. Isto é feito utilizando um método denominado execute onde são informados o método a ser executado, a forma de retorno e os parâmetros.

A forma de retorno pode ser definida de duas formas:

- *Referência remota* (*true*): Nessa forma, o retorno do método invocado é hospedado no próprio *container* e retornado ao cliente um *RemoteObject* que dá acesso a este novo objeto.
- *Objeto* (*false*): O cliente recebe o próprio objeto retornado pelo método.

O exemplo a seguir demonstra a chamada de um método remoto que mantém o resultado no próprio container:

```
RemoteObject<Ref>=(RemoteObject)<Ref>.execute(<Method>,true,<Par#1>,…,<Par#n>)
```

Para realizar a mesma operação, mas retornar o próprio objeto sem hospedá-lo no container, basta alterar o valor do segundo parâmetro para *false*, conforme o modelo abaixo:

```
<Class><Ref>=(<Class>)<Ref>.execute(<Method>,false,<Par#1>,…,<Par#N>);
```

Uma vez que um objeto está hospedado em um container, por meio do *RemoteObject* é possível solicitar que este objeto seja transferido para o cliente. Para isso é feito uso do método *getObject*:

```
<Class><Ref>=(<Class>)<Ref>.getObject();
```

Deve ser observadoque a migração de um objeto de um container para outro (por motivos de balanceamento de carga, por exemplo) torna-se uma tarefa simples, pois basta realizar uma operação *getObject* junto a um container e uma operação de exportação junto ao outro.

Finalmente, quando um objeto remoto não será mais utilizado, o cliente pode solicitar sua remoção por meio do método *kill* junto ao *RemoteObject*:

```
<Ref>.kill();
```

5.4.4.2 ***Estudo de Caso***

Para demonstrar a utilização do Sesiom na construção de um sistema distribuído foi criada uma aplicação para realizar a predição da classe de pontualidade de pagamento de um cliente.

Foi criado um banco de dados sintético contendo 30.000 clientes que é utilizado como base de treinamento para uma rede neural artificial *multilayer perceptrons* [88]. Para cada cliente foi definido:

- *Perfil*: composto pelo tipo de trabalho (autônomo ou empregado), tempo de trabalho, tempo de conta bancária e garantia de fiador (sim ou não).
- *Classe de pontualidade*: Determina a pontualidade no pagamento, podendo ser A, B ou C.

Esse aplicativo realiza o treinamento de uma rede neural utilizando a Weka API [89] e a classificação de um novo cliente (que tem seu perfil informado por meio de uma interface GUI criada com o Java *Swing*). A imagem da figura 54 apresenta o diagrama de classes.

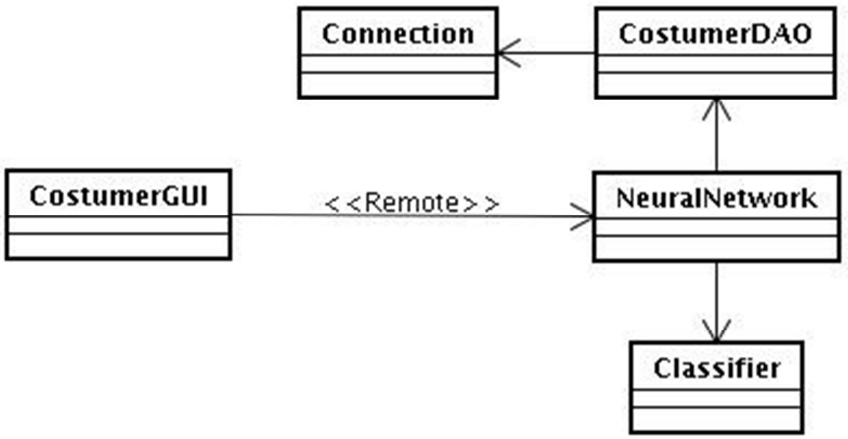

Figura 54: Diagrama de Classes da Aplicação

Embora a primeira versão dessa aplicação tenha sido desenvolvida para total execução local, o projeto de classes já contemplou uma análise para determinar quais os objetos poderiam ser executados remotamente. Para isso, utilizou-se o estereótipo "*remote*". É preciso observar que se o objeto da classe *NeuralNetwork* for distribuído remotamente, todos os objetos a ele ligados também serão distribuídos.

Para o desenvolvimento foi utilizado um notebook *Acer* com processador *Turion* (64 bits) de 2.2GHz e 2GB de memória RAM. Durante a fase de testes o tempo de leitura do banco de dados e treinamento da rede neural foi de 4.5 minutos, em média.

No entanto, no ambiente de produção a aplicação deveria ser executada em um *Thin-Client* que possui um processador *Cyrix* de 266MHz, 128MB de RAM e Linux *Slitaz*. Como já era esperado, ao tentar executar a aplicação o equipamento não teve potência computacional suficiente e tornou-se inoperante.

A alternativa para viabilizar a utilização do *Thin-Client* foi realizar a distribuição dos objetos mais pesados de forma que fossem executados remotamente. Para isso, foi utilizado um computador com processador Intel Core I7 e 4GB de memória RAM com o *container* do Sesiom instalado.

Normalmente, o processo de adaptação de uma aplicação convencional para que ela tenha seus objetos distribuídos não é trivial. No entanto, a utilização do Sesiom demonstrou uma maneira simples e rápida para realizar esta alteração.

Ao planejar a modificação foi determinado que somente a interface homem-máquina permaneceria com sua execução realizada no sistema local (*thin-client*) e os demais objetos (*CostumerDAO*, *Connection*, *NeuralNetwork* e *WekaClassifier*) seriam hospedados no servidor.

Todo o processo de alteração e distribuição da aplicação demorou pouco mais de 3 minutos. O primeiro passo foi realizar a importação das APIs do Sesiom, incluindo no início do arquivo a instrução:

```
import br.usp.icmc.lasdpc.sesiom.client.RemoteObject;
```

Após isso, foi alterada a forma de criação do objeto da classe *NeuralNetwork* para realizar sua instanciação no container. A instrução que realizava a criação local da rede era originalmente:

```
NeuralNetwork nn = new NeuralNetwork();
```

E foi substituída por:

```
RemoteObject rcd = newRemoteObject("NeuralNetwork", →
“http://192.168.0.100:8080/sosweb/sos?wsdl”,→
"http://192.168.0.100:8080/wsbclweb/wsbclService?wsdl",12345);
```

Foram alteradas as chamadas de métodos para fazer uso dos objetos remotos. Sendo assim, para realizar o treinamento o trecho de código `nn.learn()` foi substituído por `rcd.execute("learn", true)`.

Por fim, para realizar a classificação de um novo cliente o trecho de código:

```
classCode = nn.classify(salaryRange,bankTime,guarantor,jobTime,jobType);
```

Foi substituído por:

```
classCode=(String)rcd.execute("classify",false,salaryRange,bankTime, →
guarantor,jobTime,jobType);
```

Após essas alterações a aplicação foi distribuída novamente para ser executada no *Thin-Client* e a execução ocorreu com sucesso, demorando 2.5 minutos para treinar a rede.

É importante observar que uma vez que o provedor Sesiom já estava configurado, para a criação de uma nova aplicação distribuída não foi preciso realizar nenhuma transferência ou instalação no servidor, o que tornou o processo bastante simples e rápido.

5.5 API Grid Anywhere

A construção de uma grade computacional envolve diversos requisitos que são impostos em função da arquitetura do sistema composto. O *Grid Anywhere* é um *middleware* para grades computacionais que permite a construção de ambientes de compartilhamento de recursos compostos por diferentes categorias de participantes.

Conforme pode ser visto na figura 55, a aplicação é criada com o objetivo de fazer uso de recursos locais e remotos. Para que objetos possam ser hospedados e executados remotamente é feito uso do Sesiom. No entanto, esse componente, sozinho, não é capaz de descobrir recursos, negociar e escalonar atividades para eles.

O *Grid Anywhere* entra como uma infraestrutura de software e APIs que permitem que a aplicação realize a alocação de recursos que são utilizados durante a execução.

Devido à grande quantidade de classes de equipamentos computacionais que são encontrados atualmente, a arquitetura da grade computacional deve ter uma composição dinâmica para que diferentes cenários possam ser atendidos.

Por isso, o *Grid Anywhere* determina a utilização de núcleos estáticos que implementam as características básicas enquanto os requisitos com maior variação em função da arquitetura são disponibilizados por módulos acopláveis, que devem ser implementados utilizando a especificação imposta pelo *middleware*.

