UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

SEBASTIÃO EMIDIO ALVES FILHO

GRINGA: UM MIDDLEWARE ORIENTADO A SERVIÇOS PARA GRIDS DE DISPOSITIVOS EM REDES RESIDENCIAIS CENTRADAS EM TV DIGITAL INTERATIVA

NATAL – RN

2011

SEBASTIÃO EMIDIO ALVES FILHO

GRINGA: UM MIDDLEWARE ORIENTADO A SERVIÇOS PARA GRIDS DE DISPOSITIVOS EM REDES RESIDENCIAIS CENTRADAS EM TV DIGITAL INTERATIVA

Dissertação apresentada ao Mestrado de Ciência da Computação – associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Profª. Drª. Cláudia Maria Fernandes Araújo Ribeiro – UERN.

NATAL – RN

2011

Catalogação da Publicação na Fonte.

Alves Filho, Sebastião Emidio
Gringa: um middleware orientado a serviços para grids de dispositivos em redes residenciais centradas em TV Digital Interativa . / Sebastião Emidio Alves Filho . – Natal, RN, 2011.

109 f
.

Orientador(a): Prof. Dr. Cláudia Maria Fernandes Araújo Ribeiro
Dissertação (Mestre) Universidade do Estado do Rio Grande do Norte. Departamento de Ciência da computação.

1. Ciência da computação - Dissertação. 2. TV Digital Interativa. 3. Redes Locais Residenciais. I. Ribeiro,Cláudia Maria Fernandes Araújo . II.Universidade do Estado do Rio Grande do Norte. III.Título.

UERN/BC CDD 004

Bibliotecário: Sebastião Lopes Galvão Neto CRB 15 / 486

Sebastião Emidio Alves Filho

Gringa: um middleware orientado a serviços para grids de dispositivos em redes residenciais centradas em TV Digital Interativa

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade do Estado do Rio Grande do Norte (UERN) em Associação Ampla com a Universidade Federal Rural do Semi-Árido (UFERSA).

Sebastião Emidio Alves Filho

Banca Examinadora:

Profa. Cláudia Maria Fernandes Araújo Ribeiro, D.Sc. (Orientadora)
Universidade do Estado do Rio Grandes do Norte – UERN

Prof. Aquiles Medeiros Filgueira Burlamaqui, D.Sc. (Co-Orientador)
Universidade Federal do Rio Grande do Norte – UFRN

Prof. Gledson Elias da Silveira, D.Sc.
Universidade Federal da Paraíba – UFPB

Prof. Rommel Wladimir de Lima, D.Sc.
Universidade do Estado do Rio Grande do Norte – UERN

Natal-RN, 09 de agosto de 2011

À minha família, razão da minha vida.

Agradecimentos

A Deus em primeiro lugar, por ter me proporcionado tudo o que tenho de tal forma que não tenho mais o que pedir, apenas agradecer. À minha família amada, Ivone e Sebastian, por ter me apoiado e compreendido pelos períodos ausentes. À minha mãe que se esforçou por toda vida para me dar esta oportunidade de estudar e ser alguém. Aos meus amigos do trabalho, do mestrado, da vida, obrigado por existirem e tornar este mundo um lugar melhor de se viver, por transformarem os momentos de dificuldade em tranquilidade e de sofrimento em companheirismo. Seria injusto citar nomes porque neste momento o sentimento no coração é maior do que a memória.

Por fim, um agradecimento especial aos que contribuíram diretamente com a realização deste feito. À Cláudia, minha orientadora e incentivadora, grande responsável por este trabalho chegar ao fim. A Aquiles, meu co-orientador, amigo e irmão de coração, e também co-idealizador dessa proposta. Ao Ministério da Cultura pelo financiamento do projeto que viabilizou a compra dos equipamentos usados nos experimentos e dos bolsistas, Anderson, Carlos Fran, Davi e Marlos que abraçaram o projeto e fizeram ele acontecer.

A todos vocês, não há palavras para descrever o que sinto, obrigado.

O grande segredo para a plenitude é muito simples:compartilhar.

Sócrates

Resumo

A miniaturização de *chips* e outros componentes tem tornando possível a construção de vários tipos de dispositivos eletrônicos com poder de processamento, tais como *smartphone*, computadores portáteis e *Settop box* de TV Digital Interativa. Já não é mais tão raro encontrar esses dispositivos formando, junto aos computadores de mesa, redes locais residenciais (Home Area Networks - HAN). Levando-se em consideração que os recursos de dispositivos computacionais geralmente ficam ociosos a maior parte do tempo, esse cenário favorece a utilização de aplicações distribuídas nas quais os dispositivos compartilham recursos que são inexistentes ou escassos nos outros equipamentos, como em *grids* computacionais.

Dessa forma, esta dissertação apresenta o Gringa, um *middleware* para o desenvolvimento de aplicações para TV Digital Interativa, baseadas no *middleware* Ginga, que utilizam *grids* orientados a serviços formados por dispositivos de uma HAN. O uso desse modelo permitiu aumentar o grau de abstração na construção de aplicações no contexto de HAN, uma vez que cada equipamento pode possuir serviços diferentes com disponibilidades diferentes e ainda utilizar protocolos de comunicação diversos para torná-los acessíveis. Neste documento serão mostrados detalhes sobre sua arquitetura e tecnologias utilizadas, além de exemplos de aplicativos Ginga para TVDI que interagem com outros dispositivos através do Grid.

Palavras-chave: Grids; Orientação a Serviços; TV Digital Interativa; Middleware; Redes Locais Residenciais

Abstract

The miniaturization of chips and other eletronic components is making possible to build several types of processing capable devices such as smartphone, portable computer and Interactive Digital TV Settop box. It is no longer so rare to find Home Area Network (HAN) formed by these devices and desktop computers. Like computing devices stay often idle for most of the time, this scenario favors the use of distributed applications where devices share resources with others in which that it's absent or scarce, like in computing grids.

Thus, this work presents the Gringa, a middleware for application development to Ginga-based Interactive Digital TV that uses service-oriented grids formed by devices connected to a HAN. This model allows increase the abstraction of applications development in the HAN context, since each equipment may have different service with different availability and even use different communication protocols to make them available. This document will show details of its architecture and used technologies, and examples of Ginga applications that interact with other devices via Grid.

Palavras-chave: Grids; Service-oriented paradigm; Interactive Digital TV; Middleware; Home Area Networks

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ATSC	*Advanced Television Systems Committee*
API	*Application Programming Interface*
ARIB	*Association of Radio Industries and Businesses*
CSS	*Cascading Style Sheets*
DVB	*Digital Video Broadcasting*
DASE	*Digital Television Application Software Environment*
HAN	*Home Area Networks*
HDTV	*High Definition Television*
HTTP	*HyperText Transfer Protocol*
ISDB	*Integrated Services Digital Broadcasting*
JPEG	*Joint Photographic Experts Group*
MFlops	*Mega Floating point Operations Per Second*
MHP	*Multimedia Home Platform*
MPEG	*Moving Picture Experts Group*
OSGA	*Open Service Grid Architecture*
RPC	*Remote Procedure Call*
SBTVD	Sistema Brasileiro de TV digital Terrestre
SDTV	*Standard Definition Television*
SMS	*Short Message Service*
SOG	*Service Oriented Grid*
STB	*Settop Box*
TVDI	Televisao Digital Interativa
UML	*Unified Modelling Language*
XML	*Extensible markup language*

Capítulo 1

Introdução

1.1 Contexto

A Televisão se tornou um dos meios de comunicação mais importantes e expressivos no Brasil e no mundo. Segundo dados da Pesquisa Nacional por Amostra de Domicílios divulgados pelo Instituto Brasileiro de Geografia e Estatística(IBGE, 2009), há a estimativa que a TV esteja presente em 95,7% dos domicílios brasileiros, sendo o meio de comunicação de massa com maior popularidade no país.

Através do Decreto Presidencial Nº 4.901, de 26 de novembro de 2003, foi instituído oficialmente o Sistema Brasileiro de Televisão Digital(SBTVD), que tem entre outros objetivos: aprimorar a qualidade de áudio, vídeo e serviços; promover inclusão social e diversidade cultural visando a democratização da informação; propiciar a criação de rede universal de educação à distância; além de contribuir para a convergência tecnológica e empresarial dos serviços de comunicações.

No que se refere às mudanças do ponto de vista da tecnologia, como cita (PAGANI, 2003), há uma mudança na forma como as imagens são codificadas, moduladas e transmitidas. Ao invés de um sinal analógico de vídeo composto através de ondas de rádio (para a transmissão terrestre) que ocupa todo o canal de comunicação, na TV digital antes de serem transmitidos os vídeos passam por um processo de digitalização na qual os dados são transformados em uma sequência de bits, passam então por um processo de compressão e só após isso são transmitidos.

Quando chega ao espectador o vídeo sofre o processo inverso, isto é, sofre um processo de descompressão e, caso o televisor não tenha suporte a vídeo digital,

é transformado novamente em sinal analógico para ser exibido. Como cita (FERNANDES et al., 2004), os televisores comuns não são preparados para realizar esse processo, o que faz com que haja a necessidade de um equipamento específico, os chamados *Settop Boxes* (STB) ou Unidades de Recepção e Decodificação (URD), que em alguns aparelhos podem vir embarcados.

Este processo de digitalização e compressão dos dados faz com que uma maior quantidade de informação possa ser transmitida. Dessa forma é possível aumentar a diversidade da programação com uma combinação de vídeos, aumentar sua qualidade e até mesmo enviar um conjunto de dados junto com os vídeos. Estes dados podem trazer não somente informações sobre a transmissão ou a emissora mas também aplicações que podem ser executadas nos próprios STB's. Isso faz, como cita (GATIS, 2006), "[...] com que as pessoas deixem de ser meros espectadores para se tornarem também usuários de aplicativos multimídia interativos. Estes aplicativos virão com o conteúdo multimídia e irão permitir que o usuário possa interagir ativamente com o conteúdo exibido[...]". Por esta razão é que comumente chamamos este tipo de cenário de Televisão Digital Interativa (TVDI) e o suporte à essa interatividade é definido por um *middleware*, que no caso do SBTVD é chamado Ginga, que será mostrado posteriormente.

Levando em consideração que a implantação do padrão do SBTVD aconteceu em 2006, através do Decreto Presidencial nº 5.820 de 29 de junho de 2006, a primeira transmissão oficial ocorreu no fim de 2007 e que o prazo para que sejam encerradas as transmissões de TV analógica se encerra somente em 2016, podemos considerar que a implantação de transmissores, o desenvolvimento de aplicações e a aquisição dos STB's por parte dos telespectadores vem acontecendo de forma lenta.

1.2 Motivação

No Brasil, as iniciativas e incentivos para a inclusão digital tem trazido resultados expressivos nos últimos anos. Pesquisas realizada pela *International Data Corporation*(IDC), mostram que houve um crescimento de 13,6% nas vendas de computadores *desktops* (MADWAY, 2011) e 33,3% no número de *notebooks* (COMPUTERWORLD, 2011) no ano de 2010.

Já com relação à telefonia móvel, a Agência Nacional de Telecomunicações (ANATEL) relatou um crescimento de 16,66% no número de novas linhas (G1,

2011), consolidando o curioso fato de que há mais linhas de telefonia móvel que o número de habitantes do país. Aliando-se isso ao fato de que a miniaturização de *chips* e outros componentes eletrônicos tem tornado possível a construção e a popularização de vários tipos de dispositivos com capacidade de execução de software complexos e maior conectividade, é comum vermos aparelhos onde é possível baixar e instalar programas e acessar a Internet, os conhecidos *smartphones*. Esses e outros dispositivos tem a capacidade de não só interagir com receptores de TV Digital com suporte a interatividade, como também ter incorporada a própria TV como uma das suas funcionalidades. Essa possibilidade de convergência tecnológica e interatividade na TV abre um leque de oportunidades para o desenvolvimento de aplicações interativas.

Entretanto, foi visto na prática que a diferença de capacidade de processamento, por exemplo, entre um computador de mesa, que hoje conta com tecnologias como os multinúcleos (GEER, 2005), e dispositivos com capacidade mais limitada, como os STB's e *smartphones* é muito grande. Em testes que são mostrados na Tabela 5.2 no capítulo 5, um *smartphone* teve um resultado de desempenho mais de 76 vezes e o STB mais de 523 vezes inferior ao *desktop*, além de contarem com uma capacidade de memória RAM 32 vezes menor e uma capacidade de armazenamento persistente mais de 62 vezes inferior.

Some-se ainda o fato de que, enquanto a adoção e a compra de equipamentos para TV Digital tem se dado de forma lenta, a troca de equipamentos computacionais vem acontecendo de forma mais rápida a cada dia. Por exemplo, uma pesquisa divulgada no Jornal Correio do Povo (NUNES, 2011) mostrou que jovens entre 14 e 18 anos trocam de telefone celular a cada 14 meses, a procura de novos recursos. Isto pode aumentar ainda mais essa diferença de desempenho entre o STB e os demais dispositivos computacionais em um ambiente doméstico.

E essa enorme diferença pode implicar diretamente nas características das aplicações interativas para estes equipamentos, que ficariam limitadas pelo hardware. No trabalho de (GHISI et al., 2010), por exemplo, é mostrada uma aplicação que tenta enriquecer informações sobre um vídeo que está sendo exibido na TV, trazendo conteúdos relevantes através de um serviço integrado à Wikipédia. Para implementar a aplicação, o autor usou um computador de mesa como *gateway*, que era responsável por fazer uma busca em páginas web que estivessem relacionadas com as informações presentes no Guia de Programação Eletrônica (EPG) da Televisão sobre o vídeo e extrair essas informações, para só então enviá-las para a TV.

Um cenário como este, onde uma aplicação de TVDI interage com um computador de mesa, pode se repetir em outras aplicações já que não é mais tão raro encontrar esses dispositivos formando, junto a outros dispositivos com capacidade de conectividade em rede, as chamadas redes locais residenciais (Home Area Networks - HAN). Levando-se em consideração que os recursos de dispositivos computacionais geralmente ficam ociosos a maior parte do tempo, esse cenário de convergência favorece a implementação de aplicações distribuídas nas quais os dispositivos compartilham recursos que são inexistentes ou escassos nos outros equipamentos.

1.3 Problematização

Computadores pessoais, *smarphones* e STB's geralmente possuem hardware e plataformas de execução distintas. Para construir um sistema distribuído que envolva a comunicação entre elementos desse tipo, além de outros que poderiam ter capacidade de processamento pertencentes a uma HAN, não seria recomendável usar uma biblioteca de comunicação de baixo nível, como os *sockets* ou RPC (TAY; ANANDA, 1990), devido à complexidade que traria na manipulação de portas e gerenciamento das conexões.

Já o modelo de objetos distribuídos pode trazer um nível de abstração bem mais alto. Entretanto, as principais soluções de *middleware* para isso também tem suas limitações. O CORBA, conforme mostra (JAGANNADHAM et al., 2007), apresenta geralmente um desempenho muito baixo em comparação com outras tecnologias de programação distribuída. Já o Java RMI é uma tecnologia específica para a linguagem Java, o que dificultaria a interação com aplicações em outras linguagens, além dos problemas relativos a *firewall* devido às portas utilizadas na comunicação dessas duas soluções.

Por estas razões, o paradigma de orientação a serviços seria o mais recomendado pois tem como principais vantagens proporcionar um alto nível de abstração e padronização no modo como é feita a definição dos serviços e a comunicação entre quem disponibiliza um recurso e quem o requisita. Na comunicação, os serviços tem definidos um contrato com uma descrição abstrata de suas capacidades (geralmente descrito em XML(GOLDFARB; PRESCOD, 1998)) que é acessado através de uma interface remota orientada-a-mensagem, que utiliza algum tipo de descrição bem definida (VöLTER et al., 2005). Contudo, o modelo padrão de registro de serviços

encontradas nas principais soluções de *middleware* é de associar um serviço a um único provedor, o que não seria adequado ao cenário de aplicação proposto, uma vez que um mesmo serviço poderia estar disponível em mais de um equipamento.

Dentre os modelos de implementação de sistemas distribuídos existentes, os *grids* orientados a serviços vem se mostrando adequados para tratar cenários heterogêneos como estes. A própria conceituação de *grids* dada por (FOSTER; KESSELMAN, 1999) já prevê uma situação onde seus integrantes não possuem equipamentos homogêneos e na qual não é relevante saber quem está disponibilizando o recurso, mas sim como os recursos podem ser acessados. O *grid* também pode implementar mecanismos de tolerância a falhas e replicação, assim como definir uma política específica de escalonamento levando em consideração aspectos específicos, como por exemplo maximização da autonomia de energia de equipamentos que funcionem com bateria tais como os aparelhos celulares e *notebooks*.

Com isso, para abstrair a complexidade da formação de *grids* orientados a serviços em uma rede residencial heterogênea e de implementação de programas interativos para TV que utilizem esse *grid* se faz necessário o uso de um *middleware* específico com este fim.

1.4 Objetivos

A partir desta análise, podemos definir os objetivos desta dissertação como se segue.

1.4.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver um middleware, denominado **Gringa**, para simplificar a construção de aplicações interativas para TV Digital utilizando o paradigma de *grids* orientados a serviços. Esses *grids* são formados por dispositivos de uma HAN e acessados por um *Settop box* equipado com o *middleware* Ginga do Sistema Brasileiro de TV Digital.

A arquitetura do Gringa foi projetada de forma a dar suporte ao maior número de dispositivos diferentes, com conexões distintas e implementação de serviços em plataformas distintas. Além disso, por também lidar com dispositivos com autonomia

de energia limitada, também foram levados em consideração aspectos diretamente relacionados à este tema, especialmente ao tratar do escalonamento e modelo de solicitação de tarefas.

Para a validação do *middleware* foram usados dois estudos de caso com aplicações que requerem alto uso de CPU, com a finalidade de testar o desempenho geral dos *grids*: um aplicativo de multiplicação de matrizes e outro de conversão de vídeo.