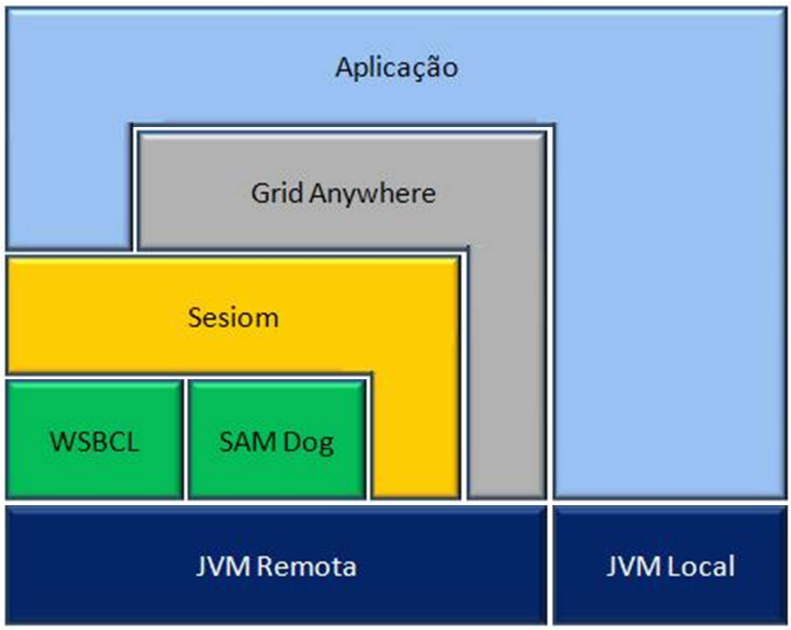

Figura 55: Arquitetura em Camadas do *Grid Anywhere*

A figura 56 contém um diagrama de blocos da arquitetura básica do *Grid Anywhere*. São os módulos do provedor:

- *Núcleo*: Componente responsável por realizar a integração e gerenciamento dos módulos que compõem o provedor de serviços.
- *Escalonador:* Responsável por realizar a negociação dos recursos locais junto ao consumidor de recursos. Por ser um módulo acoplável, novas políticas de escalonamento, formas de comunicação, entre outros, são flexíveis.
- *Listener do Escalonador*: É um módulo com um segundo nível de acoplamento permitindo refinar o comportamento do escalonador na implementação do *middleware*. A cada nova requisição de recursos esse módulo é ativado podendo interferir na decisão de admissão do cliente.

- *Gerenciador de Arquivos*: Permite a realização de processos de *Stage In* e *Stage Out* de arquivos. Pode ser implementado e acoplado ao núcleo de acordo com as necessidades da arquitetura implementada.
- *Listener do Gerenciador de Arquivos*: Localizando no segundo nível de acoplamento, permite ao desenvolvedor do *middleware* refinar o comportamento do Gerenciador de Arquivos.
- *Adaptador do Container*: Módulo responsável por realizar a comunicação do núcleo com o *container* do Sesiom. Uma das principais funções dessa comunicação é informar ao *container* se um pedido de execução foi autorizado ou não pelo escalonador, pois os processos de escalonamento e submissão de objetos são distintos e ocorrem de forma independente.
- *Módulos do Sesiom*: Componentes que compõem o container de execução que foram detalhados na seção anterior.

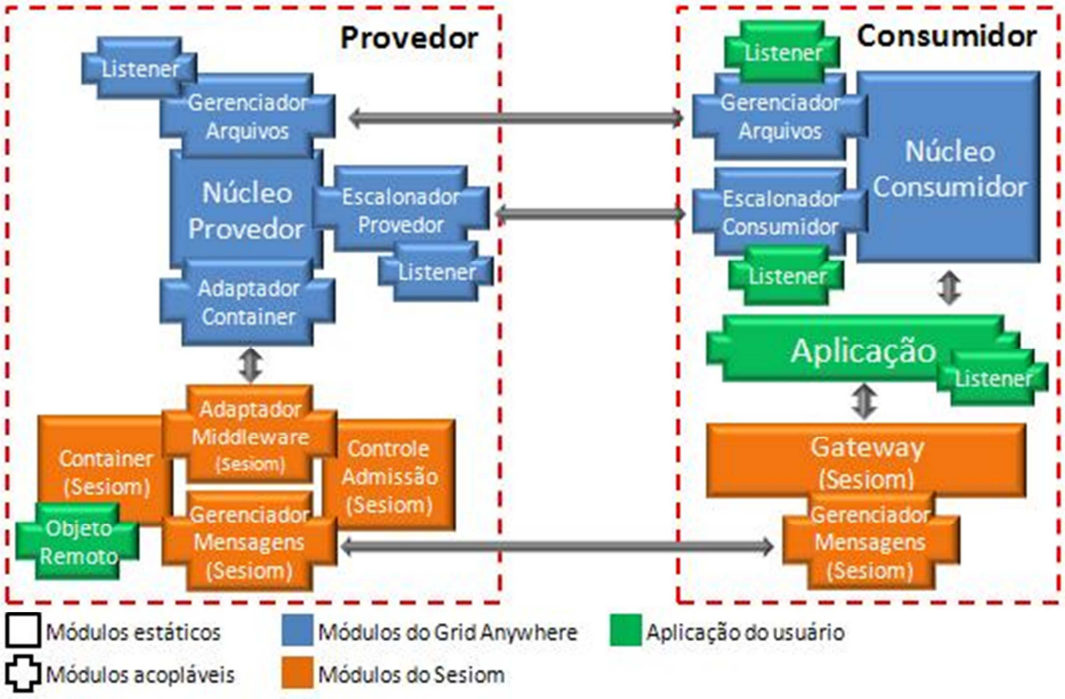

Figura 56: Diagrama de blocos da arquitetura básica do *Grid Anywhere*

No consumidor são encontrados os módulos responsáveis por viabilizar a participação do cliente na grade computacional. São eles:

- *Núcleo*: Ele é responsável por fazer o gerenciamento dos módulos acopláveis e apresentar a API de utilização da grade à aplicação do cliente.
- *Gerenciador de Arquivos*: Responsável por realizar, em conjunto com o mesmo componente servidor, as transferências de arquivos.
- *Listener do Gerenciador de Arquivos:* Componente implementado pela aplicação do usuário para determinar as ações a serem realizadas no momento do envio ou recebimento de arquivos e em casos de notificação de erros.
- *Escalonador*: Responsável por localizar provedores e negociar com eles pela utilização dos recursos remotos.
- *Listener do Escalonador:* Permite que o programador da aplicação especifique ações a serem tomadas durante os eventos de escalonamento como, por exemplo, localização de um provedor, desconexão e mudança de carga do servidor, entre outros.
- *Módulos do Sesiom:* Componentes clientes que permitem o acesso ao *container* de aplicações remoto.
- *Listener da Aplicação:* É possível que um método remoto seja invocado paralelamente em uma grade computacional *desktop*. Nesse caso, um número indeterminado de respostas pode ser obtido. Por isso, quando o Sesiom identifica um ambiente paralelo de execução ele obriga que o programador informe um componente *listener* (*ResponseListener*) que será notificado sobre o recebimento de uma resposta e realizar seu tratamento por meio do evento *OnResponse*. Esse componente deve oferecer ainda a implementação de um evento chamado *OnError* que é utilizado para realizar o tratamento de exceções de execução ocorridas nos provedores.

5.5.1 Notificação de Eventos

É importante que o estado da execução de um objeto possa ser acompanhado pelo cliente e que o estado global de um provedor possa ser acompanhado pelo seu

administrador. O *Grid Anywhere* especifica a existência de um documento XML que é responsável por descrever o processo de notificação e métodos para realizar a recepção e propagação de mensagens nos adaptadores de *middleware* e *container*.

Ao observar a arquitetura do Sesiom é possível observar que o monitor de execução é um módulo acoplável. Sendo assim, ele é implementado de forma a processar o descritor de notificações e realizar o envio de mensagens por meio do *adaptador de middleware*.

O *container* do Sesiom por sua vez realiza o monitoramento constante do estado global do provedor e envia as informações ao adaptador de *middleware* que, de acordo com o descritor, as propaga ou as ignora.

5.5.2 Arquiteturas Propostas

Utilizando o *Grid Anywhere* foram propostas duas arquiteturas de grades computacionais, que são apresentadas nas próximas subseções.

5.5.2.1 Grade Computacional como Infra Estrutura de um Ambiente de PaaS por meio da utilização de SOAP Over SOAP

Uma variação do conceito de plataforma como serviço (PaaS) é uma alternativa para realizar a execução de aplicações de usuários convencionais em um servidor de aplicações. É possível utilizar essa abordagem para permitir que equipamentos com recursos limitados sejam capazes de executar aplicações mais complexas por meio da distribuição de seus objetos que exigem maior potência computacional.