1.4.2 Objetivos Específicos

Também são objetivos desse trabalho:

- Promover discussões sobre a utilização de programas interativos de TV para promover a convergência digital utilizando o Gringa como base de seu desenvolvimento;
- Abstrair toda a complexidade de formação do *grid*, deixando para o desenvolvedor apenas a responsabilidade de construir softwares paralelos/distribuídos que utilizem a sua infraestrutura;
- Tornar o Gringa o mais flexível possível, de forma que ele possa servir de ambiente de testes para desenvolvedores de novos mecanismos de escalonamento de serviços, modelos de comunicação, tolerância a falhas etc.

1.5 Metodologia

A metodologia científica utilizada nesta dissertação pode ser resumida nos itens a seguir:

- Revisão Bibliográfica

  Foi realizada uma revisão bibliográfica sobre os principais temas envolvidos nesta dissertação, que são os *gris* orientados a serviços e a interatividade em TV Digital. Como o cenário de aplicação do *middleware* Gringa é em um ambiente residencial, a pesquisa focou no estudo de arquiteturas de plataformas para computação em *grid* que usam o paradigma de orientação a serviços e como estas se relacionam com os diversos tipos de dispositivos presentes em

uma residência. Além disso, também foi necessário investigar as características do Sistema Brasileiro de TV Digital e os modelos de programação para o desenvolvimento de aplicações interativas.

- Definição de uma arquitetura para o *middleware*

  Existem diversas soluções de *middleware* para *grids* computacionais orientados a serviços. Cada solução foi projetada para atender as demandas em um tipo de cenário para o qual o *middleware* seria utilizado. Como espera-se que o Gringa seja usado em um ambiente de rede local residencial, no qual podem haver nós de diversas capacidades de processamento, memória, conectividade, entre outros, foi necessário definir uma arquitetura de alto nível, independente de linguagem e paradigma de programação, que tornasse possível sua implementação nas diversas plataformas existentes nesse ambiente, especialmente a TV Digital Interativa.

- Criação de uma implementação de referência e execução de testes

  Para validar a arquitetura proposta, foram feitas implementações para diferentes plataformas de acordo com a proposta do Gringa. A implementação dos nós do *grid* e das aplicações clientes utilizaram tecnologias compatíveis com o padrão Ginga, do Sistema Brasileiro de TV Digital. Foram também realizados testes de desempenho para verificar a eficiência da distribuição de tarefas entre os nós e a aceleração do desempenho nas aplicações paralelas em relação à execução sequencial.

1.6 Estrutura do Documento

Para melhor entendimento do trabalho, esta dissertação está dividida da seguinte forma: o capítulo 2 contém a revisão bibliográfica onde constam os conceitos mais relevantes para o entendimento do trabalho, consistindo em uma seção sobre conectividade e interatividade em TV Digital Interativa, uma seção sobre *grids* orientados a serviços e outra sobre trabalhos relacionados que envolvem *grids* e TV Digital; no capítulo 3 é apresentada a arquitetura do *middleware* Gringa e a descrição de seus componentes; no capítulo 4 é abordada a maneira como foi feita a implementação de referência do Gringa; no capítulo 5 é feita a descrição dos testes práticos realizados para aferição de desempenho do *middleware*;e por fim, no capítulo 6 são feitas as considerações finais, mostrando possibilidades de trabalhos futuros.

Capítulo 2

Revisão de Literatura sobre Grids Orientados a Serviços e Conectividade em TV Digital Interativa

Este capítulo tem por objetivo fazer uma revisão bibliográfica sobre os principais temas abordados nessa dissertação, mostrando os trabalhos relevantes desenvolvidos nessa área.

2.1 Conectividade em TV Digital Interativa

No contexto da TVDI, o *middleware* serve para fazer uma intermediação entre a aplicação e a variedade de plataformas existentes. Para a TVDI foram criadas várias soluções de *middleware* diferentes de acordo com o sistema adotado. Baseado em (LEITE et al., 2005), podemos montar a Tabela 2.1 com um resumo das principais características de cada uma das soluções de *middleware* para TVDI terrestre mais utilizadas no mundo. Podemos notar que neles temos dois conjuntos de aplicações com perfis diferentes: declarativas e procedurais.

Segundo (LEITE et al., 2005) "As aplicações declarativas são, na realidade, documentos hipermídia a serem apresentados no terminal de acesso, com intuito principalmente informativo, e que possibilitam o encadeamento de diversos docu-

Tabela 2.1: Principais soluções de *middleware* para TV Digital

Middleware	Sistema de origem	Principais características
Multimedia Home Platform (MHP)	*Digital Video Broadcasting* (DVB) - Europeu	Utiliza uma extensão das linguagens HTML e XML, a DVB-HTML, para aplicações declarativas e uma API Java, a DVB-J para aplicações procedurais.
DTV Application Software Environment (DASE)	*Advanced Television Systems Committee* (ATSC) - Americano	Suporta aplicações declarativas em XHTML com suporte a folhas de estilo CSS , scripts ECMAScript e suporte a DOM. Para as procedurais se baseiam na API Java TV.
Association of Radio Industries and Businesses (ARIB)	*Integrated Services Digital Broadcasting* (ISDB) - Japonês	Define uma linguagem para aplicações declarativas, a *Broadcast Markup Language* (BML) e utiliza uma especificação baseada na API DVB-J do MHP para aplicações procedurais.

mentos". Já as procedurais "correspondem a programas de computador formados por comandos ou instruções a serem interpretados ou executados no Terminal de Acesso, para realizarem algum tipo de processamento de dados e interface com o usuário".

Segundo (SOUZA FILHO et al., 2007), o principal requisito da TV Digital brasileira era a convergência digital. Dessa forma, o *middleware* teria que fornecer o suporte para que a mídia fosse exibida tanto em HDTV quanto em SDTV para aparelhos fixos e móveis, como os celulares, além de dar suporte ao maior número de tipos de conexões de redes diferentes, como *Bluetooth* e *Ethernet*, por exemplo. O *middleware* do padrão brasileiro, Ginga, foi criado com esse propósito. Como fala (SOARES et al., 2007), ele é composto por três subsistemas principais: o Ginga-NCL para aplicações declarativas, o Ginga-J para aplicações procedurais e o Ginga-CommonCore (Ginga-CC) que é responsável pela decodificação e apresentação de tipos de conteúdos comuns, como JPEG e MPEG, para aplicações declarativas e procedurais (conforme ilustrado na Figura 2.1).

O Ginga-NCL manipula uma linguagem derivada do XML, a *Nested Context Language-NCL* (SOARES et al., 2007), também dando suporte a folhas de estilo CSS (CELIK et al., 2009) e à execução de scripts em ECMAScript (ECMA, 2011)

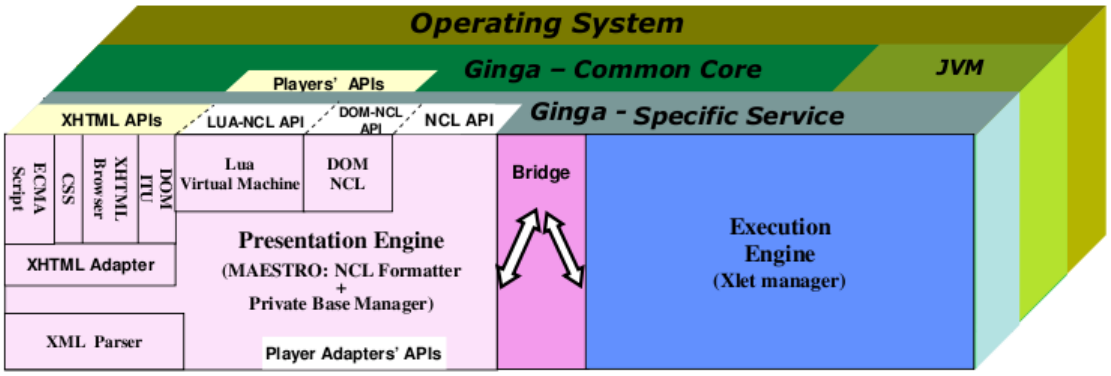

Figura 2.1: Arquitetura do *middleware* Ginga. Fonte: (SOARES et al., 2007)

e na linguagem de programação LUA (IERUSALIMSCHY et al., 1995), enquanto o Ginga-J define um conjunto de API's da linguagem de programação Java.

Conforme explica (MANHÃES et al., 2005), todo sistema de TVDI utiliza o chamado canal de interatividade para o envio e a comunicação das aplicações interativas, sejam elas declarativas e procedurais, o que para o SBTVD está especificado em (ABNT, 2008). Como mostra a Figura 2.2, ele é composto de dois tipos de comunicação:

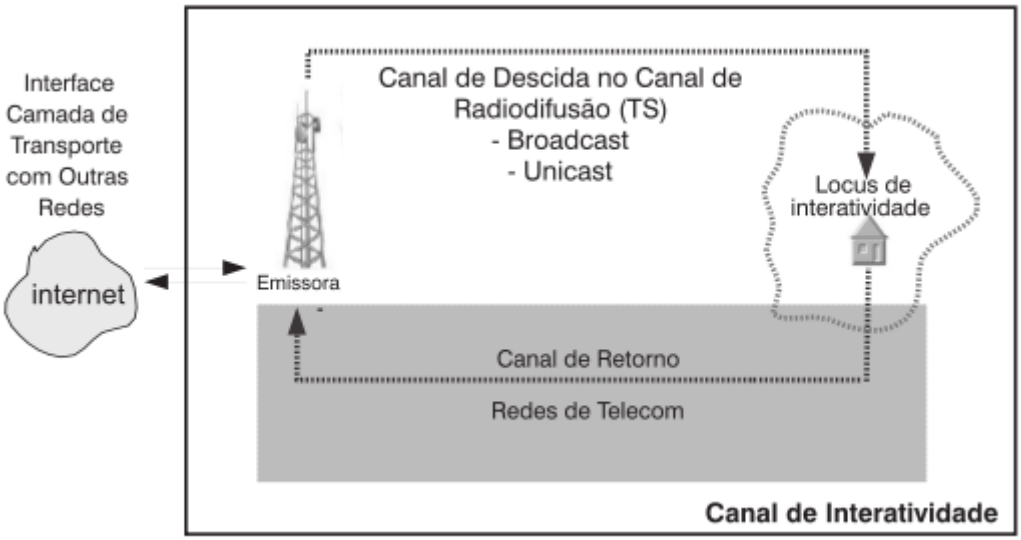

Figura 2.2: Diagrama simplificado do canal de interatividade. Fonte: (MANHÃES et al., 2005)

- Canal de descida: constitui a comunicação entre a emissora e o usuário, na qual dados e aplicações são repassadas através do canal de radiodifusão em *broadcast* ou *unicast*;

- Canal de retorno: constitui a comunicação entre o usuário e emissora, podendo ser composto por qualquer combinação de tecnologias de redes de acesso de telecomunicações.

Dependendo do tipo de comunicação que uma aplicação interativa tenha com a emissora (ou desenvolvedora do aplicativo) através do canal de retorno, o trabalho de (FERNANDES et al., 2004) classifica seu nível de interatividade como:

- Local: neste nível diz-se que as informações enviadas pela emissora para o usuário são de caráter geral e suficiente, não necessitando do canal de retorno. Tem-se como exemplos o guia eletrônico de programação (EPG) ou informações sobre *clips* musicais, como em (NASCIMENTO, 2008);

- Intermitente: é a interatividade na qual existe a comunicação de volta do usuário para a emissora/desenvolvedora, só que de modo unidirecional, isto é, o usuário envia um dado à emissora esporadicamente sem esperar desta uma resposta. Um exemplo seria uma aplicação do tipo *quiz*, como a de (JUCÁ; LUCENA, 2005);

- Permanente: constitui-se numa interatividade na qual há uma comunicação bidirecional entre o usuário e a emissora/desenvolvedora, sendo a troca de mensagens mais frequente. Isto seria recomendado para aplicações mais complexas como um aplicativo para serviços bancários como mostrado em (TOTVS, 2011).

Contudo, além desses modelos de interatividade nos quais a aplicação se comunica através do canal de retorno com a emissora/desenvolvedora do aplicativo, existem também situações em que são construídas aplicações distribuídas diversas utilizando tecnologias como *Socket* e RPC. O trabalho de (VIANA et al., 2009), por exemplo, mostra um modelo de convergência na qual um STB com Ginga ou MHP poderia se conectar a um dispositivo de controle residencial, usando uma rede IPTV (XIAO et al., 2007), para acessar sensores de monitoramento de temperatura de ambientes, por exemplo. Essas aplicações usam recursos da API do próprio *middleware*, que também estão presentes na especificação do Ginga.

2.1.1 Conectividade com a API Ginga-NCL

Conforme explicitado em (SOARES et al., 2007), a linguagem NCL é diferente das demais linguagens declarativas para TVDI, que são baseadas em XHTML, pois ela não se preocupa somente com a apresentação das mídias em si, mas sim como objetos de mídia estão estruturados e relacionados no tempo e no espaço, o que nos demais padrões era feito pelas linguagens de *script*. No caso do NCL, a linguagem LUA (e um conjunto de módulos chamados pelos desenvolvedores de NCLua) foi escolhida como linguagem de *script*, o que proporcionou vários recursos adicionais em relação às outras, como o suporte a orientação a objetos, co-rotinas e o acesso ao canal de interatividade.

Esse acesso ao canal de interatividade ocorre através da classe de eventos *tcp* pertencente ao módulo de eventos, um mecanismo bastante simples e com baixa abstração. O módulo de eventos possibilita que o formatador NCL, responsável pelo fluxo de execução das mídias em uma aplicação declarativa, e uma aplicação NCLua se comuniquem de maneira assíncrona. Como cita a norma com o padrão NCL, descrita em (ABNT, 2009a) e (ABNT, 2009b), "Após a iniciação, qualquer ação tomada pela aplicação é somente em resposta a um evento enviado pelo formatador NCL à função *handler*[...]. Um script Lua também é capaz de gerar eventos, ditos espontâneos, com uma chamada à função *event.post(evt)*".

Sendo assim, todas as conexões acontecem através da chamada ou do tratamento de um evento da classe *tcp*, cujos parâmetros definem que tipo de ação será realizada ou qual o resultado. A Tabela 2.2 demonstra os campos que são usados em cada tipo de chamada e resposta para estabelecer uma conexão, enviar/receber dados e encerrar uma conexão. A coluna *origem* indica se o evento foi provocado pela aplicação (postado) ou recebido por ela (registrado).

Deve-se notar que não há funções para recebimento de conexões, que poderia ser provida por uma biblioteca de *sockets* que não está presente na especificação, e que os dados enviados e recebidos são do tipo string. Isso torna mais complexo a implementação de programas interativos NCLua distribuídos, já que todo o controle de conexões e envio/recebimento dos dados tem que ser tratado diretamente pelo desenvolvedor.

Tabela 2.2: Eventos da classe *tcp* do NCLua.

Origem do evento	**Formato**	**Descrição**
Postado	evt = { class = 'tcp', type = 'connect', host = addr, port = number, [timeout = number] }	Tentativa de estabelecimento de conexão no endereço e porta especificados (com timeout opcional)
Registrado	evt = { class = 'tcp', type = 'connect', host = addr, port = number, connection = identifier, error = err_msg }	Resposta da tentativa de conexão, onde os campos *connection* e *error* são mutuamente exclusivos, isto é, caso haja erro o campo *error* é recebido com a descrição do erro, caso contrário o campo *connection* estará presente e armazenará um identificador para aquela conexão estabelecida.
Postado	evt = { class = 'tcp', type = 'data', connection = identifier, value = string, [timeout = number] }	Tentativa de envio de dados descritos em *value* pela conexão identificada por *connection* (com *timeout* opcional).
Registrado	evt = { class = 'tcp', type = 'data', connection = identifier, value = string, error = msg }	Recebimento de dados da conexão *tcp* identificada por *connection*. Os campos *value* e *error* são mutuamente exclusivos.
Postado	evt = class='tcp', type='disconnect', connection=identifier	Solicitação de desconexão da conexão identificada por *connection*.

2.1.2 Conectividade com a API Ginga-J

A primeira proposta para o Ginga-J (ver Figura 2.3) foi apresentada em (SOUZA FILHO et al., 2007). Nela é descrita que podem existir dois tipos de aplicações: as nativas da unidade receptora, que podem usar funcionalidades fora do padrão; e aquelas distribuídas em *broadcast* pelas emissoras, os chamados Xlets, que devem seguir estritamente a API Ginga-J padrão.

A norma Ginga-J, especificada em (ABNT, 2010a), é formada por diversas API's já existentes utilizadas em outros padrões de TV digital mescladas com API's que são

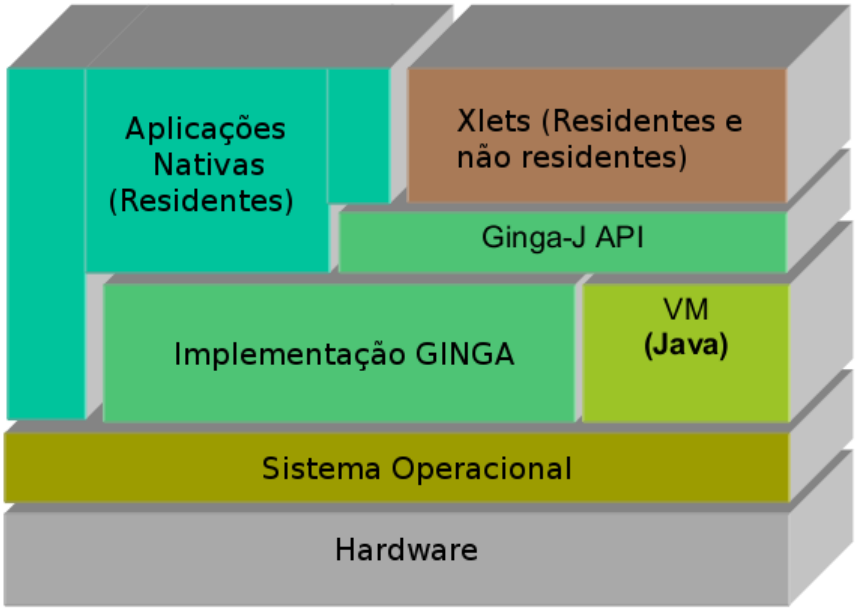

Figura 2.3: Arquitetura e ambiente de execução Ginga-J. Adaptado de: (SOUZA FILHO et al., 2007)

usadas somente por ele. A maior inovação é a adoção da API JavaDTV, especificada em (ABNT, 2010b), ao invés da *Globally Executable MHP (GEM)* como era prevista na proposta inicial.