Anteriormente foi demonstrada a utilização do Sesiom para prover um ambiente de execução de uma aplicação distribuída. No entanto, nessa configuração foi feito uso de um servidor centralizado que foi disponibilizado exclusivamente para fornecer esse tipo de serviço.

Entretanto, para ser capaz de fornecer um serviço com menor custo faz-se necessária a utilização de uma grade computacional como fonte de recursos computacionais para o fornecimento de plataforma como serviço. Para isso, é feito uso da arquitetura do *Grid Anywhere* conforme mostrado na figura 56 com a implementação dos módulos necessários.

Com o objetivo de permitir que diferentes cenários de utilização possam ser viabilizados, o Sesiom admite que novos mecanismos de transporte de mensagens SOAP possam ser implementados por um *Gerenciador de Mensagens* e acoplados ao *SOAP Engine*. Essa característica é muito importante para o *Grid Anywhere*, pois é ela que torna possível a integração de diferentes dispositivos à grade computacional.

Conforme citado na seção (5.3) sobre o WSBCL, é possível que os participantes da grade computacional tenham restrições de conectividade que impeçam o recebimento de conexões de redes externas. Para isso, um servidor Sesiom não pode atuar no papel passivo do protocolo HTTP para receber as mensagens SOAP.

Esse problema é resolvido utilizando-se os mesmos conceitos de servidor ativo e *relay* empregados no WSBCL. Conforme ilustrado na figura 57, um *relay* de requisições é o único elemento passivo na arquitetura e opera entre o cliente e o servidor.

Dessa forma, quando um cliente deseja invocar um método ele se conecta ao *relay* e envia a mensagem que representa a requisição. O *container*, por sua vez, também se conecta ao *relay* para solicitar mensagens destinadas a ele, caracterizando um processo de *pulling*.

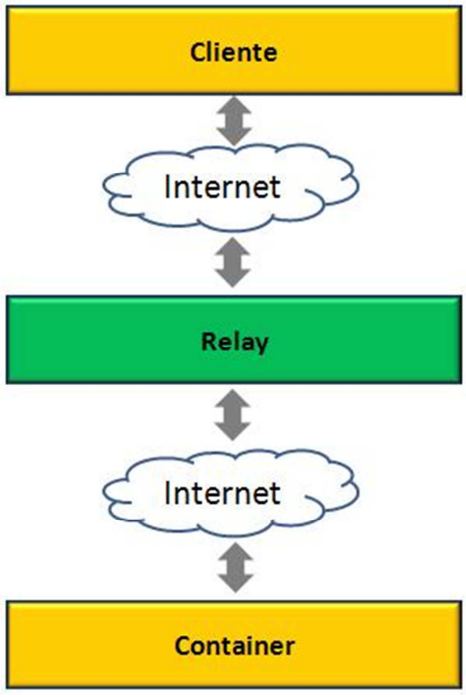

Figura 57: Relay de requisições

A requisição de invocação do método do objeto remoto é descrita por uma mensagem SOAP (#2) que é gerada pelo *gateway* existente no cliente, conforme figura 58. Uma vez que para transportar essa mensagem até o *relay* é feito uso de serviços *Web*, ela é encapsulada dentro de outra mensagem SOAP (#1) gerada pela *stub* do cliente e encaminhada via requisição HTTP ao *relay*, que a armazena em uma fila destinada à sessão.

Como o servidor é ativo, ele realiza uma invocação ao serviço *Web* do *relay* solicitando requisições destinadas a ele. Quando uma nova requisição é encontrada, a mensagem SOAP que a representa (#2) é encapsulada dentro da mensagem SOAP de retorno da chamada do serviço *Web* do *relay* (#1) cujo transporte é feito pela resposta HTTP.

Sendo assim, o *stub* do servidor ativo processa a resposta da chamada do serviço *Web* do *relay*, obtém a resposta (que também é uma mensagem SOAP) e a entrega ao *SOAP Engine* do Sesiom para processamento.

A resposta da execução do método é retornada ao cliente utilizando o mesmo princípio da chamada: inserindo uma mensagem SOAP dentro de outra.

Uma análise da figura 58 é importante. Ela é baseada na figura 4 da seção (2.2) sobre serviços *Web*. Na figura original é ilustrado um ambiente convencional com um servidor passivo, onde o processamento da requisição é feito pelo *skeleton*, elemento que não existe no servidor ativo, estando presente somente no *relay*.

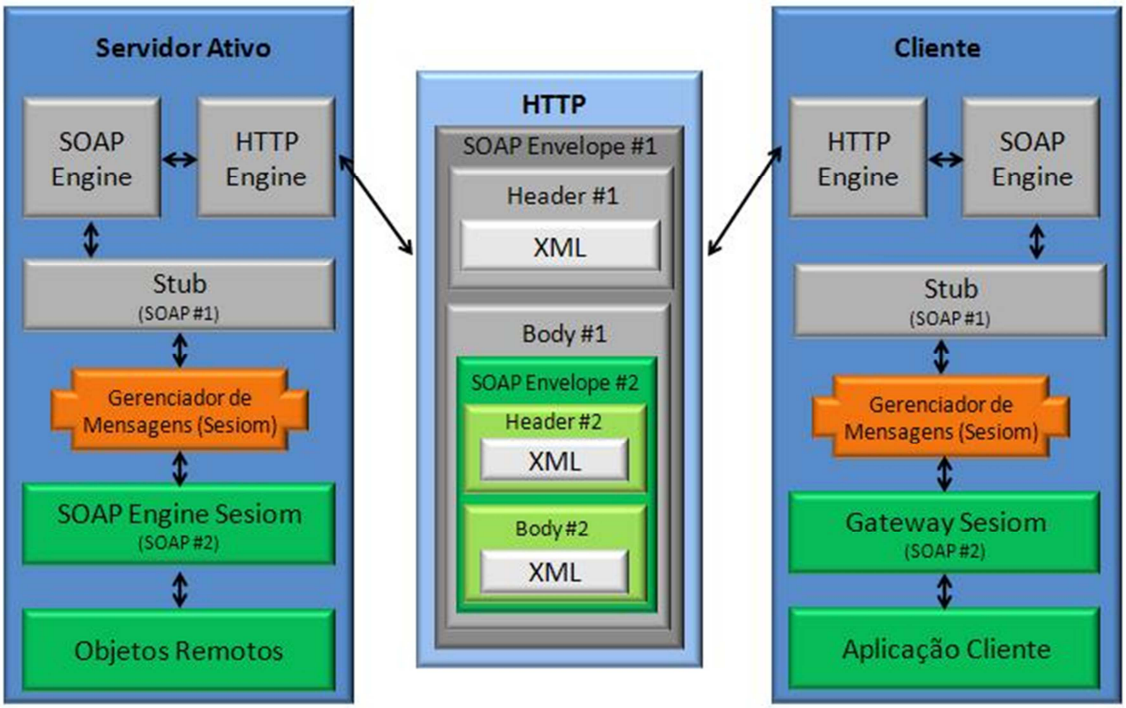

Figura 58: Ilustração da utilização de SOAP Over SOAP

5.5.2.2 *Utilização de SOAP Over DTV para Viabilizar Set-top Boxes como Provedores de Recursos*

As grades computacionais *desktop* vem há anos fazendo uso de computadores pessoais para realizar processamento paralelo de aplicações distribuídas por uma determinada instituição. Aplicações são enviadas aos colaboradores que as executam no formato de *Bag-of-Tasks* e retornam os resultados.

O *Grid Anywhere* permite a construção de grades computacionais *desktop* baseadas na migração de objetos para os voluntários e execução paralela de métodos por meio do protocolo SOAP. Dessa forma, um único objeto migrado para os voluntários pode ter seus métodos invocados quantas vezes forem necessárias.

No entanto, na grade aqui apresentada os voluntários são receptores digitais de televisão interativa que recebem dados de uma emissora via *broadcasting* e retornam informações por meio de uma conexão de Internet.

Nessa arquitetura, um consumidor de recursos que deseja realizar o processamento paralelo de uma aplicação não tem contato direto com os voluntários, mas sim com a emissora de TV. Sendo assim, o cliente envia um objeto (ou pede sua criação) à emissora que faz o encaminhamento, via difusão, para todos os receptores sintonizados a ela.

O *Grid Anywhere* permite a execução das aplicações de duas formas: a primeira utiliza o formato de *batch* (*Bag-of-Tasks*), quando o objeto é um *SesiomLet*, sendo a implementação iniciada pelo próprio container no momento de sua hospedagem. A segunda opção faz uso do protocolo SOAP para realizar a chamada remota dos métodos.

Conforme pode ser analisado na figura 59, quando o consumidor deseja fazer uso dos receptores de TV para executar uma aplicação paralela, primeiramente ele realiza, por meio das APIs do *Grid Anywhere*, o escalonamento de recursos. Esse escalonamento de recursos ocorre observando qual emissora possui uma melhor possível potência computacional naquele momento.

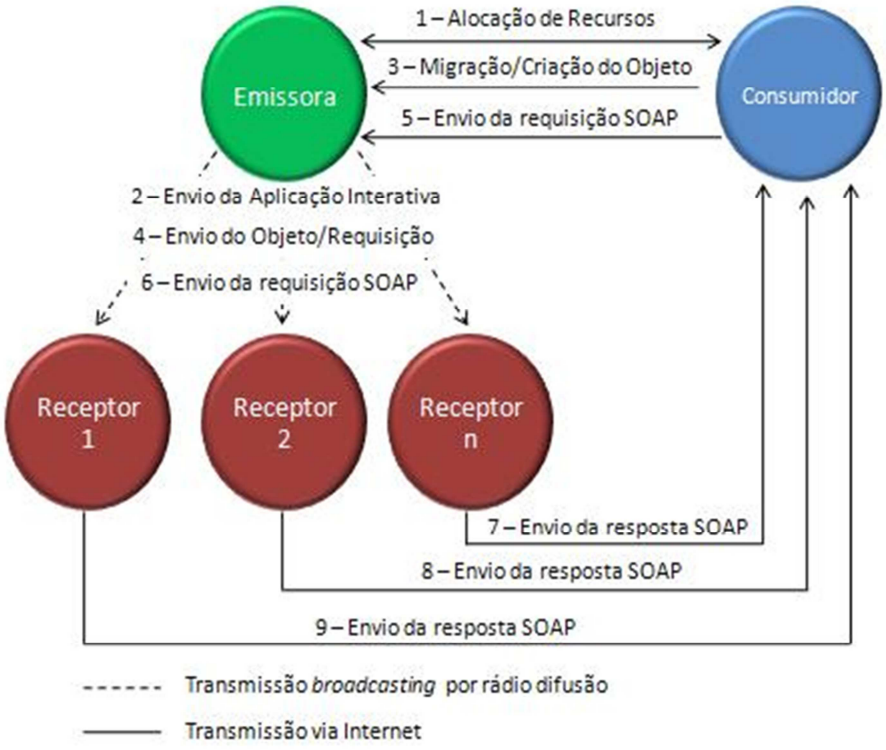

Figura 59: Utilização de SOAP em um ambiente de difusão

Quando uma emissora é selecionada para ser utilizada, ela envia uma aplicação interativa (*XLet*) via difusão que é apresentada aos usuários que possuem o *Grid Anywhere* instalado no receptor. Os telespectadores, por meio dessa aplicação, autorizam ou rejeitam (utilizando o controle remoto) a execução da aplicação da grade.

Quando o consumidor termina de realizar a alocação dos recursos ele envia o objeto a ser migrado ou a requisição de criação para a emissora que faz o encaminhamento para os receptores que realiza a hospedagem do objeto.

Para realizar a chamada de um método, que ocorre de forma paralela, o consumidor envia uma mensagem SOAP para a emissora que faz a transmissão para os receptores que recebem essa mensagem, executam o método indicado e retornam a mensagem SOAP de resposta por meio da Internet.

Para que esse modelo de execução paralela possa ser implementado é feito uso de uma configuração de módulos do *Grid Anywhere* que é demonstrada na figura 60. Nessa arquitetura há dois níveis de provedor de recursos. No primeiro nível encontra-se a emissora de TV e no segundo os receptores digitais (*set-top boxes*).

Os módulos acopláveis do *Grid Anywhere* foram desenvolvidos e organizados de forma a atender cada um dos níveis de provedor. Muitos componentes se encontram em ambos, mas com requisitos diferentes:

- *Escalonador*: Na emissora é responsável por negociar com o consumidor pelo uso da grade *desktop*. Esse módulo no receptor é responsável por determinar se a aplicação deverá ser executada localmente de acordo com as preferências do telespectador que podem incluir, por exemplo, o limite máximo de utilização dos recursos computacionais.
- *Gerenciador de arquivos*: Na transmissão entre emissora e consumidor a comunicação de dados é bidirecional e é realizada utilizando a Internet. O envio de arquivos da emissora para os receptores é realizado utilizando o multiplexador que fará a transmissão utilizando o carrossel de dados. Sendo assim, no *set-top box* o gerenciador de arquivos apenas recebe dados da emissora, mas pode enviar e receber diretamente do consumidor.
- *Gerenciador de Mensagens*: Quando um objeto é invocado pelo consumidor a mensagem SOAP é propagada até a emissora por meio da Internet e sua recepção é feita pelo gerenciador de mensagens. Esse componente ao receber o documento XML realiza o seu encaminhamento via carrossel de dados e a recepção em feita no *set-top box* por um outro gerenciador de mensagens que realiza a leitura dos dados por meio do *XLET* JavaDTV. No momento do envio da mensagem SOAP de resposta, o documento é enviado via Internet diretamente para o consumidor

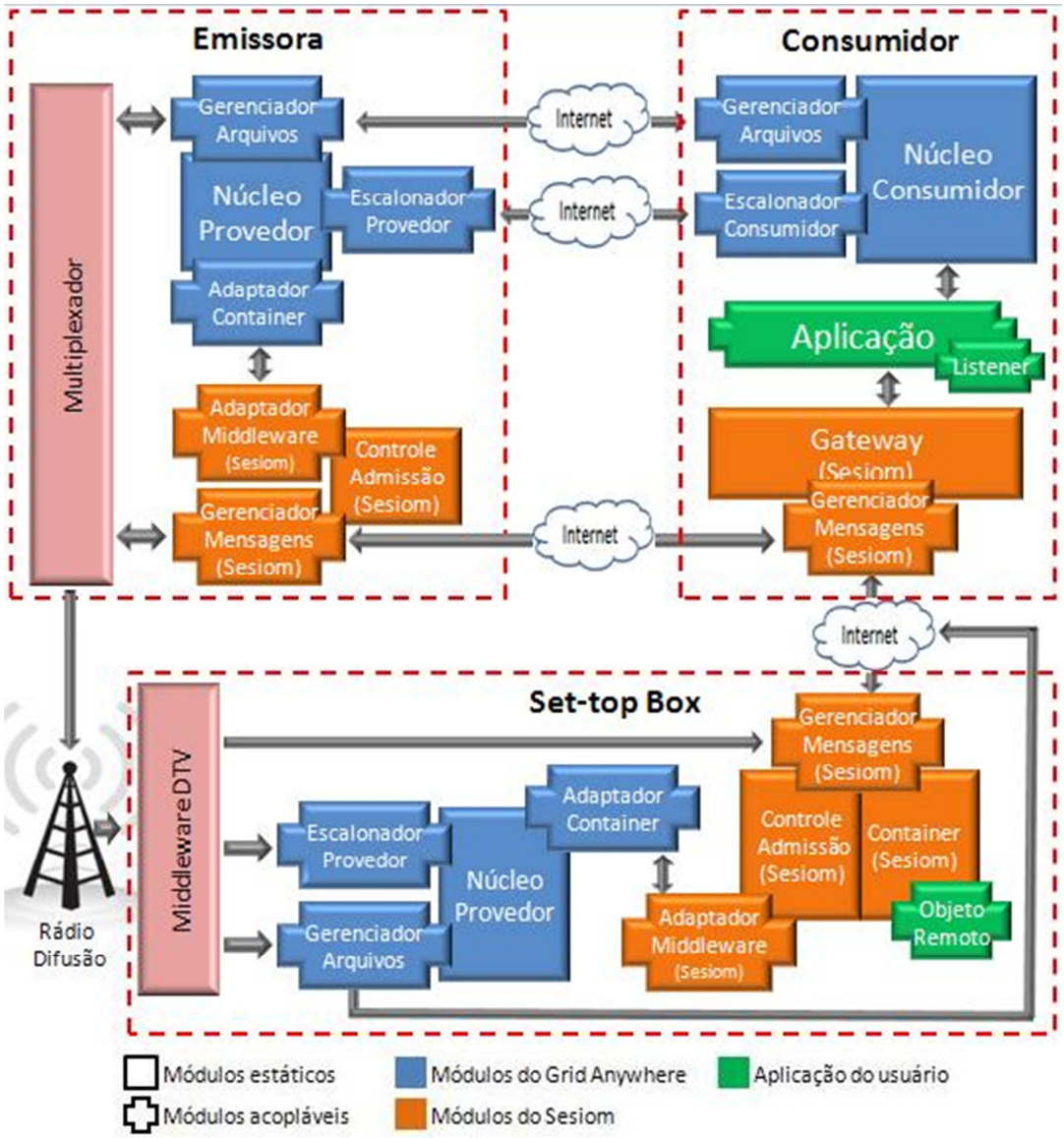

Figura 60: Arquitetura de grade computacional para utilização de receptores de TV Digital na grade computacional

5.5.3 Gerenciamento de Mensagens no Set-Top Box

Quando uma emissora admite a execução de uma aplicação para a grade ela necessita enviar uma aplicação para todos os receptores sintonizados a ela. Essa aplicação tem como objetivo permitir a interatividade do telespectador para que ele opte por aceitar ou não a execução das tarefas da grade computacional e também alimentar o *middleware* do *Grid Anywhere* instalado no *set-top box* com mensagens obtidas do carrossel de dados.