A estrutura do Java DTV, mostrada na Figura 2.4 tem como base de seu ambiente de execução algumas API's bastante conhecidas e usadas no desenvolvimento de programas para dispositivos com limitações de recursos em Java com suporte a implementações completas da Máquina Virtual Java: CDC, FP e PBP.

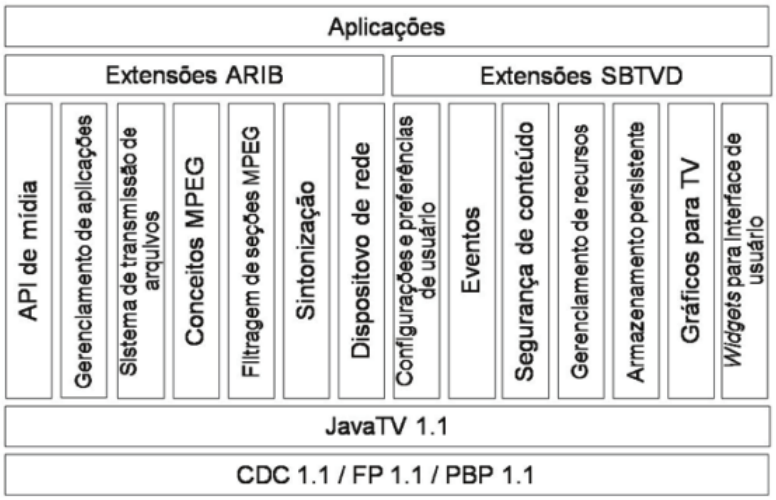

Figura 2.4: Estrutura da API JavaDTV. Fonte: (ABNT, 2010b)

Do ponto de vista da criação de aplicações distribuídas, a Tabela 2.3 resume os principais recursos disponíveis em cada API.

Tabela 2.3: Suporte à programação distribuídas em API's Java para TV Digital

API	**Características**
CDC - Connected Device Configuration (ORACLE, 2011a)	Suporte a conexões IPv4 e IPv6 e transferência de pacotes através de *sockets* UDP; Mecanismos de segurança básicos e gerência de certificados.
FP - Foundation Profile (ORACLE, 2011b)	Suporte a conexões multicast e transferência de pacotes através de *sockets* TCP; Definições sobre Listas de controle de acesso (ACL) e especificação de algoritmos e chaves de criptografia.
PBP - Personal Basis Profile (ORACLE, 2011c)	Suporte para registro e Invocação Remota de Métodos (RMI).

Apesar desses recursos de programação disponíveis, por questão de segurança a Java DTV limita o comportamento da implementação desses componentes, relativos controle de conectividade além de limites do dispositivo, ao elemento *NetworkDevice*. Para compreender como se dá esse processo é necessário antes falar sobre autenticação de programas e recursos escassos.

Conforme consta em (ABNT, 2010a) "A autenticação de qualquer conteúdo ou aplicativo é especificada usando metadados que são assinados e podem ser autenticados no dispositivo[...]". Além disso "A política de aplicativo (conhecida por arquivo de requisição de permissão) define como um aplicativo requisita acesso às funções que são concedidas apenas aos aplicativos confiáveis". Dessa forma, o acesso a recursos e até a comunicação entre os Xlets em execução pode ser definida pela presença (ou não) da assinatura do aplicativo e de sua política.

Alguns recursos são classificados como escassos (*scarce resources*) e precisam de tratamento especial para que sejam alocados e liberados. Um aplicativo não assinado não pode ter acesso a qualquer recurso escasso, e os assinados devem receber acesso de acordo com ações listadas como permitidas na política da aplicação. São considerados escassos, por exemplo, o *tunner*, a tela, os eventos de entrada do usuário e as interfaces de rede.

No caso das interfaces de rede, o seu acesso é feito através de uma classe

NetworkInterface. Através dela o aplicativo pode ou não reservar este recurso para ter acesso à rede, dependendo de sua assinatura e política. E como citado em (ABNT, 2010a) "Uma vez que o aplicativo tenha sido capaz de reservar tal instância, que então em troca pode controlar quando conectar ou desconectar e qual perfil na preferência do usuário utilizar para a conexão".

Com isso, temos que mais uma vez o desenvolvedor de aplicações interativas distribuídas deve se preocupar, apesar de ser em bem mais alto nível que em NCLua, com detalhes de comunicação e acesso a recursos que fogem do escopo da aplicação. Além de que o padrão de comunicação e o modelo de troca de mensagens entre um STB e outro dispositivo poderia ser diferente para cada aplicação.

2.2 *Grids* Orientados a Serviços

Quando um processador de propósito geral é projetado ele visa atender picos de demanda dos usuários e não manter-se numa taxa de utilização alta o tempo todo. Dessa forma, apesar do uso de técnicas como a de escalonamento de processos e *multithreading* que visam aproveitar ao máximo a capacidade dos processadores, é comum encontrarmos máquinas com recursos ociosos. Sabendo disso, um grupo de pesquisadores liderados por Ian Foster (FOSTER; KESSELMAN, 1999) criou o conceito de grids computacionais. Ele foi inspirado pelo funcionamento das redes elétricas (*power grids*) no qual um consumidor ao utilizar uma tomada para conectar um aparelho elétrico não precisa saber da heterogeneidade, da origem e nem de que forma a energia da rede elétrica chega até ele. De forma análoga, nos grids computacionais um conjunto de recursos, especialmente processamento, deve estar disponível para o usuário, idealmente, sem restrições de quantidade e localização dos recursos (SUBRAMANIAN; GOODMAN, 1999).

A ideia era utilizar várias máquinas conectadas através de uma uma rede, de forma que seus recursos se equiparassem aos de supercomputadores. A partir da tradução do trabalho de (QUOCIRCA, 2003) podemos definir um grid como "uma abordagem arquitetural para criar uma infraestrutura tecnológica que disponibiliza um conjunto de recursos de redes, equipamentos e programas [...] que trabalham juntos para maximizar a utilização de cada componente e minimizar a necessidade de *upgrade* da capacidade de componentes individuais". Segundo (STANOEVSKA-SLABEVA et al., 2010), os *grids* podem ser classificados de acordo com o uso dos

equipamentos, o recurso focado e o escopo de compartilhamento.

Quanto ao uso dos equipamentos podemos ter:

- Dedicado: são *grids* nos quais os equipamentos são alocados exclusivamente para servi-lo;
- Oportunista: os equipamentos são usados para outros fins e o *grid* usa os recursos quando estes estão disponíveis.

Já com relação ao escopo de compartilhamento dos recursos, (STANOEVSKA-SLABEVA et al., 2010) distingue os grids em quatro categorias:

- *Cluster Grids*: ou *clusters* são uma coleção de computadores conectados através de uma rede local de alta velocidade usada para recursos de computação ou processamento de dados;
- *Enterprise Grids*: se referem a *grids* que executam aplicações que se limitam a uma única companhia;
- *Utility Grids*: são *grids* implementados por uma empresa para prover a terceirização da utilização dos recursos por outras empresas;
- *Partner/Community Grids*: originado da *eScience* surgiu como um esforço conjunto de cientistas de compartilhar recursos de suas instituições de pesquisa com o restante do mundo. Atualmente, existem comunidades que contam com pessoas que doam os recursos de seus computadores pessoais para projetos dos mais diversos tipos, como no *World Community Grid(http://www.worldcommunitygrid.org)*.

Já a classificação de *grids* de acordo com o recurso baseia-se no fato de que o objetivo dos *grids* é o compartilhamento de algum tipo de recurso. De acordo com esse tipo, (QUOCIRCA, 2003) distingue quatro tipos de *grid*:

- *Compute Grids*: foca em compartilhar recursos de computação, isto é, processamento;
- *Data Grids*: criados para gerenciar grandes quantidades de dados heterogêneos e distribuídos;

- *Application Grids*: provê mecanismos para gerenciamento de aplicações que acessam programas e bibliotecas de forma transparente;
- *Service Grids*: resultado da convergência entre *grids* e o paradigma de orientação a serviços.

Segundo (STANOEVSKA-SLABEVA et al., 2010) estes quatro diferentes tipos de *grid* estão convergindo para *middlewares* que combinam essas funcionalidades. O foco deste trabalho está em *grids* orientados a serviços focado em redes locais, e por esta razão eles terão seu conteúdo mais aprofundado. Mas antes de falar sobre *grids* orientados a serviços em si, será feita uma rápida abordagem sobre orientação a serviços.

2.2.1 Orientação a Serviços

O conceito de software orientado a serviços surgiu como uma alternativa às metodologias de construção de sistemas distribuídos, como CORBA e RPC/RMI. O grande problema envolvendo essas tecnologias é a dificuldade de promover a interoperabilidade entre aplicativos implementados em plataformas distintas. Baseado em (ERL, 2009b), podemos definir a orientação a serviços como um paradigma baseado em unidades de lógicas de solução que podem ser utilizadas coletivamente e de maneira repetida, com o objetivo de atender a metas estratégicas. Cada unidade dessas, que é fisicamente independente de software, pode ser chamada de serviço, que também pode ser compreendido como um contêiner de funcionalidades/capacidades associadas com um propósito comum.

Ainda segundo (ERL, 2009b), na prática podemos dizer que atualmente os mecanismos de implementação de serviços são na forma de:

- Componentes: um software projetado para ser uma parte de um sistema distribuído, geralmente projetado para usar plataformas e tecnologias específicas;
- *Web Services*: um conjunto de lógicas de solução descritas por um contrato consistindo de uma definição usando a linguagem WSDL (CHINNICI et al., 2007) e uma ou mais definições de esquemas em XML;
- *REST Services*: provê um meio de construir sistemas distribuídos baseados em recursos.

Independentemente do mecanismo escolhido, na comunicação os serviços tem definidos um contrato com uma descrição abstrata de suas capacidades (geralmente em XML) que é acessado através de uma interface remota orientada-a-mensagem que utiliza algum tipo de descrição bem definida (VöLTER et al., 2005). Em uma aplicação orientada a serviços temos dois atores principais: aquele que invoca e interage com o serviço, chamado consumidor ou requerente do serviço; e aquele que disponibiliza e provê o serviço, chamado provedor do serviço. É importante observar que o papel de requerente ou provedor é uma condição temporal, como mostrado na Figura 2.5 na qual o elemento do centro é provedor para o elemento acima e requerente para o elemento que está abaixo dele.

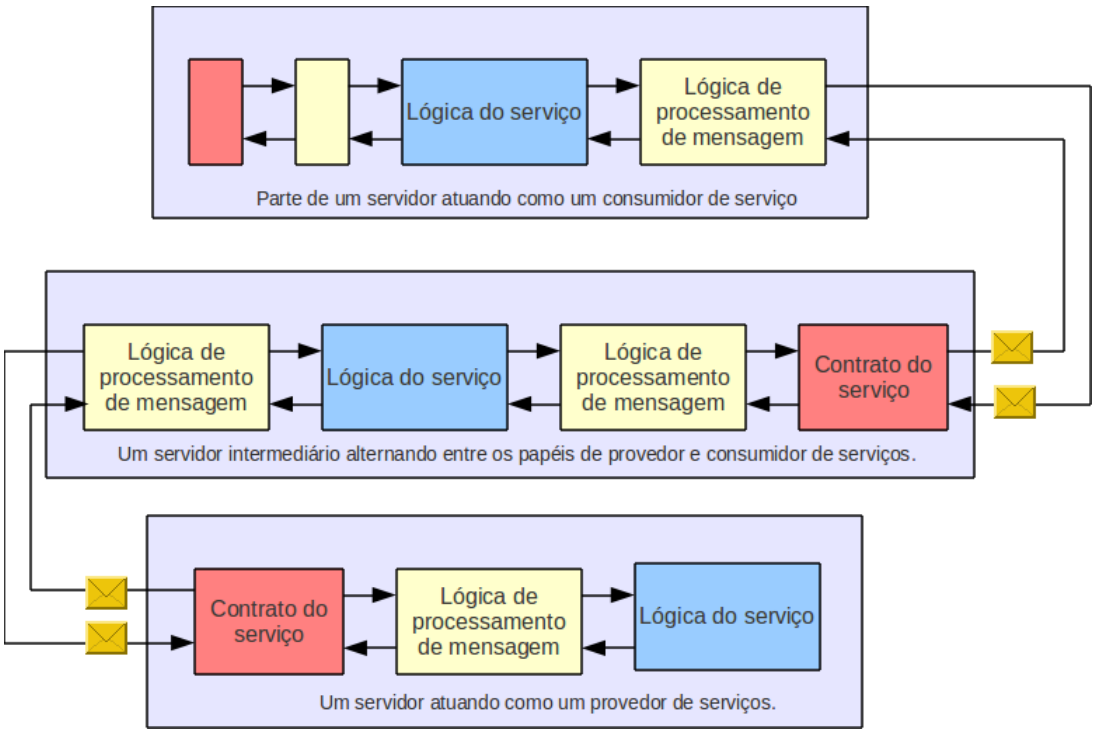

Figura 2.5: Exemplo de comunicação entre provedores e consumidores de serviços. Adaptado de: (ERL, 2009b)

Isso ocorre porque uma das características de uma arquitetura de software orientada a serviços é a composição de serviços. Segundo (ERL, 2009a) ela pode ser definida como uma agregação de serviços coletivamente compostos para automatizar uma tarefa em particular ou um processo de negócios. Geralmente os serviços estão organizadas em coleções padronizadas e gerenciadas dentro de um contexto de uma organização, os chamados inventários de serviços.

Essa forma de organização para construção de sistemas orientados a serviços

fazem com que os projetos orientados a serviços sejam definidos por oito princípios (ERL, 2009b):

- Contrato de serviço padronizado: serviços de um mesmo inventário devem compartilhar do mesmo formato de contrato para que a comunicação ocorra de forma efetiva;
- Baixo acoplamento: um serviço não deve depender de muitos outros elementos para que possa ser usado;
- Abstração: somente devem estar disponíveis informações essenciais no seu contrato, que deve ser a única informação a ser publicada;
- Reusabilidade: os serviços devem ser projetos de forma a garantir seu uso recorrente;
- Autonomia: controle em alto nível sobre seu ambiente de execução;
- *Statelessness*(Ausência de estado): não gerencia o armazenamento de estado;
- Possibilidade descoberta de serviços: fornece informações sobre os serviços disponíveis;
- Possibilidade de composição de serviços: permite a execução de um serviço como uma composição de outros.

2.2.2 Características de *Middleware* para *Grids* Orientados a Serviços

Conforme dito anteriormente, os *grids* orientados a serviços (*Service-Oriented Grids - SOG*) representam o resultado da convergência entre *grids* e a orientação a serviços. Uma grande quantidade de *middleware* para SOG's foi desenvolvida desde o início da década de 2000, com características e protocolos e padrões diferentes. O trabalho de (ANDREOZZI et al., 2006) mostra uma iniciativa de padronização dos vários SOG's com fins de pesquisa existentes na Europa, apontando o que havia em comum e quais as principais tendências para cada um dos aspectos relacionados. Já em (FOSTER et al., 2005) encontramos o resultado do esforço do *Global Grid Forum* em definir um conjunto de diretrizes para uma arquitetura genérica de SOG's como estrutura em três camadas, conforme mosta a Figura 2.6. A camada inferior, que

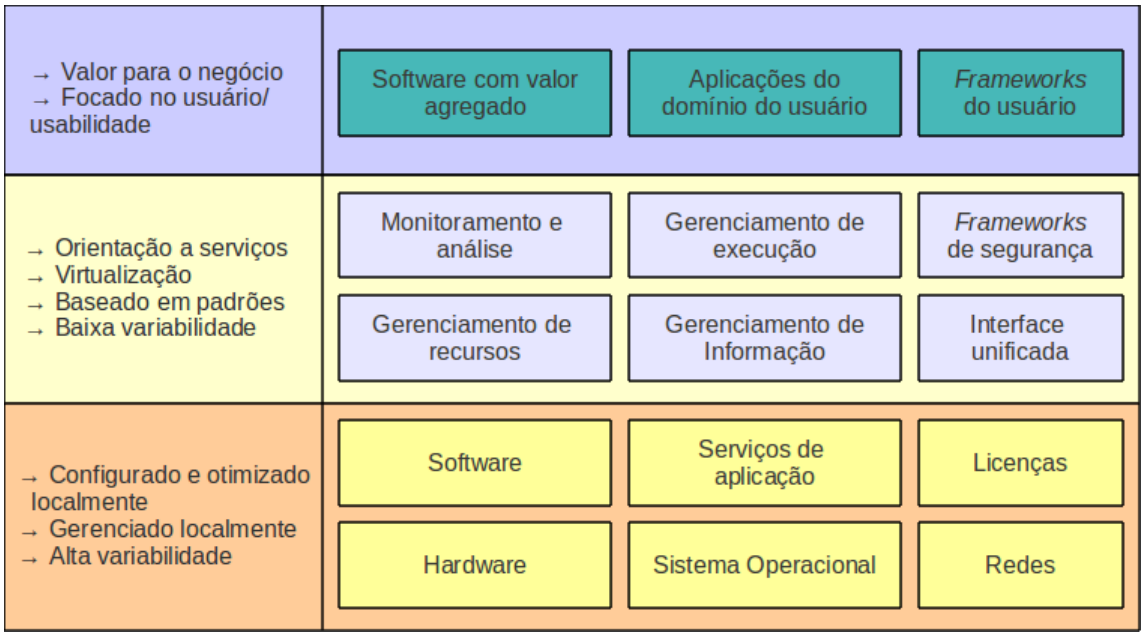

Figura 2.6: Visão conceito da infraestrutura de um *grid* (FOSTER et al., 2005).

está sujeita a alta variabilidade, diz respeito à infraestrutura de cada organização e é gerenciada localmente. A camada do meio trata da padronização do acesso aos serviços e o gerenciamento de suas execuções. Já a terceira é focada no usuário do *grid* e suas aplicações.