O aplicativo transmitido consiste em um *XLet* do *Grid Anywhere* que é assinado pela emissora de TV. As mensagens são documentos XML que podem conter

informações de controle (escalonamento da grade, estabelecimento de sessões do Sesiom, etc) e de chamadas remotas de métodos (SOAP).

A transmissão das mensagens é feita utilizando o *Broadcasting File System* especificado pelo JavaDTV. Na emissora, o gerenciador de mensagens do Sesiom realiza a inserção de um arquivo chamado *gaw.xml* no carrossel de dados (passos 1 e 2 da figura 61) que será posteriormente consumido pelo *XLet*.

Para cada nova mensagem a ser enviada o arquivo *gaw.xml* é atualizado pela emissora, o que é verificado pelo *middleware* DTV (3) que realiza o disparo (4) de um evento (*com.sun.dtv.broadcast.BroadcastFileEvent*) que é recebido (5) e processado pelo *XLet* (*com.sun.dtv.broadcast.BroadcastFileListener*).

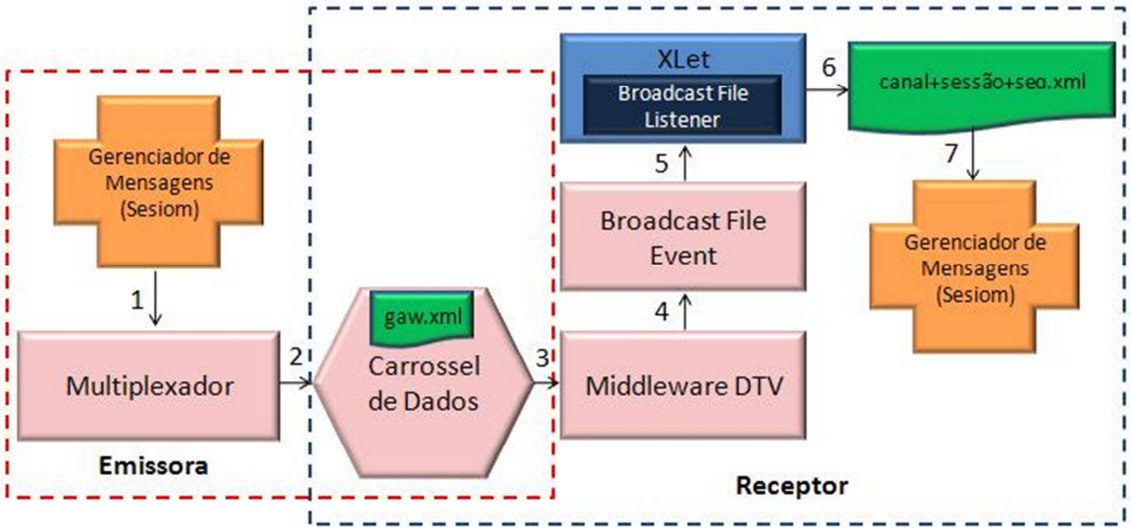

Figura 61: Transmissão das mensagens SOAP pelo fluxo de dados do sistema de TV Digital

O *XLet* tem a função de fazer a leitura dos novos dados e gerar (6) um documento no sistema de arquivos local utilizando o diretório especificado para a aplicação. Esse arquivo tem o nome composto pelo número do canal da emissora, número da sessão estabelecida pelo Sesiom e número de sequência das mensagens (dados que são obtidos do documento *gaw.xml*).

Um módulo gerenciador de mensagens realiza a busca desses arquivos e os entrega ao *middleware* para grade computacional (7). É importante observar que o processamento desses dados só é realizado quando o telespectador selecionou a opção “sim” apresentada na interface do *XLet*.

Essa arquitetura é baseada na especificação Java DTV. Isso permite que o *Grid Anywhere* seja acoplado a qualquer *middleware* que atenda às normas. Neste trabalho foi feito uso do *middleware* Ginga [73] [72] para realizar a validação do *XLet*.

Utilizando uma máquina virtual disponibilizada pelos desenvolvedores do Ginga, foram criados os receptores. Esses receptores receberam as implementações dos componentes descritos na figura 60. Para poder simular nessas máquinas virtuais o comportamento do carrossel de dados e do evento *BroadcastFileEvent* foi feito uso do NFS (*Network File System*), que por meio de um sistema de arquivos distribuído simulou a chegada e atualização dos arquivos que são transmitidos pela emissora.

Na figura 62 é apresentada a imagem da interface do Ginga realizando a transmissão de um vídeo. No momento do inicio da alocação de recursos pelo *Grid Anywhere* é apresentada a solicitação de permissão de execução de aplicações de grade. Os dados que são apresentados ao usuário são obtidos do arquivo XML transmitido juntamente com a aplicação.

Figura 62: XLet para autorização de execução da aplicação do Grid

5.6 Considerações Finais

Este capítulo descreveu o *middleware Grid Anywhere*. Foram apresentados os diversos módulos que o compõe: *Sam Dog* (segurança), WSBCL (carregamento de classes), Sesiom (*container* de execução) e a API *Grid Anywhere*. Todos os modelos foram implementados e a arquitetura básica do middleware se mostrou eficaz para a composição de novos ambientes de grades computacionais.

Com a composição dos elementos que formam a arquitetura básica foi possível realizar a construção de duas grades computacionais com objetivos distintos. Uma para ofertar plataforma como serviço para usuários convencionais e outra para utilizar os receptores digitais como provedores de recursos em uma grade computacional *desktop*.

Foi possível verificar que a fisiologia básica do *middleware* que considera o acoplamento dos módulos foi fundamental para que novos cenários fossem criados sem a necessidade de implementações muito complexas.

É importante ressaltar que, com exceção da API *Grid Anywhere*, os demais módulos (*Sam Dog*, WSBCL e Sesiom) são totalmente independentes e integráveis, o que permite que seus modelos e implementações possam ser empregados em outros tipos de sistemas distribuídos.

Esses módulos tiveram seus desempenhos mensurados e avaliados, o que demonstrou que o mecanismo de segurança (*Sam Dog*) gera um esforço computacional adicional pequeno. O WSBCL, por sua vez, pode consumir um tempo considerável no momento da carga de uma classe. Entretanto, isso ocorre somente uma vez para cada conjunto de *bytecodes* requisitado pela máquina virtual e a utilização de um meio de comunicação de dados eficiente colabora de forma significativa com a diminuição do tempo de espera.

Capítulo

6

Conclusões

6.1 Considerações Iniciais

Esta tese de doutorado teve como objetivo analisar e propor alternativas para o cenário de computação em grade atual, para que esse paradigma possa ser estendido de forma que seja capaz de abrigar novos tipos de dispositivos computacionais, atuando tanto como provedores como consumidores de recursos.

Na revisão bibliográfica, que é composta pelos capítulos 2, 3 e 4, foram apresentadas as tecnologias e metodologias para construção de sistemas distribuídos de uma forma geral, *middlewares* para construção de grades computacionais e um estudo sobre o sistema de televisão digital interativa, o qual foi idealizado como um possível meio de transmissão de dados para um sistema de compartilhamento de recursos.

O capítulo 5, que apresenta os resultados das investigações realizadas, foi dividido em subseções que são responsáveis por descrever cada um dos módulos propostos. Esses módulos, em conjunto, formam o *Grid Anywhere*, mas podem ser utilizados de forma independente em sistemas de outras naturezas.

6.2 Principais Resultados e Contribuições

Durante os estudos realizados foi possível identificar os problemas que são comuns às possíveis arquiteturas, como também aqueles que necessitam de tratamento individual para cada circunstância. Por isso, este trabalho apresenta modelos e implementações concretas para as questões que são mantidas entre as diversas possíveis

configurações de uma grade computacional e especificações que permitem que os requisitos variáveis possam ser resolvidos e acoplados, de maneira simples, àqueles já prontos.

Assim, as principais contribuições desta tese podem ser destacadas como:

- *Especificação da arquitetura*: Foram geradas interfaces que permitem que módulos com maior dinamismo sejam implementados e acoplados, por meio de injeção de dependência, aos núcleos do *middleware*.
- *Container de objetos com transporte flexível de mensagens SOAP*: Uma vez que o *middleware* proposto tem seu foco no compartilhamento de recursos de processamento, foi criado o Sesiom, um ambiente de hospedagem de objetos e invocação remota de métodos totalmente baseado em SOAP. Para isso, novos elementos foram inseridos no HEADER do protocolo para dar suporte ao modelo de execução proposto. As mensagens desse protocolo podem ter sua forma de transporte implementada em um módulo distinto que é acoplado ao *container*.
- *Transporte de SOAP por meio de outra mensagem SOAP*: Para permitir que no ambiente da grade computacional todos os participantes atuem como agentes ativos e consigam fazer parte do sistema mesmo estando em situações com conectividade restrita (atrás de *firewalls* e *NATs*, por exemplo) foram utilizados serviços *Web* para implementar um ambiente de transporte de mensagens SOAP, baseado na existência de um intermediário passivo.
- *Transporte de objetos e mensagens SOAP por meio de um sistema de televisão digital interativo*: A integração do *Grid Anywhere* com o *middleware* JavaDTV permite que o ambiente de comunicação das emissoras de TV seja utilizado como meio de transporte de objetos Java, que são transmitidos via difusão e posteriormente invocados através de mensagens SOAP, também transmitidas via *broadcasting*.
- *Carregamento de classes por meio de SOA*: As soluções para carregamento de classes Java disponibilizadas pela literatura não foram adequadas para situações onde há problemas de conectividade e de baixa

potência computacional dos participantes de sistema. Por isso, um mecanismo totalmente baseado em serviços *Web* (WSBCL) foi desenvolvido durante este trabalho, para permitir que o processo de carga também possa ser realizado de forma que tanto o cliente quanto o servidor atuem como agentes ativos na arquitetura.

- *Ambiente seguro de execução com configuração escalável:* Devido ao aumento do número de consumidores proporcionado pelas grades computacionais providas pelo *Grid Anywhere*, foram apresentadas a proposta e a implementação de um ambiente seguro de execução de aplicações Java (*Sam Dog*), que permite a herança e sobreposição de regras que permitem, de forma flexível, definir o que pode ser feito, por quem e em quais contextos.
- *Arquitetura de grade computacional como infraestrutura de um ambiente de PaaS*: Configuração dos componentes implementados para permitir que recursos compartilhados possam ser ofertados como serviço por usuários convencionais.
- *Arquitetura de grade computacional desktop que faz uso de receptores de TV como provedores de recursos*: O *Grid Anywhere* foi utilizado para permitir que os receptores *(set-top boxes)* de sistemas de televisão digital interativa baseados em JavaDTV possam atuar como provedores de recursos.

Com esses resultados é possível concluir que a tese atingiu plenamente seus objetivos. As pesquisas realizadas culminaram em modelos que foram validados por meio das implementações, as quais deverão ser disponibilizadas pelo Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) para a implementação de aplicações reais.

6.3 Trabalhos Futuros

Uma vez que o presente trabalho teve como objetivo resolver os problemas recorrentes em diferentes possíveis arquiteturas de grades computacionais, foram disponibilizados os núcleos centrais do *middleware* e os módulos que são comuns.

No entanto, muitas outras questões ainda precisam ser pesquisadas e desenvolvidas, gerando possibilidades de novos trabalhos de iniciação científica, mestrado, doutorado e pós-doutorado.

- *Permitir que aplicações convencionais já prontas possam utilizar a grade computacional:* O presente trabalho permite que aplicações sejam desenvolvidas de forma a serem executadas no modelo proposto de grade computacional. No entanto, muitas aplicações já se encontram desenvolvidas e alterá-las para o novo modelo envolve um volume bastante grande de trabalho, aceitação dos desenvolvedores e nova distribuição das aplicações. Para dar um grande passo rumo à evolução do *Grid Anywhere*, pretende-se investigar, propor e desenvolver uma plataforma que seja capaz de analisar uma aplicação Java já pronta (diretamente nos arquivos executáveis) e definir quais partes podem ser migradas. Com isso, por meio de instrumentações nos códigos já compilados, será possível permitir que essas aplicações façam uso de recursos remotos de forma transparente.

- *Integração com a plataforma Android*: Quando as soluções do item anterior já estiverem definidas e testadas nas plataformas Java, serão realizados estudos e desenvolvimentos para viabilizar o mesmo para a plataforma *Android*. Isso permitirá que dispositivos móveis (como *smartphones e tablets*), que apresentam uma menor potência computacional, possam fazer uso de aplicações que requisitem uma maior capacidade de *hardware.*

- *Escalonamento e troca de arquivos utilizando P2P*: Para realizar o compartilhamento de recursos entre usuários convencionais é interessante que os recursos possam ser encontrados e utilizados de uma maneira descentralizada. Por isso, a aplicação de algoritmos P2P em uma implementação do módulo de escalonamento torna-se bastante interessante. O mesmo se aplica à transferência de arquivos.

- *Módulo para confiabilidade de resultados retornados:* Nas grades computacionais *desktop*, é comum que uma mesma tarefa seja enviada a mais de um provedor com o objetivo de comparar os resultados para

verificar a veracidade do resultado do processamento. Essa técnica diminui, de forma considerável, a vazão de um sistema como esse. Por isso, uma pesquisa aprofundada para procurar novas formas para obter os mesmos resultados sem precisar replicar o processamento apresentará grandes contribuições para a área.

- *Módulo de replicação de objetos:* Tendo em vista que as aplicações do *Grid Anywhere* podem não ser caracterizadas como *Bag-of-Tasks*, garantir a disponibilidade dos estados dos objetos por meio da replicação das mensagens SOAP é inviável, pois os métodos executados podem realizar ações externas. Sendo assim, é preciso disponibilizar um mecanismo de replicação que atenda a esses requisitos.

- *Mecanismos de contabilização e cobrança de consumo de recursos*: Os módulos que compõem o *Grid Anywhere* podem ser utilizados para compor um ambiente onde os usuários possam fazer uso de recursos que são disponibilizados como serviços por provedores comerciais como, por exemplo, uma operadora de telefonia móvel. Sendo assim, é preciso que o Sesiom seja capaz de contabilizar e cobrar pelo consumo de recursos.

- *Criação de APIs para outras plataformas*: Uma vez que todo o processo de comunicação é feito por meio de XML, é possível que clientes desenvolvidos em outras linguagens de programação (.Net, por exemplo) possam fazer uso de objetos Java remotos. Para isso, é preciso desenvolver as APIs do *Grid Anywhere*, o *gateway* do *Sesiom* (incluindo o ROI) e o servidor ativo do WSBCL utilizando essas plataformas.

- *Interface para monitoramento*: Atualmente as notificações emitidas pelo container não possuem uma forma amigável de apresentação ao usuário. É preciso que uma interface seja desenvolvida para permitir que o acesso a essas informações seja realizado de forma simples.

- *Interface de gerenciamento do Sesiom:* Permitir que o usuário possa, por meio de uma página *Web*, analisar um servidor Sesiom de maneira a observar os *containers* em execução, os objetos hospedados, recursos consumidos, entre outros. Além disso, a interface deve permitir que o

administrador realize ações (finalização de execução, por exemplo) junto ao servidor.

- *Ferramentas de instalação e configuração do ambiente*: Atualmente todo o processo de instalação e configuração dos módulos do *Grid Anywhere* é realizado manualmente. É importante que uma ferramenta auxilie usuários com menor familiaridade com o sistema.
- *Hardware específico para processamento de SOAP over SOAP*: O encapsulamento de uma mensagem SOAP dentro de outro documento desse mesmo protocolo pode consumir um tempo considerável de processamento. É interessante que estudos sejam realizados para analisar a viabilidade de construir uma placa de rede (FPGA) específica para o Sesiom.