A partir desse esforço foi proposta a *Open Service Grid Architecture* (OSGA), com os principais requisitos e capacidades que estariam presentes em cada SOG. Destacamos aqui os principais aspectos nesse contexto:

- Interoperabilidade e suporte a ambientes dinâmicos e heterogêneos: uma vez que um SOG pode tratar uma grande variedade de ambientes, sistemas operacionais, tipos de dispositivos e serviços. São recomendados quatro requisitos: Virtualização dos recursos, a fim de reduzir a complexidade de seu gerenciamento; Gerenciamento uniforme de capacidades, onde é necessário que haja mecanismos comuns e consistentes de gerenciamento; Consulta e descoberta de recursos, no qual deve ser possível descobrir recursos com atributos desejados e ter como retorno suas propriedades; Protocolos e esquemas padrões, importantes para a interoperabilidade e simplificar a transição de aplicações para SOG's;

- Otimização: refere-se ao uso de técnicas usadas na alocação de recursos que atendam aos requisitos tanto do consumidor quanto do fornecedor do serviço;

- Garantia de Qualidade de Serviço (QoS): apresentar mecanismos de garantir a QoS em dimensões que incluem, mas não estão limitadas a segurança, disponibilidade e performance;

- Execução das tarefas: o *grid* deve prover meios de gerenciar a execução das tarefas de seus usuários durante o seu ciclo de vida. Entram aí requisitos como gerenciar vários tipos de tarefas (simples e complexas), escalonamento de acordo com critérios flexíveis e provisionamento de recursos para automatizar a alocação, implantação e configuração de serviços. O trabalho de (ANDREOZZI et al., 2006) também fala sobre descrição das tarefas submetidas.

- Gerenciamento de dados: fazer de forma eficiente o acesso e a movimentação de grande quantidade de dados. Pode-se citar requisitos como acesso, consistência, persistência, integração e gerenciamento de localização de dados. O trabalho de (ANDREOZZI et al., 2006) também fala sobre replicação de dados;

- Segurança: deve haver um controle de acesso aos serviços de acordo com os protocolos e política de segurança especificados. Aqui podem ser incluídos vários requisitos como: autenticação, autorização, delegação de autorização, gerenciamento de contas, detecção de intrusão, *logging*, entre outros;

- Redução de custos de administração: deve haver tarefas administrativas automatizadas para diminuir os custos de administração e a possibilidade de erro humano;

- Escalabilidade: gerenciar de forma adequada quando o número de nós do *grid* e/ou o número de serviços disponíveis é muito grande, especialmente no caso de programas paralelos que desejam obter uma resposta rápida para o seu conjunto de tarefas (*throughput*).

- Disponibilidade: garantir políticas de flexibilidade e recuperabilidade através de mecanismos de recuperação de desastre e gerenciamento de falhas;

- Facilidade de uso e extensibilidade: o *middleware* do SOG deve mascarar a complexidade do ambiente, se requerido, e ter uma arquitetura na qual os componentes principais possam ser extendidos ou substituídos.

Levando em consideração esses requisitos foi apresentado o OSGA Framework que apresenta as principais capacidades dos SOG's. No *framework* são especificados

7 tipos de serviços: infraestrutura, gerenciamento de execução, dados, gerenciamento de recursos, segurança, auto-gerenciamento e de informações. A partir do OSGA Framework, foi criada uma rede de colaboração internacional, a Globus Alliance(www.globus.org/), que disponibiliza ferramentas para usar o *grid*, o Globus Online, e outra para construção de *grids*, a Globus Toolkit, ambas tendo como infraestrutura o padrão de WebServices.

Uma inicativa semelhante com outro foco é o g-Eclipse (www.eclipse.org/geclipse/), que serve tanto para *grids* como para *cloud computing*. O g-Eclipse usa a IDE Eclipse para fornecer ferramentas para que usuários possam criar, submeter e depurar a execução de tarefas, gerenciar recursos disponibilizados, além de permitir extensões para que outros *grids/clouds* se conectem à sua organização virtual(KURNIAWAN; ABRAMSON, 2007). Em ambos os casos, Globus e g-Eclipse, o objetivo principal é dar suporte a iniciativas de colaboração científica ou comercial, o que chamamos de *e-science* e *e-business*, respectivamente.

Contudo, esses conceitos não são aplicáveis a todos os cenários de uso de SOG's. O mesmo grupo do OSGA também produziu um documento no qual são descritos casos de uso para a OSGA, onde são apontadas que características devem ser mais enfatizadas dependendo do propósito de uso do SOG (FOSTER et al., 2004). Um dos casos de uso é chamado de *Grid Lite* e trata sobre SOG's compostos por PDA's, *smartphones* e outros dispositivos com capacidade de hardware reduzidas. Existem também trabalhos como os de (LI et al., 2009) e (MANVI; BIRJE, 2010) que falam sobre as especificidades quando tratamos de *grids* onde os nós geralmente são móveis e tem uma autonomia de energia limitada. Nessas situações há preocupações adicionais além destas apresentadas na OSGA, como:

- Mecanismos de monitoramento dos nós: devido à mobilidade dos equipamentos, os nós podem entrar e sair facilmente do *grid*. Um grande problema disso diz respeito à saída, que pode não ser comunicada ao equipamento responsável pela coordenação dos nós, o que poderia ocasionar a espera por um resultado ou o envio de uma nova tarefa a um dispositivo inalcançável;

- Complexidade da troca de mensagens: como os dispositivos nesses cenários possuem baixo poder computacional, deve haver um cuidado para que a complexidade envolvida na troca de mensagens entre os nós não sobrecarregue o nó de forma a comprometer seu desempenho. Em alguns casos, aspectos como tolerância a falhas e segurança poderiam ficar em segundo plano ou serem

retiradas do *middleware*;

- Autonomia de energia: como os dispositivos móveis podem ter restrições relativas à fonte de energia, o critério de uso racional visando o aumento da permanência dos nós no *grid* pode ser mais importante que outros critérios, como buscar o melhor desempenho das aplicações;

2.3 Trabalhos Envolvendo *Grids* e TV Digital

Atualmente existem ainda poucos trabalhos envolvendo os dois temas, *grid* e TVDI. O trabalho de (BATISTA et al., 2007) fala sobre o TVGrid, uma arquitetura de *grid* que usa os recursos disponíveis em uma rede formada por STB's para execução de aplicações distribuídas em *broadcast*, através do protocolo BIOP (Broadcast Inter-ORB Protocol), definido pelo modelo CORBA (ETSI, 2003). Cada nó do *grid* executa uma aplicação de TVDI em Java (Xlet) e retorna o resultado para a emissora de TV que enviou a requisição. Esse trabalho está mais focado em definir o modelo de distribuição das tarefas entre os STB's e o recebimento dos resultados das tarefas que no próprio gerenciamento do *grid*. Além disso, somente os STB's fazem parte do *grid*, os demais equipamentos de uma rede residencial não são citados na proposta.

Já no trabalho de (COSTA et al., 2009) o objetivo é não só integrar STB's no processamento, mas também outros dispositivos de processamento não tradicionais, como PDA's e telefones móveis. Ele se baseia no princípio de *Infrastructure as a Service* (IaaS), isto é, um *grid* formado por uma organização poderia disponibilizar como um serviço a sua infraestrutura de dispositivos, sejam eles dedicados ou não. Essa infraestrutura seria utilizada de acordo com a demanda de aplicações descritas como do tipo *many-task computing*, isto é, uma aplicação é dividida em várias tarefas que não demandam uma carga excessiva de processamento. A execução destas tarefas seria controlada por agentes que estariam em cada unidade de processamento. Mais uma vez, as aplicações seriam distribuídas em *broadcast*.

O que vemos nesses dois trabalhos é uma visão tradicional do *grid* como um ambiente para execução de aplicações para *e-science* e *e-business*. Entretanto, a convergência entre *grids* e orientação a serviços tem provocado uma mudança de visão sobre os *grids*. Segundo (AUGUSTIN et al., 2008), se antes eram vistos apenas como plataformas para redes de volumosos recursos computacionais, voltados para uma demanda de experimentos científicos, ultimamente os *grids* estão se movendo

em direção a tornar-se uma plataforma genérica para compartilhamento de vários tipos de recursos na rede. Esse mesmo artigo mostra uma tendência chamada de *grid* pervasivo, que explora dispositivos ubíquos ou como fonte de dados ou como uma interface de saída usando uma infraestrutura de *grid* para sua comunicação. O problema nessa abordagem é que nem todos dispositivos para computação ubíqua tem a capacidade de interpretar as mensagens enviadas por um nó do *grid* dependendo do modelo de comunicação utilizado.

Outros trabalhos, como o de (OLIVEIRA et al., 2009) e (VIANA et al., 2009), não usam o paradigma de *grid*, mas apresentam proposições semelhantes para controlar dispositivos espalhados no ambiente residencial através de programas de TVDI. O primeiro utiliza um software próprio, o DIGA Ginga, e o segundo fazendo uma interface de comunicação entre uma aplicação Ginga e o *framework* Services Gateway initiative (OSGi). Em ambos, o STB aparece como controlador de um conjunto de dispositivos que fornecem dados ou atuam no ambiente de forma específica. Isso poderia ser mapeado em serviços disponibilizados pelo STB para que outros equipamentos pudessem utilizá-los de forma indireta.

Por fim, os trabalhos de (MANVI; BIRJE, 2010) e (CORONATO; PIETRO, 2008) falam sobre *grids* envolvendo dispositivos móveis. No primeiro é feita uma revisão sobre *grids* feitos exclusivamente por dispositivos móveis, muitos usando comunicação *ad hoc*. Um problema nessa abordagem é que todos os elementos são pontos críticos, passíveis de falha, o que exige um mecanismo eficiente para monitoramento dos nós. Além disso, pode-se questionar a efetividade desse tipo de *grid* em um ambiente residencial, já que há a expectativa da existência de equipamentos com poder de processamento para substituir vários nós de uma vez.

Já No segundo trabalho, é descrito o *middleware* MiPeG para integrar dispositivos móveis a um *desktop grid* usando ferramentas do *Globus Toolkit* (FOSTER, 2006) em um *grid* pervasivo. Entretanto, segundo (LI et al., 2009) o MiPeG não tratava bem problemas relativos à mudança dinâmica de dispositivos e tinha um fraco suporte a implantação de serviços em dispositivos móveis com restrições de recursos, além de não ter feito testes relativos à aplicação do *grid* à TV. Este trabalho é o que tem mais semelhança ao objetivo do Gringa, mas percebe-se que a opção pela compatibilidade com os padrões do *Globus Toolkit* impôs uma complexidade relativa ao suporte para os dispositivos com capacidade limitada.

A proposta do Gringa difere destes trabalhos em vários aspectos. Em primeiro

lugar, na formação do *grid* são utilizados todos os equipamentos do ambiente doméstico com capacidade computacional, não somente o STB. Esses equipamentos não se restringiriam a executar aplicações de alta carga de processamento, mas de qualquer serviço, incluindo o acesso a outros dispositivos que não podem ser alcançados pelo *grid* de forma direta, utilizando protocolos simples que possam ser gerenciados tanto em um ambiente heterogêneo com máquinas com grande poder computacional como por dispositivos de capacidade limitada.

Capítulo 3

O *Middleware* Gringa

Neste capítulo faremos uma descrição da arquitetura do *middleware* Gringa, proposto como tema dessa dissertação. O nome Gringa foi escolhido por representar a união dos nomes das principais tecnologias envolvidas no trabalho: *Grids* e Ginga. Além disso, a palavra gringa é usada no Brasil como sinônimo de estrangeiro, alguém que vem de fora do país, o que também casa perfeitamente com o objetivo do *middleware* que é proporcionar o desenvolvimento de aplicações onde parte das funcionalidades e os resultados de sua execução são obtidos de outros equipamentos.

3.1 Requisitos e Arquitetura

Conforme descrito na introdução desta dissertação, o Gringa foi projetado para aplicações que possam executar em um *grid* formado por um STB de TV Digital e um conjunto de dispositivos que estejam em uma rede local residencial (HAN). Tais dispositivos poderiam ser tanto um computador *desktop* com recursos abundantes, quanto *notebooks* e *netbooks*, que também tem bons recursos, mas que se usados em demasia podem comprometer a autonomia de energia, como também dispositivos menores e com menor quantidade de recursos como *tablets*, PDA's e telefones celulares com capacidade de instalação de novos programas, os *smartphones*, como mostra a Figura 3.1.

Todos os nós que integram o *grid* usam uma tecnologia orientada a serviços em comum como base para sua comunicação, e os componentes e serviços disponíveis variam de acordo com o dispositivo e as configurações feitas pelos usuários.

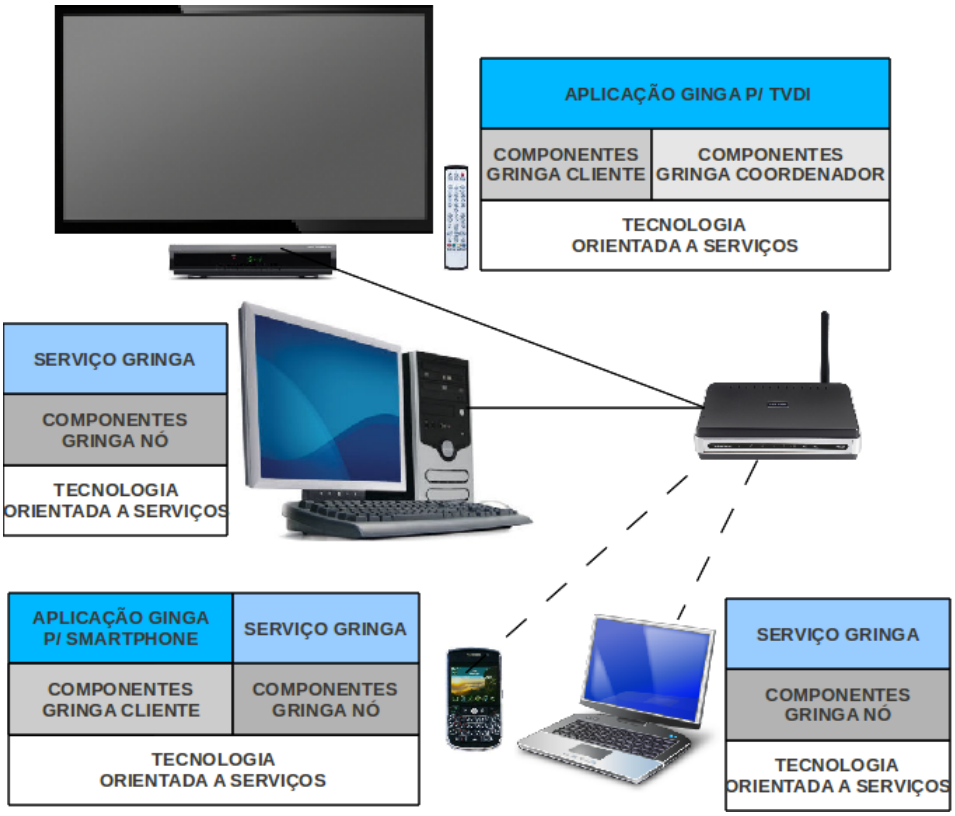

Figura 3.1: Cenário de uso do *middleware* Gringa.

Ambientes como estes são bastante heterogêneos e não se aplicam diretamente às propostas de *middleware* apresentadas na seção 2.3, uma vez que as aplicações interativas para TV podem tanto requerer serviços de nós móveis (como um envio de SMS) quanto dos *desktops* (tarefas de grande carga de processamento). Baseado nisso, podemos definir para o Gringa um conjunto de requisitos não-funcionais:

- Oportunismo: o *grid* utilizará os recursos disponíveis em cada equipamento, mas estes não serão dedicados;
- Interoperabilidade: qualquer nó do *grid* solicita ou oferece serviços através de uma interface comum, independente de plataforma;
- Flexibilidade: a arquitetura deve permitir que cada componente usado (referente a infraestrutura de comunicação, política de segurança, escalonamento, etc) possa ser substituído sem afetar os demais;
- Simplicidade: como o número de dispositivos e serviços tende a ser baixo, pelo foco do Gringa ser em ambiente doméstico, os mecanismos de comunicação e

gerenciamento dos recursos devem ser feitos, preferencialmente, da forma mais simples possível;

- Segurança baseada em autenticação centralizada ou ausente: para diminuir a complexidade de gerenciamento dos nós;
- Solicitações e transferências de dados baseadas em HTTP: para facilitar configurações de *firewall* e diminuir o número de protocolos diferentes a ser suportado;
- Comunicação assíncrona e gerência centralizada: como espera-se um cenário dinâmico, onde dispositivos possam entrar e sair do *grid* frequentemente, a comunicação será feita de modo assíncrono e as informações deverão ser concentradas em um equipamento que pode ser localizado facilmente e que esteja disponível por grande quantidade de tempo, por exemplo um STB que ficará em um ponto fixo de uma residência e permanecerá ligado enquanto as pessoas assistirem a programação da TV;
- Uso de serviço de *proxy*: quando não há uma comunicação direta entre a aplicação e o coordenador do *grid*.

Podemos ver que como o ambiente residencial, para o qual o Gringa foi projetado, pode ter dispositivos bastante heterogêneos, ele não contempla de forma integral todos os requisitos definidos pela OSGA nem os requisitos para os *grids* móveis mostrados na subseção 2.2.2. Ele traz aspectos presentes em ambos, e flexibiliza outros, como descritos no OSGA Use Cases, de forma a equilibrar a relação entre desempenho e otimização.

3.1.1 Descrição Lógica/Estrutural do Gringa

Os *grids* que usam o *middleware Gringa* são formados por um conjunto de nós, que disponibilizam serviços de acordo com as solicitações que são repassadas por um nó que desempenha a função de coordenação. É importante salientar que não há diferença específica do nó coordenador para o demais do *grid*, a não ser pelos serviços que são oferecidos. O objetivo disso é fazer com que qualquer nó possa assumir o papel de coordenador, bastando apenas que o mesmo habilite esses serviços, já que há uma expectativa de não termos um grande número de nós no *grid* uma vez que o Gringa é focado em um ambiente residencial.

No momento de inicialização do *grid*, um dispositivo, que deseja tornar-se coordenador, publica em um servidor de registro que disponibilizará os serviços de coordenação. O servidor de registro, para o nosso caso, deve ser mantido no STB, pois ele é um equipamento que estará fixo na casa e ficará sempre ligado quando quisermos fazer uma interação entre um programa interativo e o *grid*.