6.4 Publicações Originadas ou Relacionadas à Tese

- TEIXEIRA, F. C. ; SANTANA, M. J. ; SANTANA, R. H. C. ; ESTRELLA, J. C.; BRUSCHI, S.M.:"WSBCL: *Web Services Based Classloader*"; *20th International Conference on Collaboration Technologies and Infrastructures*, IEEE *Computer Society*, Paris, 2011.
- TEIXEIRA, F. C. ; SANTANA, M. J. ; SANTANA, R. H. C. ; ESTRELLA, J. C.; BRUSCHI, S.M ."*Sam Dog: A Java Sandbox Using a Cascading Access Control List Approach*";*20th International Conference on Collaboration Technologies and Infrastructures*, IEEE *Computer Society*, Paris, 2011.
- TEIXEIRA, F. C. ; SANTANA, M. J. ; SANTANA, R. H. C. ; ESTRELLA, J. C.: "*Grid Anywhere: An Architecture for Grid Computing Able to Explore the Computational Resources of the Set-top Boxes*". *Third International ICST Conference on Networks for Grid Applications*, Atenas, 2009.
- BATISTA, B. G.; SANTANA, R. H. C.; SANTANA, M. J. ; TEIXEIRA, F. C.: "Simulação de uma Grade Computacional Utilizando Aplicações Interativas sobre os Receptores de Sinais Digitais". XLIII Simpósio Brasileiro de Pesquisa Operacional, Ubatuba, 2011.

6.5 Considerações Finais

O trabalho aqui desenvolvido teve como objetivo analisar o cenário científico atual que envolve a computação em grade, propor novas arquiteturas para esse paradigma e disponibilizar uma implementação real que torne possível a aplicação prática dos conceitos propostos.

Os produtos desenvolvidos irão contribuir com trabalhos científicos por meio da ampliação da potência computacional de grades computacionais *desktop* que poderão fazer uso de TVs digitais e receptores como provedores de recursos. Quando os usuários convencionais são considerados, a possibilidade de utilizar recursos remotos para ampliar a potência computacional local é uma forma de incentivo ao processo de inclusão digital.

As contribuições foram endereçadas tanto ao meio científico quanto ao mercado tecnológico. Os modelos desenvolvidos podem ser utilizados em novas pesquisas para atender a demandas relacionadas à área de sistemas distribuídos. Por sua vez, as implementações dos módulos podem ser aplicadas em questões práticas que envolvam os problemas aqui discutidos.

Por fim, a conclusão desse trabalho abrirá possibilidades para que outros pesquisadores apresentem novas ideias que poderão ser implementadas e validadas por meio de um ambiente em operação.

Referências Bibliográficas

[1] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, vol. 55, pp. 42-47, 2002.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, pp. 200-222, 2001.

[3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Global Grid Forum*, 2002.

[4] R. Buyya, D. Abramson, and S. Venugopal, "The Grid Economy," *Proceedings of the IEEE*, vol. 93, pp. 698-714, 2005.

[5] M. Dantas, *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*.: Editora Axcel Books do Brasil, 2005.

[6] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *IFIP International Conference on Network and Parallel Computing,Springer-Verlag LNCS 3779*, 2005, pp. 2-13.

[7] I. Foster. (2005, Julho) A Globus Primer. Or, Everything You Wanted to Know about Globus, but Were Afraid To Ask. Describing Globus Toolkit 4. [Online]. http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf

[8] W. Cirne and E.S. Neto, "Grids Computacionais: da Computação de Alto Desempenho a Serviços sob Demanda," SBRC, Minicurso.

[9] W. Cirne et al., "Grid Computing for Bag of Tasks," in *Third IFIP Conference on E-Commerce, E-Business and E-Goverment*, São Paulo, 2003, pp. 591-609.

[10] (2006, Dezembro) OURGRID Home-Page. [Online]. http://www.ourgrid.org

[11] D.P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4-10.

[12] (2008, Novembro) Projeto SETI@home. [Online]. http://setiathome.berkeley.edu

[13] (2012, Março) LHC@home. [Online]. http://lhcathome.web.cern.ch/LHCathome/

[14] (2012, Março) Rosseta@home. [Online]. http://boinc.bakerlab.org/

[15] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop*, 2008, pp. 1-10.

[16] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus toolkit for market-oriented cloud computing," *Cloud Computing*, pp. 24-44, 2009.

[17] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architec-tures and Applications*. Springer, 2004.

[18] M.K. Zuffo, "TV Digital Aberta no Brasil – Políticas Estruturais para um Modelo Nacional," Escola Politécnica,USP, São Paulo, 2003.

[19] C.E.C.F. Batista et al., "TVGrid: A Grid Architecture to use the idle resources on a Digital TV network," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 823-828.

[20] R. Costa, F. Brasileiro, and D.M, Lemos Filho, G. Sousa, "OddCI: on-demand distributed computing infrastructure," in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids*

and Supercomputers, 2009.

[21] C. Herpel, "Elementary Stream Management in MPEG-4," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 315-324, 2002.

[22] G. Coulouris, J. Dollimore, and T. Kindberg, *"Distributed Systems: Concepts and Design"*, Fourth Edition ed.: Addison-Wesley, 2005.

[23] R. Aversa, B. Di Martino, and S. Venticinque, "Integration of Mobile Agents technology and Globus for assisted design andautomated development of Grid Services," in *International Conference on Computational Science and Engineering*, 2009, pp. 118-125.

[24] (2006, Março) GT 4: Common Runtime Components. [Online]. http://www.globus.org/toolkit/docs/4.0/common

[25] F. Curbera et al., "Un-raveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, pp. 86-93, 2002.

[26] C. F. Goldfarb and P. Prescod, *The XML Handbook*, second edition ed.: Prentice-Hall, 2000.

[27] J. Siegel, *Corba 3: Fundamentals and Programing*, second edition ed.: Wiley, 2000.

[28] (2012, Janeiro) SOAP Tutorial. [Online]. http://www.w3schools.com/soap/default.asp

[29] J. Robichaux. (2012, Janeiro) Practical Web Services in IBM Lotus Domino 7: What are Web services and why are they important? [Online]. http://www.ibm.com/developerworks/lotus/library/web-services1/

[30] S. Allamaraju, *RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity*.: Yahoo Press, 2010.

[31] C. Pautasso, O Zimmermann, and F. Leymann, "Restful web services vs. big'web services: making the right architectural decision," in *Proceeding of the 17th international conference on World Wide Web*, 2008, pp. 805-814.

[32] S. Tyagi. (2011, Janeiro) RESTful Web Services. [Online]. http://www.oracle.com/technetwork/articles/javase/index-137171.html

[33] S. Pericas-Geertsen and M. Potociar, "JAX-RS: Java™ API for RESTful Web Services," Oracle, API Specification 2011. [Online]. http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-edr-spec/jaxrs-2_0-edr-spec.pdf

[34] A. Genco, *Mobile Agentes: Principle of Operation and Applications*.: WITPress, 2008.

[35] F. Hohl, P. Klar, and J. Baumann, "Efficient code migration for modular mobile agents," in *3rd ECOOP Workshop on Mobile Object Systems*, 1997.

[36] K. Hagab and T. Helmy, *Developing Advanced Web Services through P2P Computing and Autonomous Agents: Trends and Innovations*.: Information Science Reference, 2009.

[37] (2012, Janeiro) The Fundation for Intelligent Physical Agents. [Online]. http://www.fipa.org

[38] (2012, Janeiro) Aglets. [Online]. http://aglets.sourceforge.net/

[39] (2012, Janeiro) Voyager. [Online]. http://www.recursionsw.com/products/voyager/voyager-intro.html

[40] D. Wong et al., "Concordia: An infrastructure for collaborating mobile agents," *Mobile Agents, Springer*, pp. 86-97, 1997.

[41] (2012, Janeiro) Java Agent Development Framework. [Online]. http://jade.tilab.com/

[42] F.C. Teixeira, "Um Escalonador para Grades Computacionais Utilizando Modelos Econômicos," Instituto de Computação - UNICAMP, Campinas, Dissertação de Mestrado 2006.

[43] (2006, Abril) About the Globus Toolkit. [Online]. http://www.globus.org/toolkit/about.html

[44] N. Andrade, L. Costa, G. Germóglio, and W. Cirne, "Peer-to-Peer grid computing with the OurGrid Community," in *SBRC*, Fortaleza, 2005.

[45] N. Constantinescu-Fülöp, "A Desktop Grid Computing Approach for Scientific Computing and Visualization," Norwegian University of Science and Technology, Tese de doutorado 2008.

[46] P. Kacsuk et al., "Sztaki desktop grid: a modular and scalable way of building large computing grids," in *Parallel and Distributed Processing Symposium*, 2007, pp. 1-9.

[47] P. Golle and I. Mironov, "Uncheatable Distributed Computations," *Lecture Notes in Computer Science*, vol. 2020, p. 83, 2001.

[48] (2006, Junho) OGSA, OGSI and GT3. [Online]. http://gdp.globus.org/gt3-tutorial/multiplehtml/ch01s01.html

[49] (2006, Junho) The WS-Resource Framework. [Online]. www.globus.org/wsrf

[50] I. Foster et al. (2006, junho) The WS-Resource Framework. [Online]. http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[51] (2012, Janeiro) Amazon EC2. [Online]. http://aws.amazon.com/pt/ec2/

[52] (2012, Janeiro) Google App Engine. [Online]. http://code.google.com/intl/pt-BR/appengine/

[53] (2012, Janeiro) Sales Force. [Online]. http://www.salesforce.com/br/

[54] (2006, Fevereiro) GridFtp Update January 2002. [Online]. http://www-fp.mcs.anl.gov/dsl/scidac/datagrid/GridFTP%20Overview.pdf

[55] (2006, Fevereiro) Data Management: Key concepts of RLS. [Online]. http://www.globus.org/toolkit/docs/4.0/data/key/rls.html

[56] L.B. Costa et al., "MyGrid – A complete solution for running Bag-of-Tasks Applications," in *SBRC*, 2004.

[57] (2012, Janeiro) Projeto Boinc. [Online]. http://boinc.berkeley.edu/

[58] R. Rigues. (2007) TV Digital é o Terceiro Grande Marco da História da TV Brasileira. [Online]. www.abert.org.br/novosite/clipping/clipping_resultados.cfm?cod=112299

[59] (2008, Setembro) Decreto Presidencial 4901 de 26 de novembro de 2003. [Online]. http://www.planalto.gov.br/CCIVIL/decreto/2003/D4901compilado.htm

[60] J. Fernandes, G. Lemos, and G. Silveira, "Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas," in *Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação*, 2004.

[61] "Modelo de Referência: Sistema Brasileiro de Televisão Digital Terrestre,".

[62] J.D., Berger, T., Lookabaugh, T., Lindbergh, D., Baker, R.L. Gibson, *"Digital Compression for Multimedia: Principles and Standards"*.: Morgan Kaufmann Publishers, 1998.

[63] P.A. Sarginson, "MPEG-2: A Tutorial Introduction to the System Layer," in *IEEE Colloquium on MPEG-2: What it is and what it isn't*, 1995.

[64] J. Watkinson, *The MPEG Handbook*.: Focal Press, 2001.

[65] G. Pelletier, "An overview of the MPEG-2 Transport Stream in Digital Video Broadcasting," Departamento de Ciência da Computação, University of Luleå, Suécia, 2001.

[66] M.S. Richer et al., "The ATSC Digital Television System," *Proceedings of IEEE*, vol. 94, pp. 37-43, 2006.

[67] U. Reimers, "DVB-T: The COFDM-based system for terrestrial television," *Electronics & Communication Engineering Journal*, vol. 9, pp. 28-32, 1997.

[68] M. Uehara, M. Takada, and T. Kuroda, "Transmission Scheme for the Terrestrial ISDB System," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 101-106, 1999.

[69] M. Takada and M. Saito, "Transmission System for ISDB-T," *Proceedings of the IEEE*, vol. 94, pp. 251-256, 2006.

[70] J.B. Santos Jr, I.C. Abrão, E. Barrére, and P.M. Ávila, "Interactive Digital Television Programs:

Formatting, Presentation and Interaction with the Viewer," in *EuroITV*, 2008.

[71] M.C.Q. Farias, M.M. Carvalho, and M.S. Alencar, "Digital Television Broadcasting in Brazil," *IEEE Multimedia*, vol. 15, pp. 64-70, 2008.

[72] L.F.G. Soares, R.F. Rodrigues, and M.F. Moreno, "Ginga-NCL: The Declarative Environment of the Brazilian Digital TV System," *Journal of the Brazilian Computer Society*, vol. 13, 2007.

[73] G. Lemos, L.E.C. Leite, C.E.C.F. Batista, and T.P. Falcão, "Ginga-J: The Procedural Middleware for the Brazilian Digital TV System," *Journal of the Brazilian Computer Society*, vol. 13, 2007.

[74] G. Lemos and L.F.G. Soares, "Interactive Television in Brazil: System Software and the Digital Divide," in *EuroITV*, 2007.

[75] J. Pielsing, "The DVB Multimedia Home Platform (MHP) and Related Specifications," *Proceedings of IEEE*, vol. 94, pp. 237-247, 2006.

[76] A. Das Gupta, "DASE: The Data Applications Software Environment for DTV Data Broadcasting," in *International Conference on Consumer Eletronics*, 1999, pp. 6-7.

[77] C.T. Oliveira, "Um Estudo Sobre os Padrões de Middleware para a Televisão Digital Interativa," CEFET-CE, Monografia de conclusão de curso 2005.

[78] (2012, Abril) Fórum SBTVD. [Online]. http://www.forumsbtvd.org.br/

[79] A.C. Marosi, P. Kacsuk, G. Fedak, and O. Lodygensky, "Sandboxing for Desktop Grids Using Virtualization," in *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2010.

[80] F. Cappello et al., "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid," *Future Generation Computer Systems*, 2005.

[81] A., Calder, B., Elbert, S., Bhatia, K. Chien, "Entropia: architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, 2003.

[82] M. Bartoletti, G. Costa, and R. Zunino, "Jalapa: Securing Java with Local Policies," *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2009.

[83] (2012, Fevereiro) Java (TM) Security Overview. [Online]. http://docs.oracle.com/javase/6/docs/technotes/guides/security/overview/jsoverview.html

[84] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement,Simulation, and Modeling*. New York: Wiley- Interscience, 1991.

[85] G. Travis. (2012, Fevereiro) Understanding the Java ClassLoader. [Online]. http://www.ibm.com/developerworks/java/tutorials/j-classloader

[86] H.Q. Mahmoud. (2012, Fevereiro) Understanding the Network Class Loaders. [Online]. 2012

[87] R. Kapitza, H. Schmidt, U. Bartlang, and F.J. Hauck, "A generic infrastructure for decentralised dynamic loading of platform-specific code," in *Proceedings of the 7th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, 2007.

[88] D. Michie, D. J. Spiegelhalter, and C.C. Taylor, *Machine Learning, Neural, and Statistical Classification.* New York: Ellis Horwood, 1994.

[89] M. Hall et al., "The Weka Data Mining Software: An Update," *ACM SIGKDD Explorations Newsletter*, 2009.

[90] A.S. Tanembaum, *"Redes de Computadores"*.: Elsevier, 2003.

[91] U.H. Reimers, "DVB – The Family of International Standards for Digital Video Broadcasting," vol. 94, pp. 173-182, 2006.

[92] (2008) Digital Video Broadcasting Project. [Online]. www.dvb.org

[93] Z. Xiong, Y. Yang, X. Zhang, M. Zeng, and L. Liu, "A Grid Resource Discovery Model Using P2P

Technology," in *IIHMSP'08 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2008, pp. 1553-1556.

[94] B Hudzia, T.N. Ellahi, L. McDermott, and T. Kechadi, "A Java Based Architecture of P2P-Grid Middleware," Arxiv preprint cs.DC/0608113 2006.

[95] (2006, Junho) Towards Open Grid Services Architecture. [Online]. http://www.globus.org/ogsa

[96] (2006, Fevereiro) Data Management: Key Concepts. [Online]. http://www.globus.org/toolkit/docs/4.0/data/key/index.html#id2488793

[97] R. K. Madduri, C. S. Hood, and Allcock W. E., "Reliable File Transfer in Grid Environments," in *27th Annual IEEE Conference on Local Computers Networks*, 2002.

[98] (2006, Fevereiro) The OGSA-DAI Project. [Online]. http://www.ogsadai.org.uk

[99] (http://www.globus.org/toolkit/docs/4.0/info/key-index.html#id2770079, Fevereiro) Information Services: Key Concepts. [Online]. 2006

[100] (2006, Fevereiro) GT 4 Release Notes: Index Service. [Online]. http://www.globus.org/toolkit/docs/4.0/info/index/WS_MDS_Index_Release_Notes.html#overview

[101] R.F. Lopes, "Mag: uma grade computacional baseada em agentes móveis," Universidade Federal do Maranhão, Dissertação de mestrado 2006.

[102] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G.C. Bezerra, "InteGrade: object-oriented Grid middleware leveraging the idle computing power of desktop machines," *Concurrency and Computation: Practice and Experience*, vol. 16, pp. 449-459, 2004.