Quando um outro nó deseja se unir ao *grid*, ele realiza um processo de descoberta, buscando pelos serviços de coordenação. Ele obtém do servidor de registro uma referência do coordenador e a partir daí o nó, usando um mecanismo padrão de comunicação, repassa ao coordenador que serviços poderão ser invocados.

Geralmente, essa invocação é demandada por um cliente, que não faz parte do *grid* e apenas solicita execuções de tarefas. Para o cliente não é necessário ter a referência de quantos e quais são os nós que podem prover aquela funcionalidade desejada, basta que ele tenha uma referência para o coordenador. A partir daí, ele realiza suas solicitações, ficando o coordenador responsável por procurar o nó que melhor atenda a solicitação do cliente. A Figura 3.2 resume as demandas entre clientes e nós.

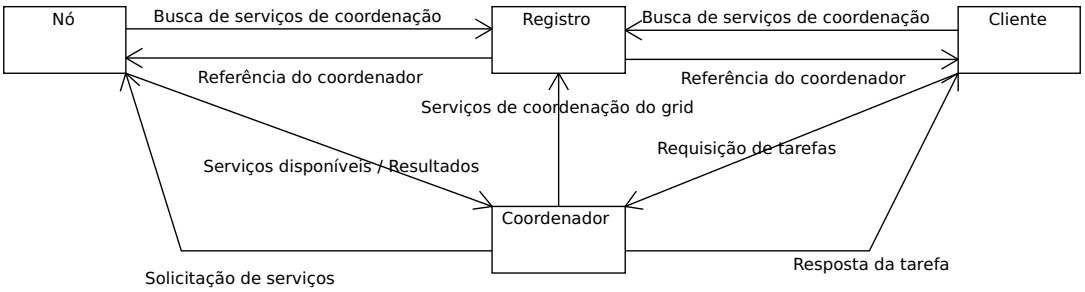

Figura 3.2: Relação entre clientes e nós de um *grid* usando Gringa.

Nela podemos ver que os clientes, após a descoberta do coordenador, simplesmente realizam requisições de tarefas e esperam pelas respostas das mesmas. Uma aplicação cliente pode requisitar diversas tarefas e pode estar executando em um dispositivo qualquer, desde um STB até um *desktop*.

Como as chamadas de tarefas são mapeadas em serviços, é necessário que haja uma padronização nos formatos das mensagens de solicitação e resposta, assim como também é recomendável um mecanismo que abstraia a comunicação de baixo nível, como o descrito pelo padrão de projeto *Broker* (VöLTER et al., 2005). Isso facilitaria a criação um conjunto de ferramentas e bibliotecas, organizados em *frameworks*,

tornando mais ágil o processo de construção de aplicações para *grids*.

A partir disso podemos definir a arquitetura de um cliente – solicitante de serviços – em três camadas conforme mostra a figura 3.3. A camada de aplicações contém as implementações, para os mais diversos tipos de dispositivos, de programas que fazem uso do *grid*. As solicitações são enviadas e recebidas através da camada inferior, de comunicação, onde haverão um *broker* e um *framework* para a comunicação com o coordenador, usando alguma tecnologia de orientação a serviços. Na camada intermediária estaria uma camada de acesso a serviços, que seria papel do Gringa no cliente. Nela podemos ver dois *frameworks*: um de comunicação com o *grid*, que acessa diretamente a camada de comunicação para utilizar as funcionalidades providas pelo *grid*; e outro de submissão de tarefas, que é acessado pela camada de aplicações a fim de abstrair toda a complexidade presente nas solicitações de tarefas.

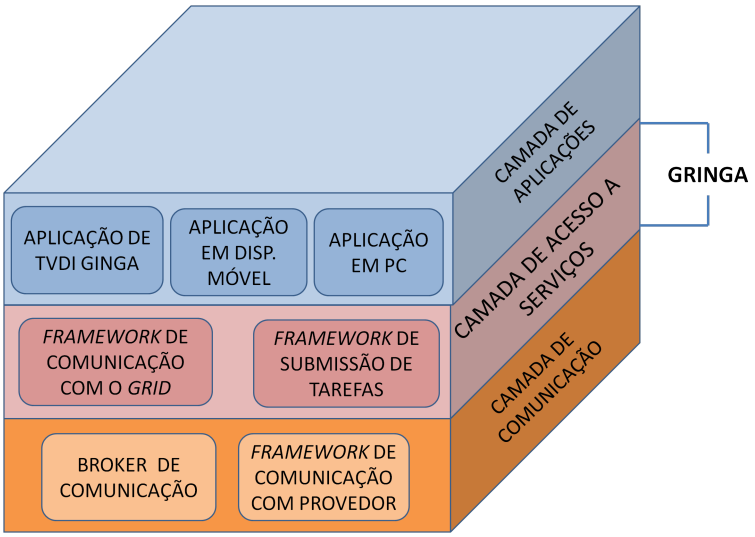

Figura 3.3: Arquitetura de um cliente do *grid* Gringa.

Para os nós com o Gringa – que vão disponibilizar serviços – seguimos um caminho inverso, isto é, eles recebem requisições feitas pelo coordenador e mapeiam em um serviço. Esse serviço acessa um recurso provido pelo nó, que pode ser um componente de um sistema distribuído, o acesso indireto a outro dispositivo, ou até mesmo uma chamada a outro serviço.

Por essa razão, um nó do *grid*, mostrado na figura 3.4, também é subdividido em três camadas e, assim como o cliente, tem em sua camada de comunicação um *broker* e um *framework* de comunicação para acessar outros nós provedores de serviços. Além disso, ele possui um outro elemento para fazer uma interface entre as requisições e o contêiner dos serviços que são oferecidos no nó, sendo este também

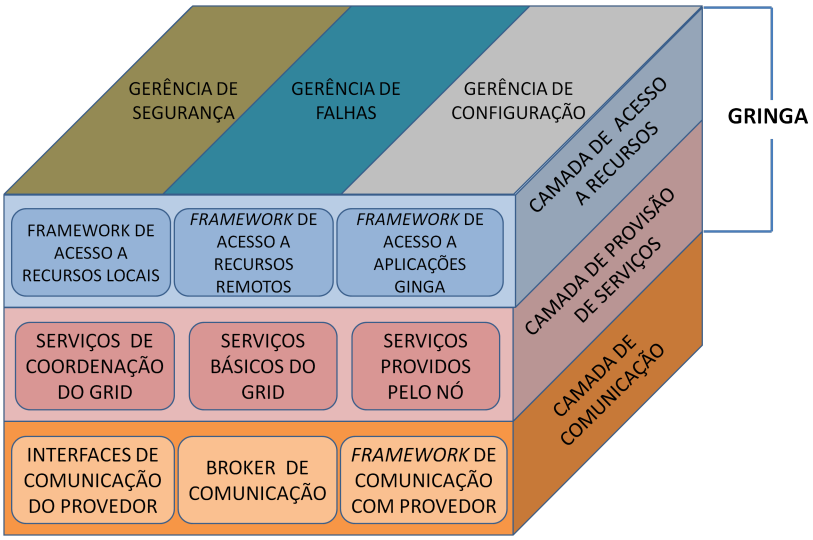

Figura 3.4: Arquitetura de um nó de um *grid* Gringa.

parte da tecnologia de orientação a serviços que está sendo usada.

Nas outras duas camadas, temos componentes providos pelo Gringa. A camada do meio se refere aos serviços disponibilizados nos nós Gringa. Estes serviços estão separados em três categorias: os serviços de coordenação do *grid*, os serviços básicos do *grid* e os providos pelo nó. Os chamados serviços de coordenação desempenham um papel de administração do *grid* e estão disponíveis em apenas um nó, o coordenador. Os serviços básicos estão presentes em todos os nós (inclusive o coordenador), provendo funcionalidades comuns que são necessárias à execução das tarefas. Já os serviços providos pelo nó representam as tarefas em si, isto é, são aqueles que serão requisitados pelos clientes e cuja disponibilidade varia de nó para nó.

Na terceira camada da arquitetura há ainda a camada de acesso aos recursos, sejam eles locais ou remotos. Além dos recursos físicos do próprio equipamento, os componentes do Gringa também são tratados como recursos locais. Temos ainda que nem todos os dispositivos que estão conectados à HAN podem ter suporte às bibliotecas de serviços, e é também papel dos elementos dessa camada servir como interface de acesso entre o nó e esse equipamento.

Um desses equipamentos pode ser um STB, e para este caso, o Gringa tem um componente específico usado para acesso a aplicações de TVDI. Com isso, um STB pode assumir um papel tanto de cliente como de nó do *grid*, podendo os serviços serem disponibilizados de forma direta, através de aplicações nativas ou no padrão Ginga no próprio STB, ou de forma indireta, através de um nó que se comunique

com ele através da camada de acesso a recursos.

Ainda na Figura 3.4, temos que as camadas do nó estão sujeitas à mecanismos de gerência que tratam temas transversais a todas: segurança, para o acesso aos nós e aos serviços; falhas, para determinar as políticas para detecção e correção de falhas; e configuração, para controlar quais mecanismos estão sendo usados na execução da comunicação e nos serviços do *grid*. Essas gerências não estão implementadas em componentes específicos, mas servem como diretrizes para a implementações dos componentes do *middleware*.

3.1.2 Descrição Comportamental do Gringa

Como já explicado, todos os nós Gringa oferecem um conjunto de serviços básicos para o funcionamento do *grid* e um conjunto de serviços específicos para execução de tarefas. Apenas um nó no *grid* oferece também serviços administrativos e este nó, chamado de coordenador, é responsável por gerenciar a entrada e saída de nós no *grid*, assim como controlar a requisição e execução de tarefas.

A definição sobre quais seriam esses serviços foi realizada a partir da análise da OSGA e seus casos de uso, assim como das definições dos principais serviços para *grids* de dispositivos móveis, seguindo os requisitos listados no início deste capítulo.

Os serviços básicos são classificados em quatro tipos: Notificação, Gerenciamento de Tarefas, Armazenamento de dados e Atualização de Informações do Nó. Podemos descrevê-los da seguinte forma:

- Notificação - Tem o papel de notificar o cliente ou nó solicitante de alguma tarefa sobre informações relativas a alteração do status da mesma, ou seja, se foi concluída, cancelada, se houve erros etc. Esses serviços são usados para notificar observadores interessados na execução do serviço usando o método *publish/subscribe*;

- Gerenciamento de Tarefas - Responsáveis por receber solicitações de execução dos serviços disponíveis no nó, instanciar a sua execução e alterar seu status de acordo com o ocorrido, armazenando o resultado a ser devolvido. Usam os serviços de notificação para informar alterações no status e podem usar serviços de transferência de dados para obter ou armazenar arquivos usados nas tarefas ou o seu resultado;

- Armazenamento de dados - Manipulam um repositório de arquivos do nó, podendo receber solicitações de *download*, *upload* ou listagem de arquivos disponíveis. Os arquivos podem ser persistidos de forma permanente ou por tempo determinado, ou até mesmo replicado. São usados pelo serviços de gerenciamento de tarefas conforme descrito no item anterior;

- Atualização de Informações do Nó - Enviam ao nó coordenador, quando um nó ingressa no *grid* e quando atualiza seu status, informações relativas às características do nó sobre informações de desempenho utilizadas pelo escalonador e que serviços estão disponíveis. Também informam ao coordenador que o nó está saindo do *grid*.

Os serviços administrativos são classificados em: Gerenciamento de Diretório, Gerenciamento de Aplicações, Gerenciamento de Nós e de Escalonamento. Podemos descrevê-los da seguinte forma:

- Serviços de diretório - Armazenam e disponibilizam informações sobre os serviços disponíveis no *grid* e sobre nós que fazem parte dele. Os serviços de diretório são acessados tanto pelo Gerenciador de Nós para guardar as informações obtidas através deste, quanto pelo Escalonador que usa suas informações para definir a distribuição dos serviços;

- Gerenciamento de aplicações - Gerenciam as tarefas distribuídas pelo escalonador para cada aplicação cliente diferente, fazendo a interface entre ele e o nó que está fornecendo o serviço. Esses serviços interagem diretamente com os de escalonamento pois guardam uma referência de tarefas submetidas pelo escalonador e também notificam o escalonador em caso de falha ou demora de resposta de um nó em dar um resultado;

- Gerenciamento de nós - Monitora a chegada e a saída dos nós do *grid*, assim como faz uma checagem da permanência dos nós que estão conectados, a fim de descobrir eventuais falhas de conexão. Os serviços de gerenciamento de nós guardam todas as informações recebidas dos nós no servidor de diretório.

- Escalonamento - Responsáveis pelo recebimento de requisições e pela distribuição das tarefas pelos nós do Gringa, de acordo com os critérios pré-estabelecidos em seu algoritmo de controle. O escalonador consulta as informações no servidor de diretório e pode ser acionado pelo gerenciador de nós caso haja falha na execução de uma tarefa.

Cada serviço pertencente a um destes tipos é usado como interface para acessar os componentes que implementam as funcionalidades do *middleware*. Esses componentes são mostrados através de um diagrama de componentes UML na Figura 3.5 e são detalhados posteriormente, na seção 3.2.

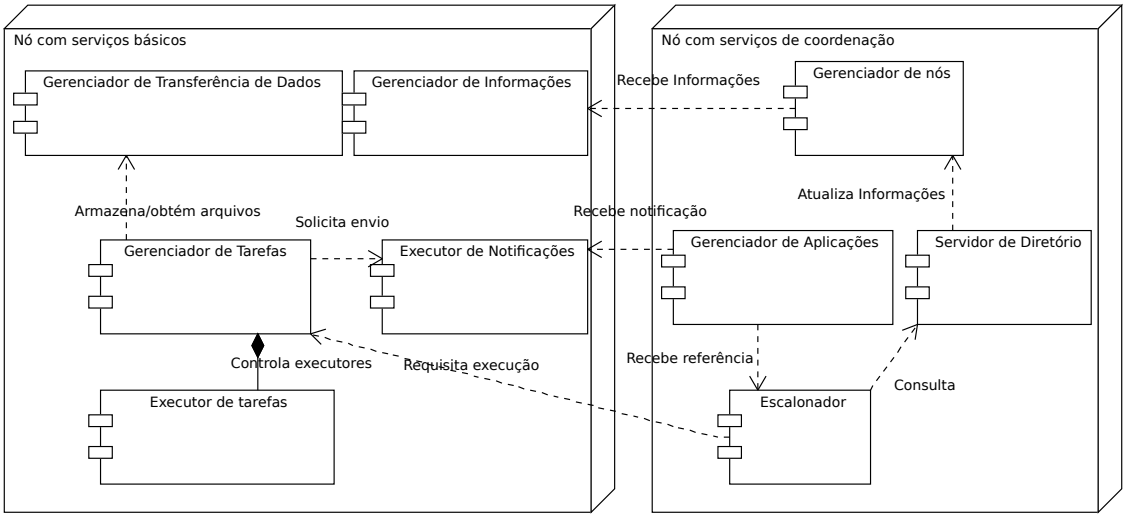

Figura 3.5: Principais componentes do Gringa e suas relações.

Eles interagem de forma a possibilitar os processos de conexão, desconexão ou saída voluntária de um nó do *grid*, assim como do recebimento de uma solicitação de uma tarefa pelo cliente e envio da resposta. Esses procedimentos são detalhados a seguir através o uso de diagramas de atividades UML.

O processo de conexão admite que o nó já tem a referência remota do coordenador obtida no servidor de registro. Ele então solicita ao seu Gerenciador de Informações que efetue o registro junto ao Gerenciador de Nós do coordenador, que envia ao Servidor de Diretório informações a respeito do equipamento e uma lista com os serviços que está disponibilizando. Caso haja serviços que sejam desconhecidos pelo Servidor de diretório, o mesmo envia uma solicitação para que o Gerenciador de Informações do Nó mande suas descrições. Esse processo está descrito na Figura 3.6.

Após esse processo, o nó figurará na lista de disponíveis para envio de tarefas. Uma vez que pelo menos um nó se conecte e passe oferecer um serviço, já é possível que uma aplicação cliente faça requisições. As tarefas são submetidas ao *grid* através de chamadas de serviços cujo conteúdo é a própria chamada da tarefa.

Inicialmente, é necessário que o cliente se registre no Gerenciador de Aplicações

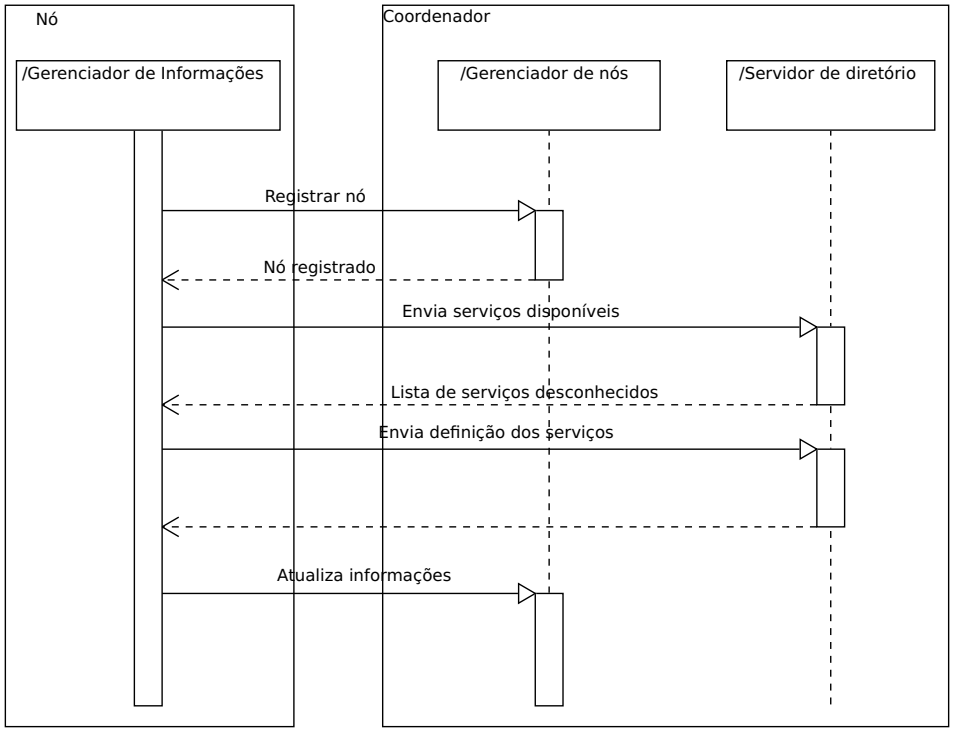

Figura 3.6: Diagrama de sequência para conexão no *grid*.

do coordenador. O Gerenciador devolve ao cliente um identificador que será usado para referenciar as tarefas relativas àquela aplicação. O cliente então solicita ao Escalonador que requisite a execução das tarefas em algum nó do *grid* que a esteja ofertando.

Para processar a solicitação das tarefas, o escalonador verifica junto ao Servidor de Diretório se ele possui o serviço cadastrado em sua base de informações e se sua requisição está correta. Caso esteja, são retornadas informações sobre aquele serviço que possam ser úteis ao algoritmo de escalonamento, como a complexidade da tarefa e os recursos que ela utiliza, por exemplo. Após isso, o Escalonador verifica no Servidor de Diretório quais os nós que disponibilizam aquele serviço solicitado, para que seja feita a requisição.

Cada requisição é enviada para o nó através do seu Gerenciador de Tarefas, que instancia o serviço e devolve uma confirmação que aceitou a requisição.

Por exemplo, suponha que haja um serviço disponível em um nó que realiza uma multiplicação de duas matrizes, cujos arquivos com conteúdos das mesmas serão baixados pelo Gerenciador de Transferência de Arquivos usando referências remotas enviados na requisição. Quando uma requisição de uma tarefa desse tipo

é enviada pelo cliente, o Escalonador verifica no Servidor de Diretório quais os nós que estão aptos a executar esta tarefa e qual a carga de processamento de cada. De posse dessas informações, o Escalonador define qual dos nós poderão executar a multiplicação com mais rapidez e faz uma solicitação para que ele a execute. O nó tenta instanciar um executor para aquela tarefa e caso tenha sucesso notifica o Escalonador. Esse processo de requisição de tarefas é mostrado através do diagrama de sequência da Figura 3.7.

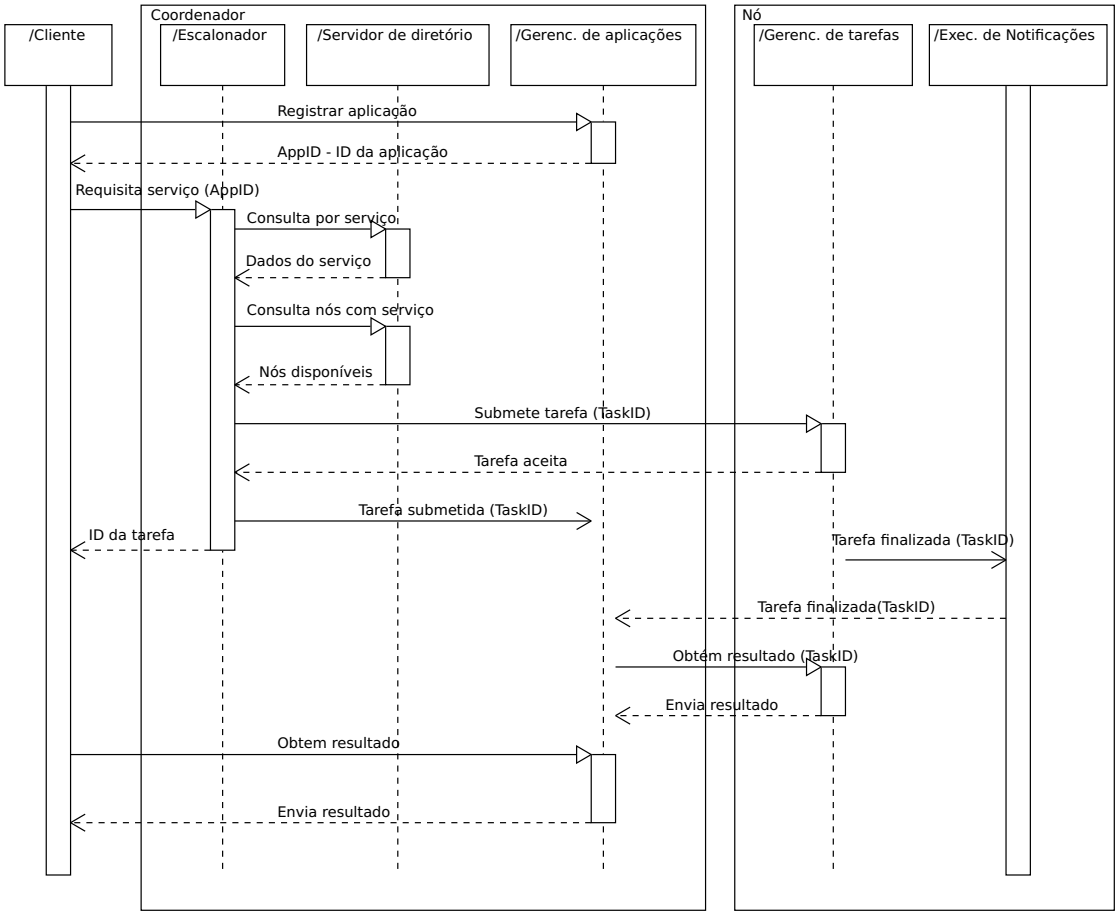

Figura 3.7: Diagrama de sequência para a requisição de serviços.

Após isso, os identificadores das tarefas são repassados para o Gerenciador de Aplicações que será o responsável por monitorar a conclusão delas. Quando concluída uma tarefa, o Executor de Notificações do nó envia uma mensagem ao Gerenciador de Aplicações do coordenador, que solicita o resultado para que seja armazenado e devolvido à aplicação cliente. Vale salientar que em um serviço semelhante ao da multiplicação de matriz comentada anteriormente, o seu resultado pode ser armazenado pelo Gerenciador de Transferência de Dados e nesse caso a

resposta seria apenas um identificador para a recuperação do mesmo.

Também é preciso dizer que a sequência só segue estes passos quando existem nós que disponibilizem o serviço requisitado e que tanto os serviços básicos do nó quanto a aplicação cliente não são encerrados no meio da execução. Várias situações podem ocorrer durante essas etapas e as tarefas podem assumir os estados mostrados na Figura 3.8.

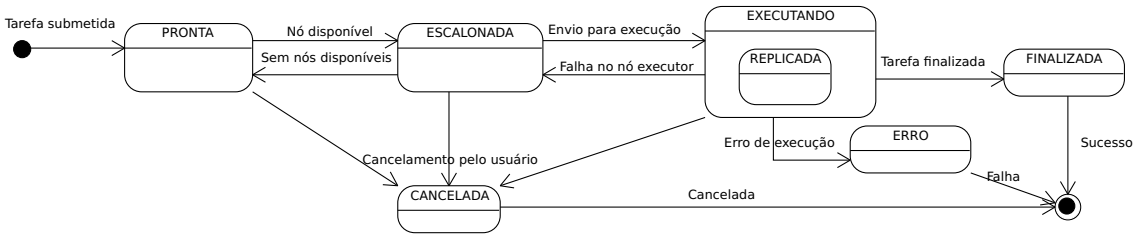

Figura 3.8: Diagrama de estados das tarefas no *grid*

.

Quando submetida, uma tarefa vai para o estado de PRONTA. Caso haja algum nó com o serviço disponível, ela entrará na fila do escalonador e fica no estado de ESCALONADA. Uma vez que sua requisição seja feita ao nó, ela entra no estado de EXECUTANDO ou REPLICADA, dependendo do algoritmo de escalonamento usado. Uma vez concluída, a tarefa passa para o estado de FINALIZADA.

Outros dois estados são mostrados: o de ERRO quando a execução da tarefa é finalizada por problemas na execução e CANCELADA, quando a aplicação cliente desiste de sua execução. Uma tarefa também pode voltar do estado de execução para escalonada caso haja problemas na execução no nó e não exista replicação, assim como pode ir de escalonada para pronta caso os nós que estavam oferecendo o serviço não estejam mais no *grid*.

Já o processo de remoção do nó do *grid* pode acontecer de forma espontânea, por requisição do mesmo, ou provocada pelo gerenciador de nós, caso ele não se comunique com o coordenador por um determinado período (*timeout*) ou então não responda a uma requisição de serviço. A primeira providência é o cancelamento de todas as tarefas em execução.

Quando isso termina, o nó interage com o Gerenciador de Nós do coordenador, mandando uma mensagem indiciando que quer se retirar do *grid*. O Gerenciador de Nós manda o Gerenciador de Aplicações reescalonar as tarefas canceladas, distribuindo-as para outros nós. A Figura 3.9 mostra o processo quando o próprio

nó requisita sair do *grid*.

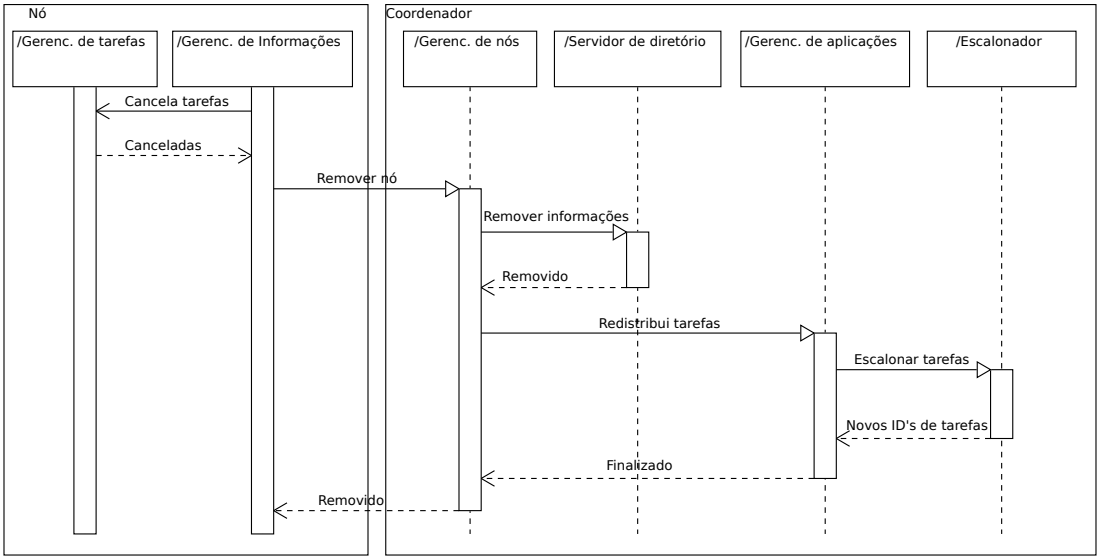

Figura 3.9: Diagrama de sequência do processo de desconexão do *grid*.

3.2 Detalhamento dos Elementos da Arquitetura

Como dito na seção anterior, o Gringa foi modelado em um conjunto de componentes que são acessados usando usando serviços como interface. Nesta seção mostraremos em mais detalhes como foram projetados cada um desses componentes. Para a modelagem desses elementos foi usado um estilo arquitetural de Componentes e Conectores.

Nesse estilo, cuja descrição pode melhor ser compreendida em (WESLEY, 2003), um sistema é decomposto em elementos concretos que definem as funcionalidades do software, chamados Componentes(representados pela retângulos de cor azul), e outros que executam transferência de controle e dados entre componentes, chamados Conectores (representados pela retângulos de cor amarela). Na prática, um componente pode ser desde uma simples classe em um programa orientado a objetos até um subsistema complexo que pode ser abstraído pela funcionalidade provida por ele. Já os Conectores são entidades que conectam componentes, podendo representar fluxos de dados, chamadas a procedimentos (locais ou remotos), interfaces para controle de acesso a recurso, entre outros.

Os componentes somente interagem uns com os outros através dos conectores,

e esta interação é demonstrada através de links, representados pelas linhas, e por interfaces representadas pelas setas que ficam nas extremidades das linhas. Cada interface representa um ponto de conexão onde se define, entre outras coisas, uma predominância da direção de fluxo de mensagens (entrada, saída ou entrada/saída) e o tipo de dependência entre o componente a qual ela está vinculada e o conector com o qual ela se liga(se ela requer, provê, ou requer e provê serviços).

O software utilizado para realizar a modelagem dos componentes do Gringa nesse padrão arquitetural foi o ArchStudio 4(DASHOFY et al., 2007). Ele trabalha com a linguagem de descrição arquitetural xADL(KHARE et al., 2001) e possui um editor gráfico, o *Archipelago*, no qual foram feitas as figuras demonstradas nessa seção. O ArchStudio também possui um editor baseado num modelo de árvores de diretórios e arquivos de propriedades, o *ArchEdit*, no qual a arquitetura pode ser definida e o gráfico resultante pode ser gerado automaticamente. Além destas duas ferramentas para a elaboração da arquitetura, o ArchStudio também possui a *ArchLight*, uma ferramenta para verificar a consistência da arquitetura descrita levando em consideração, dentre outras coisas se:

- Cada elemento possui um nome (identificador) e se o modelo não apresenta duplicidade de nomes;
- Todas as interfaces entre componentes e conectores apresentam as direções de trocas de mensagens e definem se a interface requer ou oferta um serviço;
- Cada elemento da arquitetura possui um tipo definido e se as suas interfaces condizem com a especificação de seu tipo;
- As ligações estão consistentes, se um elemento que tem uma interface que requer um serviço está ligado a um elemento cuja interface provê um serviço.

Além disso, o Archstudio possui uma ferramenta *binding* para geração de código na linguagem Java (DASHOFY, 2001), que foi usado como base para a implementação do *middleware* Gringa.

Sendo assim, temos que nesta ferramenta (e linguagem arquitetural) os componentes são descritos em função de seu tipo, da sua função na arquitetura e de suas interfaces de interação com os outros. Como exemplo, a Figura 3.10 representa o conector *broker* de comunicação, presentes nas Figuras 3.4 e 3.3, que representam a funcionalidade de comunicação que é implementada pelas plataformas de serviços.

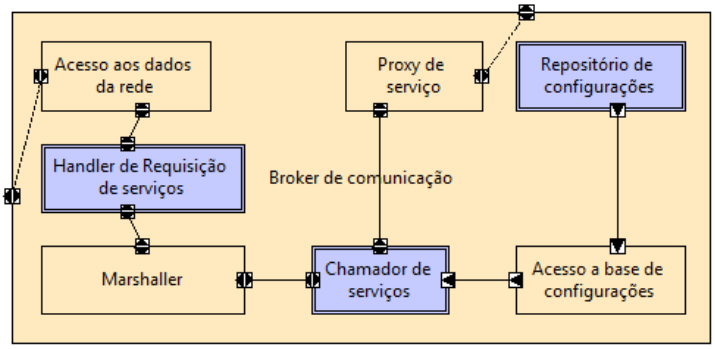

Figura 3.10: Arquitetura de um *broker* de comunicação.

Na sua borda temos uma interface para um conector "Acesso à interface de rede" que acessa dados no dispositivo utilizado para realizar a comunicação com o dispositivo (*Wifi, Bluetooth, Ethernet* etc). Ela provê os dados que são requeridos pelo "*Handler* de requisições de serviços" que é responsável por gerenciar o recebimento de requisições de serviços. Essas requisições são passadas para um conector "*Marshaller*" que filtra as informações a partir dos dados recebidos e entrega ao "Chamador de serviço" que passa a acompanhar a execução do serviço, controlando-a através do conector "*Proxy* de serviço".

É importante ressaltar que todas as descrições contidas no documento tiveram sua consistência validada pela ferramenta *ArchLight*. Isso significa que para cada componente e conector existe um modelo e tipo definido, bem como os tipos de relações que eles tem com os demais.

Passaremos agora a descrever cada elemento que é acessado pelos serviços do Gringa, em nível de nó com serviços básicos e nó com serviços de coordenação descritos na seção anterior. Pela sua simplicidade, os detalhes do servidor de notificação serão omitidos, assim como os serviços de acesso a recursos locais e remotos, que são semelhantes aos que serão mostrados no Gerenciador de Transferência de Dados.

A arquitetura completa de um nó provedor de serviços e de um nó coordenador estão mostrados nas Figuras 3.11 e 3.15. Nas subseções a seguir serão mostrados os componentes e conectores relativos a cada elemento da arquitetura especificado.

3.2.1 Componentes de Serviços Básicos

A arquitetura com a interação de componentes e conectores que executam os serviços básicos de um nó é representada na Figura 3.11. Veremos em detalhes cada

um deles nas subseções a seguir.

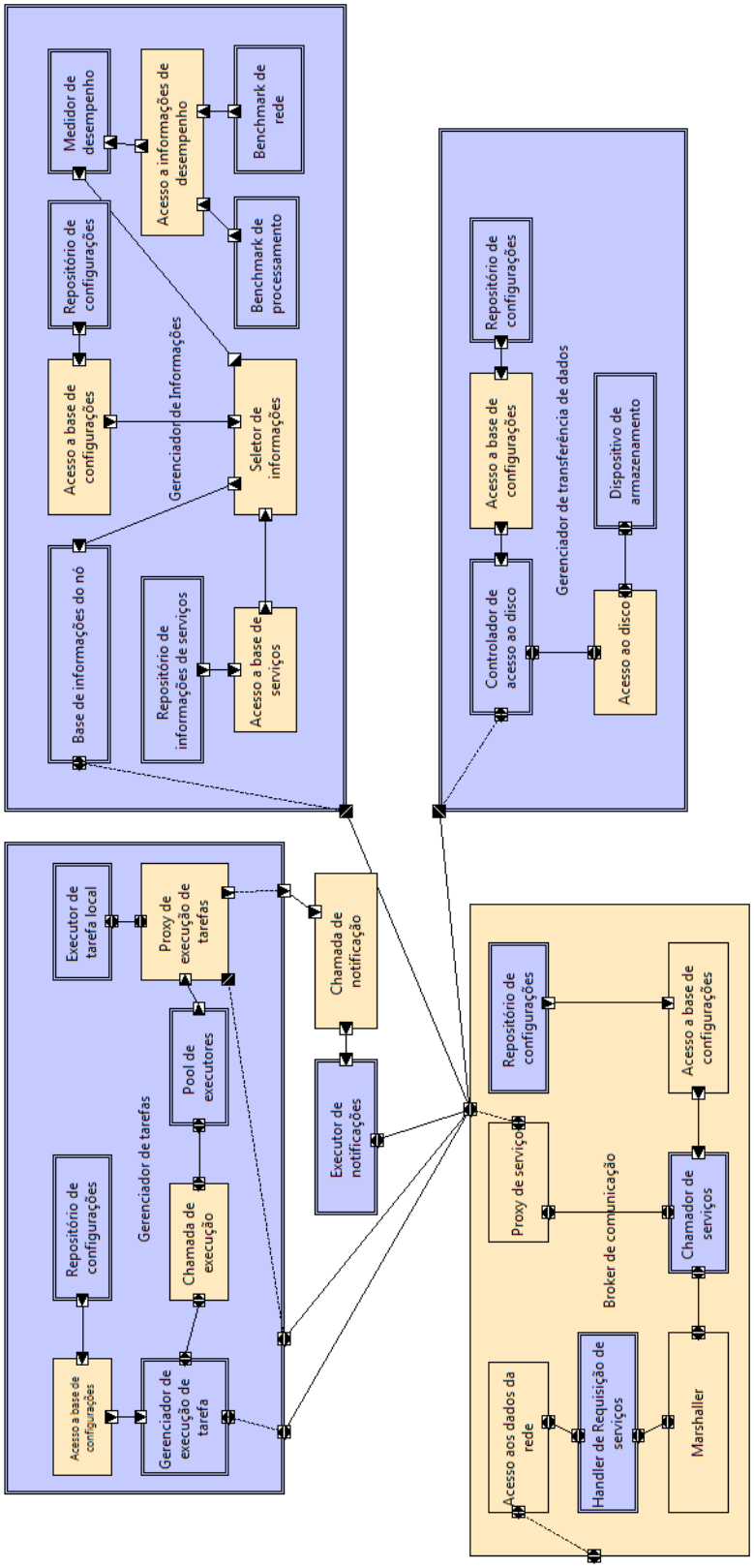

Figura 3.11: Arquitetura dos componentes dos serviços básicos do *grid*.

3.2.1.1 Gerenciador de Tarefas

O Gerenciador de Tarefas, que está mostrado na Figura 3.12, é o componente responsável por acompanhar a execução das tarefas no nó. Podemos notar que ele e é ligado externamente ao *broker* de comunicação, especificado na Figura 3.10 e presente em todos os componentes desta arquitetura. O *broker* realiza a abstração da comunicação entre o nó e o restante do *grid*. Através dele são recebidas chamadas externas de requisição de serviços e são enviadas as respostas das tarefas.

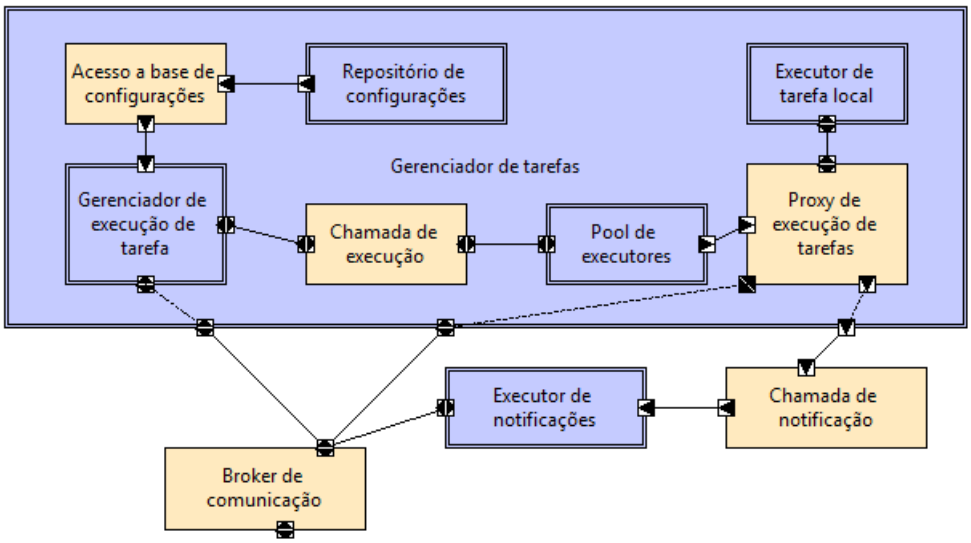

Figura 3.12: Arquitetura do serviço de execução de tarefas.

As requisições são processadas pelo componente interno "Gerenciador de Execução de Tarefas". Esse gerenciador recebe, através de um conector de "Acesso à base de configurações" um conjunto de parâmetros vindos do componente onde está o "Repositório de configurações do nó", que pode conter por exemplo, quais serviços estarão disponíveis, limites de *timeout* para respostas, referências remotas dos serviços do coordenador, entre outros.

O "Gerenciador de Tarefas" controla, através de um conector de "Chamadas de execução", o comportamento de um "*Pool* de executores" que usam o padrão de projeto *Per instance request* (VöLTER et al., 2005). Com ele, quando uma chamada de execução de tarefas é recebida, o gerenciador faz com que uma instância de um "Executor de tarefas local" seja criada e acessada por um conector "*Proxy* de execução de tarefas". Caso necessite de um serviço que não está disponível localmente ou que esteja disponível através de uma composição de serviços, o *proxy* é usado para acessar o próprio *broker* de serviços para fazer a requisição externa.

Uma vez que uma tarefa esteja concluída, o *proxy* envia uma "Chamada de Notificação" para que o "Serviço de notificação" possa avisar ao coordenador do *grid* que a tarefa foi concluída. Uma vez que uma tarefa é concluída, o resultado continua armazenado na instância do executor até que seja requerido e o executor seja desalocado. O *pool* de executores usa o padrão de gerenciamento de ciclo de vida *Leasing*(VöLTER et al., 2005), ou seja, a tarefa fica executando e guardando o resultado por um período determinado no executor, e caso o resultado não seja requerido dentro desse período o executor é removido.

3.2.1.2 Gerenciador de Transferência de Dados

O Gerenciador de Transferência de Dados está especificado na Figura 3.13. Ele é o componente responsável pelo armazenamento persistente de dados que podem auxiliar, serem parâmetros de entrada ou resposta das chamadas de um serviço.

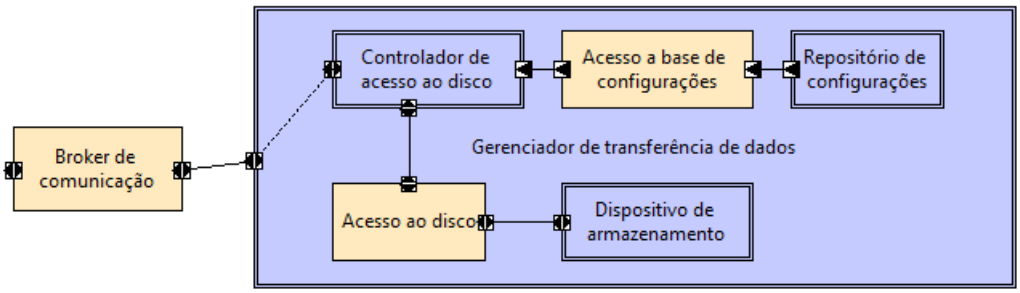

Figura 3.13: Arquitetura do Gerenciador de Transferência de Dados.

Assim como o Gerenciador de Tarefas, ele também tem um conector para acessar as configurações do serviço, que no caso podem ser: tamanho máximo do arquivo, quantidade em disco reservada, política de acesso e tempo até ser excluído. O componente que faz acesso é o "Controlador de acesso ao disco". Ele funciona como um padrão de projeto *Facade* fornecendo uma forma simples de acessar algum dispositivo de armazenamento que possa ser usado pelo nó.

O padrão *Facade* usado na arquitetura desse componente também pode ser usado para os outros componentes da camada de acesso a recursos do nó. Aqui é mostrado o gerenciador de transferência de dados por este ser um serviço básico padrão do Gringa. Entretanto, se trocarmos o foco do disco para obter/enviar informações a um sensor ou a um aparelho de TV que não tem suporte a se conectar diretamente com o *grid* teremos componentes internos com funções semelhantes.

3.2.1.3 Gerenciador de Informações

O Gerenciador de informações do nó é mostrado na Figura 3.14. Ele é responsável por coletar e enviar ao coordenador do *grid* informações relevantes sobre o equipamento onde o *middleware* está rodando.

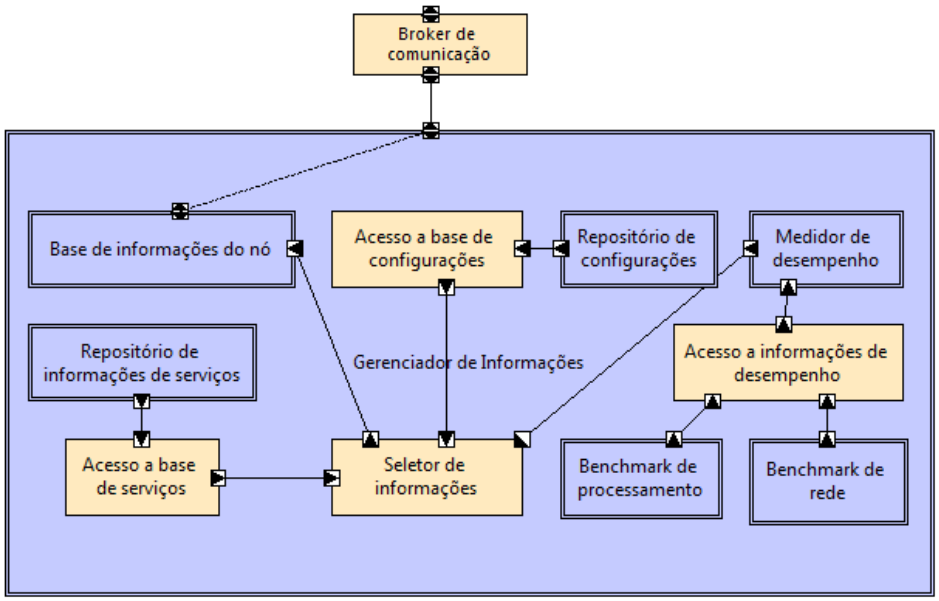

Figura 3.14: Arquitetura do Gerenciador de Informações do nó.

Ele é composto por um componente principal "Base de informações do nó", que concentra as informações que vem através do conector "Seletor de informações". Esse seletor faz acesso a três bases de informações diferentes. No centro, assim como os demais serviços, está o acesso às configurações internas do próprio nó.

Além destas, também são repassadas informações relativas ao "Repositório de informações de serviços", por exemplo que serviços estão disponíveis no nó e como é sua interface de acesso. Mais a direita temos o componente "Medidor de desempenho", que através de componentes que utilizam algoritmos de *benchmark* verificam a capacidade de processar informações e transferir dados pela rede do equipamento.

Para isso são usados dois componentes: o *Benchmark* de processamento, que dá uma medida aproximada da quantidade de operações que o nó executa em um tempo específico; e o *Benchmark* de rede, que estima a quantidade de dados transferidos pela pela rede em um período de tempo. Ressaltamos que esses componentes ficam fazendo avaliações periódicas de atualização, já que um equipamento pode estar executando uma aplicação que demanda mais recursos em um certo momento.

3.2.2 Componentes de Serviços de Coordenação

A arquitetura com a interação de componentes e conectores que executam os serviços de coordenação do *grid* é representado na Figura 3.15. Veremos em detalhes cada um deles nas subseções a seguir.

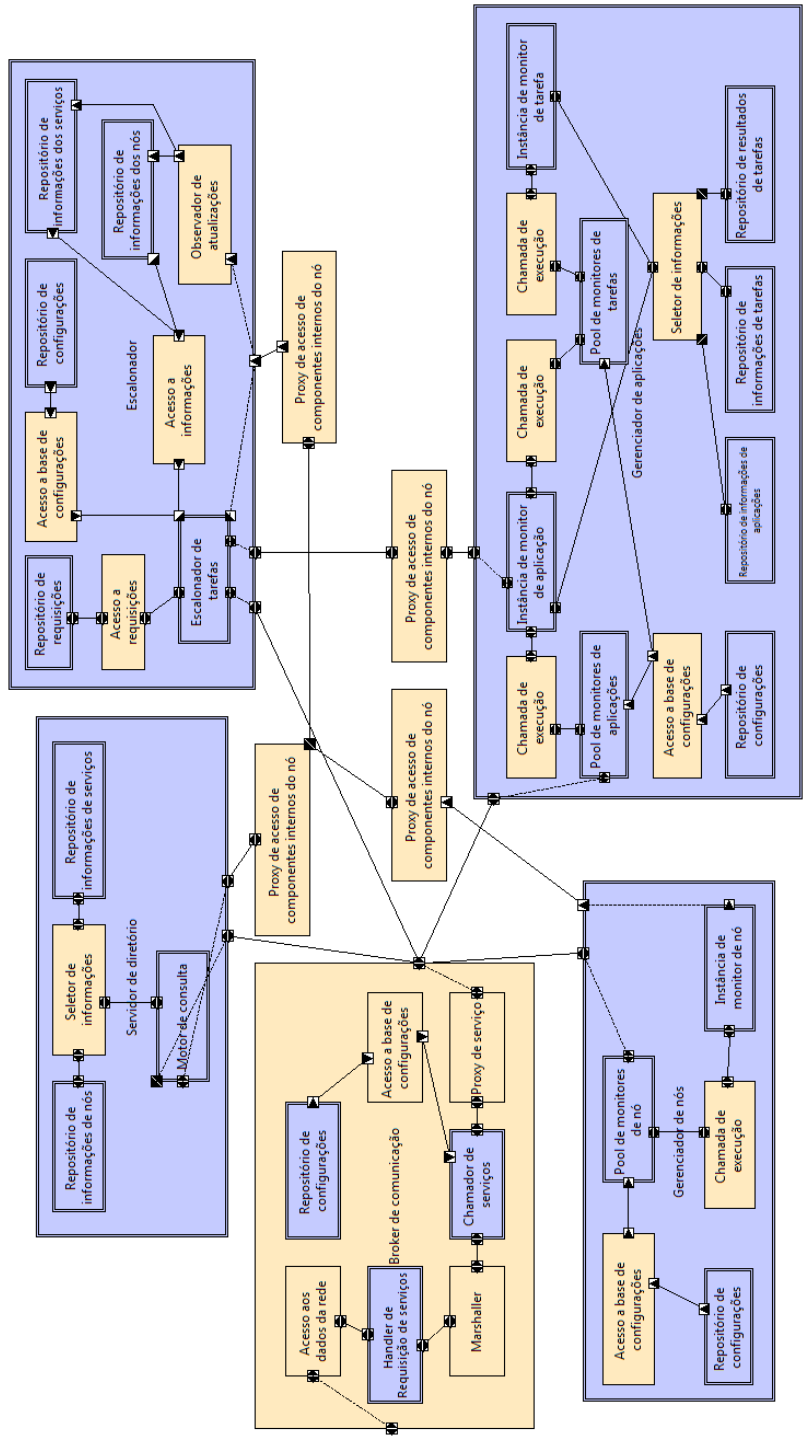

Figura 3.15: Arquitetura dos componentes dos serviços de coordenação do *grid*.

3.2.2.1 Servidor de Diretório

O Servidor de diretório está explanado na Figura 3.16. Ele é responsável por reunir todas as informações necessárias sobre os nós que estão no *grid* e os serviços disponibilizados através dele.

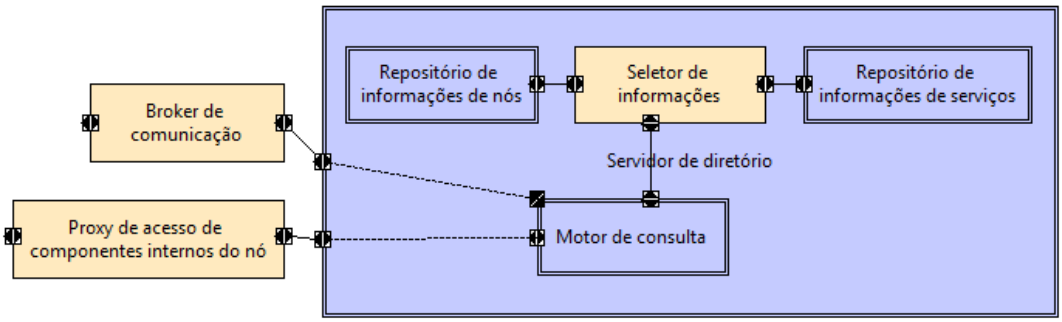

Figura 3.16: Arquitetura do Servidor de diretório do Gringa.

O seu elemento principal é o componente "Motor de consultas", que acessa dois repositórios através do componente seletor de informações: um "Repositório com informações de nós" e o "Repositório de informações de serviços".

Uma diferença desse componente em relação aos anteriores é que ele não é acessado somente através de solicitações de serviços externos. Ele também é utilizado por componentes internos do nó coordenador através do conector "*Proxy* de acesso a componentes internos do nó". Isso ocorre porque o Servidor de Diretório é atualizado pelo Gerenciador de Nós e é consultado pelo Escalonador de tarefas para utilização em seus algoritmos, ambos componentes integrantes do nó coordenador.

3.2.2.2 Gerenciador de Aplicações

A arquitetura do Gerenciador de aplicações pode ser vista na Figura 3.17. Ele tem o papel de monitorar as tarefas das aplicações que estão sendo executadas no *grid*.

Entretanto, como o próprio nome já diz, o papel desse *pool* é apenas de monitorar as tarefas em execução de uma determinada aplicação. É importante dizer que no modelo adotado pelo Gringa, cada cliente pode executar várias aplicações e cada aplicação pode requisitar a execução de várias tarefas, tendo assim que uma tarefa deve informar qual a aplicação a que está vinculada.

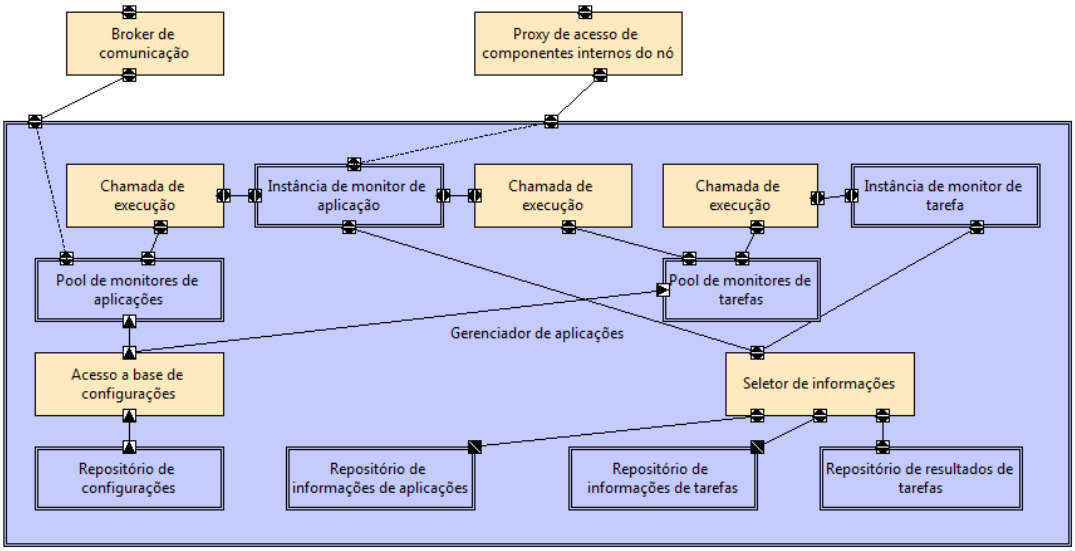

Figura 3.17: Arquitetura do Gerenciador de aplicações.

Para isso cada monitor, representado pelo componente "Instância de monitor de aplicações" controla um outro *pool*, que monitora as tarefas solicitadas. Cada "Instância de monitor de tarefas" controla três repositórios: um repositório com informações das tarefas, seu estado, quem está executando, *timeout*, entre outros; um repositório no qual se armazena temporariamente os resultados das tarefas enviados pelos nós que estavam executando aquele serviço, até que a aplicação cliente solicite a resposta; e outro com informações da própria aplicação, quantas tarefas possui, quem é o cliente, *timeout*, etc.

Esses repositórios também são acessados pelos monitores de aplicação, porque estes são responsáveis por interagir com o Escalonador para receber as informações das tarefas escalonadas e para solicitar um novo escalonamento caso haja problemas de remoção de nós ou estouro de tempo.

3.2.2.3 Gerenciador de Nós

Em um ambiente residencial com um *grid* oportunista que é o cenário de utilização do Gringa, a mobilidade de um nó deixando-o fora da rede ou o desligamento de um equipamento, de forma intencional ou não, pode ocorrer a qualquer instante. Por exemplo, alguém sai de casa levando o *smartphone* ou desliga a TV após assistir a programação de seu interesse. "O Gerenciador de nós", que está mostrado na Figura 3.18, tem o papel de monitoramento dos nós do *grid*.

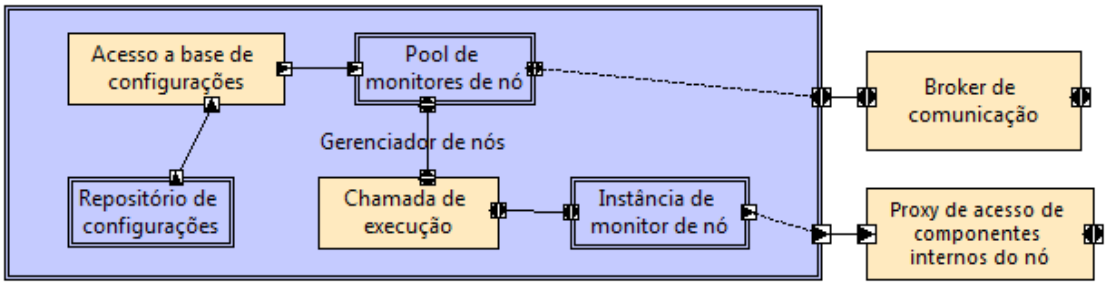

Figura 3.18: Arquitetura do Gerenciador de nós.

Para cada nó do *grid*, o Gerenciador de Nós mantém uma "Instância de Monitor de Nó". Ele também usa um *pool* para gerenciar um conjunto de instâncias, que segue o mesmo molde dos outros componentes citados anteriormente, isto é, as instâncias são alocadas conforme a demanda e são controladas por um ciclo de vida.

O repositório de configurações é usado para acessar definições de regras para entrada de nós no *grid* e para controlar o ciclo de vida dos nós. O ciclo de vida é mantido a partir de mensagens de atualização enviadas pelo serviço de informações dos nós provedores de serviço de forma periódica. Cada vez que recebe estas informações o gerenciador de nós as envia ao Servidor de Diretórios, que as atualiza e torna disponíveis para os demais componentes do coordenador.

Quando um nó é removido, seja a saída voluntária ou não, o Gerenciador de Nós interage com o Gerenciador de Aplicações, que vai verificar que tarefas estavam sendo executadas para que seja feito um reescalonamento. Essa interação é feita diretamente por cada instância de monitor através do conector "Proxy de Acesso de Componentes Internos do Nó".

3.2.2.4 Escalonador

O escalonador é o componente responsável por receber as requisições para execução de tarefas nos nós e, a partir de informações obtidas no Servidor de Diretórios, escolher qual o melhor nó para executar aquela tarefa. A arquitetura do Escalonador é mostrada na Figura 3.19 e o componente "Escalonador de tarefas" é o componente central desse serviço.

Para definir seu comportamento, o Escalonador acessa o repositório de configurações para o serviço, que contém entre outras coisas a decisão sobre qual algoritmo será utilizado e quais os parâmetros passados para que ele possa executar. Além

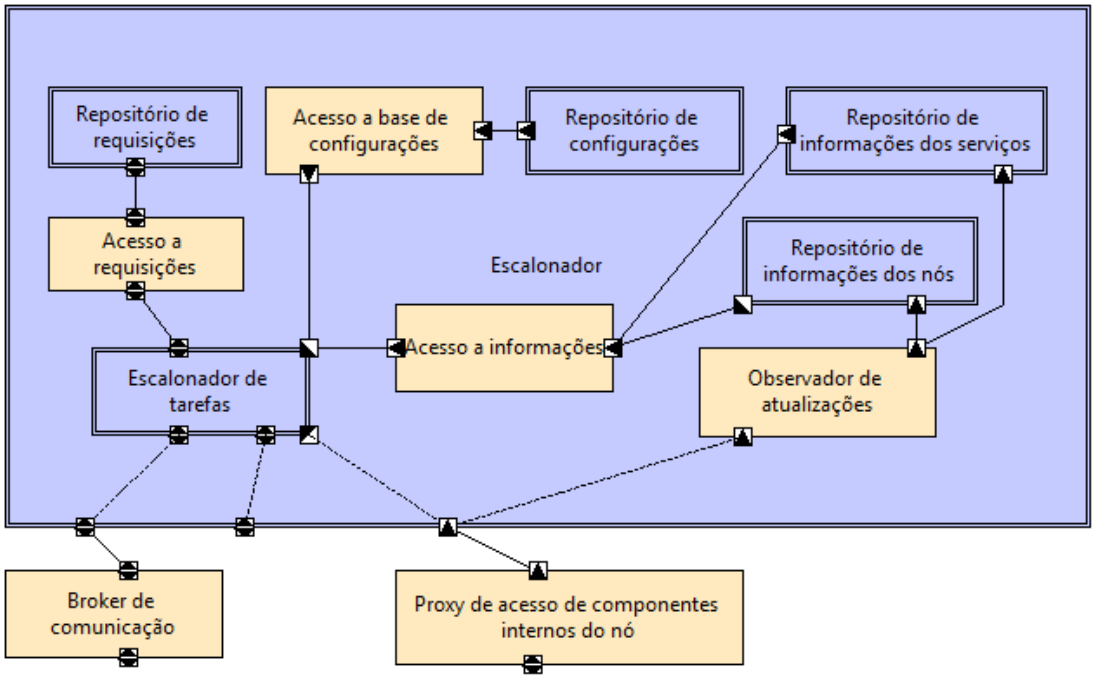

Figura 3.19: Arquitetura do componente Escalonador.

disso, o Escalonador também mantém um "Repositório de solicitações" contendo todos os pedidos de tarefas vindos das aplicações clientes ou de outros nós. A estrutura de dados na qual essas solicitações são armazenadas dependem da técnica de escalonamento utilizada.

O Escalonador separa nesse repositório as tarefas que tem nós disponíveis para a execução daqueles que não possuem, isto é, só concorrem ao escalonamento tarefas para as quais existem nós que estão ofertando o serviço. Se um nó deixa de executar uma tarefa ou demora muito para dar a resposta, o Gerenciador de Aplicações do nó manda uma requisição para reescalonar essas tarefas.

Além disso, o Escalonador mantém um repositório com informações específicas usadas no escalonamento sobre os nós (medidas de desempenho) e sobre as tarefas (definição e disponibilidade das mesmas nos nós). Para não sobrecarregar a todo momento o serviço de diretórios, o Escalonador possui um conector para acesso interno através do qual essas informações são monitoradas pelo "Observador de atualizações", e os repositórios internos só são atualizados quando alguma informação usada pelo escalonamento é alterada.

Capítulo 4

Implementação de Referência

Neste capítulo apresentaremos detalhes da implementação do *middleware Gringa* a partir das especificações mostradas no capítulo anterior.

4.1 Descrição da Implementação de Referência

A implementação de referência do Gringa foi projetada em três tipos de equipamentos diferentes que poderiam fazer parte de uma rede local residencial: computadores pessoais (*desktops* e *notebooks*), dispositivos móveis (*smartphones* e *tablets*) e STB de TV Digital Interativa com suporte a Ginga. Também foi levado em consideração na implementação os principais tipos de conexões utilizadas por estes dispositivos: conexão TCP/IP (por rede cabeada ou sem fio com *WiFi*) e transferências via *Bluetooth*.

Dessa forma, o primeiro passo foi escolher uma plataforma para execução de serviços que fosse flexível o suficiente para rodar em dispositivos com estes requisitos e dentre as pesquisadas a escolhida foi a XML-RPC[1]. Conforme fala (RIVERA et al., 2005), trata-se de uma especificação simples e padronizada de chamadas remotas de procedimentos implementadas em uma linguagem XML geralmente transportada através de protocolo HTTP. A simplicidade é a grande vantagem do XML-RPC, uma vez que ela possui suporte a tipos de dados simples (inteiros, lógicos, reais e strings) e compostos (vetores e estruturas). Há implementações para as linguagens mais populares usadas atualmente, além de ser de fácil implementação, pois seu padrão

[1] http://www.xmlrpc.com

não ocupa mais de 10 páginas, compensando *overheads* provocados pelo transporte e decodificação das mensagens em XML/HTTP. Além disso, o XML-RPC também suporta reflexão e os serviços podem ser auto-descritos.

Com relação ao desempenho, o trabalho de (JAGANNADHAM et al., 2007) mostra uma comparação entre tecnologias para construção de programas distribuídos em Java. Em testes realizados em redes locais viu-se que o desempenho do XML-RPC era aproximadamente 2,5 vezes melhor que o JAX-WS, que é uma implementação Java para *Web Services*, o que poderia atenuar os problemas descritos para nós móveis de *grids* como relatados em (LI et al., 2009).

Pensando na portabilidade do *middleware*, o Gringa foi implementado usando a linguagem Java para codificação dos componentes acessados pelos serviços. Sendo assim, foram escolhidas duas implementações de XML-RPC para Java a serem usadas:

- *Apache XML-RPC*[2]: uma implementação robusta criada pela Fundação Apache para *desktops* que é compatível com o padrão original do XML-RPC, mas que também dá suporte a extensões. Além disso ele já vem com um micro servidor web implementado que pode ser facilmente usado.

- *KXML-RPC*[3]: Uma implementação leve de um cliente XML-RPC para dispositivos móveis com suporte a JavaME, utilizando o parser Kxml;

Já para o STB, foi utilizada a implementação *Lua XML-RPC*[4] para a linguagem Lua a ser usada junto com o NCL e KXML-RPC para Ginga-J. Todas essas implementações possuem código aberto, o que foi essencial uma vez que alguns códigos tiveram que ser modificados. Para o *KXML-RPC* foi adicionado o suporte para *Bluetooth* que ele não possuía. Já para o *Lua XML-RPC* foi alterado o analisador sintático (*parser*), uma vez que ele usava como padrão o *XPath* que não está incluído no padrão Ginga. Todos os códigos do Gringa estão disponíveis através do software de controle *Subversion* no endereço *svn://di.uern.br/gringa*.

No que tange a segurança da informação, o padrão XML-RPC não oferece qualquer mecanismo de confidencialidade, integridade ou autenticidade. Isso não significa que toda comunicação XML-RPC é insegura, mas sim que esses aspectos não

[2]http://ws.apache.org/xmlrpc/
[3]http://sourceforge.net/projects/kxmlrpc/
[4]http://keplerproject.github.com/lua-xmlrpc/

constam no envelope do serviço. No caso, essas funções são repassadas geralmente para os contêineres dos serviços, como servidores de aplicação e, mais comumente, os servidores web que recebem as requisições HTTP. No Apache XML-RPC por exemplo, é possível usar certificados digitais e realizar comunicação via protocolo HTTPS, além de também ser permitida a criação de usuários para acesso ao servidor web.

4.2 Implementação de Aplicações Gringa Cliente

A construção de uma aplicação cliente para o Gringa pode ser feita chamando diretamente os serviços através do XML-RPC. Entretanto, para facilitar a construção de aplicações clientes para o Gringa há um *framework* Gringa Cliente, onde foram criadas em Java uma interface chamada *GringaClientConfig* e uma classe abstrata chamada *GringaApplication*. A primeira, que herda da interface *Map*, é usada para definir as propriedades a serem atribuídas às chamadas da aplicação cliente, e a segunda usada para abstrair as chamadas para a execução de serviços.

Para o XML-RPC foram criadas as classes *XmlRpcGringaClientConfig* e *XmlRpcGringaApplication* que implementam os métodos necessários para essa tecnologia. A Figura 4.1 mostra os métodos existentes na classe *GringaApplication*.

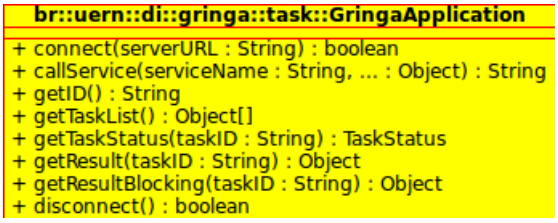

Figura 4.1: Classe GringaApplication.

Os métodos existentes na classe GringaApplication possibilitam através de uma chamada simples, sem conhecer a tecnologia orientada a serviços utilizada na construção do *middleware*, funcionalidades para conexão e desconexão, chamada a um serviço/tarefa, além de obtenção de valores como ID da aplicação, lista de tarefas entre outros. A simplicidade dessa classe pode ser vista no código código de uma aplicação simples usando estas classes, mostrado na Figura 4.2.

```
1  // Importando classes necessárias
2  import br.uern.di.gringa.task.TaskStatus;
3  import br.uern.di.gringa.xmlrpc.XmlRcpGringaApplication;
4  import br.uern.di.gringa.xmlrpc.XmlRpcGringaClientConfig;
5  import java.io.IOException;
6  public class Hello {
7      public void sayHello(String coordinatorURL) throws IOException {
8          // Inicializando as configurações a partir de um arquivo
9          XmlRpcGringaClientConfig config = new XmlRpcGringaClientConfig();
10         config.loadConfig("clientconf.properties");
11         // Criando uma nova aplicação
12         XmlRcpGringaApplication app = new XmlRcpGringaApplication(config);
13         app.connect(coordinatorURL);
14         // Invocando um serviço
15         String taskID = app.callService("HelloWorld", "Hello");
16         // Verificando seu status e pegando o resultado
17         String resultado=null;
18         TaskStatus status = app.getTaskStatus(taskID);
19         if(status == TaskStatus.DONE)
20             resultado = (String) app.getResult(taskID);
21         else
22             resultado = (String) app.getResultBlocking(taskID);
23         // Encerrando a aplicação
24         app.disconnect();
25     }
26 }
27
```

Figura 4.2: Exemplo de aplicação cliente Gringa.

Nas linhas de 2 a 5 são importadas as classes de aplicação e configuração, além da classe *TaskStatus*, que é uma enumeração contendo os possíveis estados das tarefas conforme exposto na Figura 3.8. Nas linhas 9 e 10 é feita a criação de uma variável de configuração cujas definições estão no arquivo "clientconf.properties", que segue o padrão de arquivos de propriedades conhecido no Java. Nas linhas 12 e 13 é feita a criação de uma nova aplicação para o *grid*, cuja URL do coordenador é definida pela variável *coordinatorURL*. Na linha 15 é feita uma chamada a uma tarefa do serviço "HelloWord.hello". Na linha 18 é realizada uma consulta ao status da tarefa cujo resultado é testado na linha 19. Se o resultado for "DONE" significa que a tarefa já foi executada e o cliente pode acessar a resposta da execução do serviço através do método "getResult". Se o resultado não estiver pronto, é feita a chamada na linha 22 por "getResultBlocking" onde a chamada de serviço ficará bloqueada até que o resultado seja devolvido. Não são feitos quaisquer tratamentos de erros no código, mas vale ressaltar que qualquer chamada de serviço ou obtenção de resultado que não obtiver êxito retornará valor nulo.

Já para a linguagem Lua usada em aplicações Ginga-NCL, a implementação é feita em um módulo chamado "gringa-xmlrpc" que contem chamadas de métodos semelhantes aos de Java, com a diferença que os tipos são mapeados em tabelas Lua pelo *Lua XML-RPC*. Sendo assim, por exemplo,uma classe Java é mapeada em uma *struct* da especificação XML-RPC e é decodificada como uma tabela nativa do Lua.

Os códigos das aplicações Lua usadas para os testes de performance do Gringa com o módulo gringa-xmlrpc estão nos apêndices A e B.

4.3 Implementação de Novos Serviços Gringa para os Nós

A implementação de serviços Gringa é geralmente simples, bastando que seja usado um *parser* no arquivo da chamada remota comparando-o com os serviços disponíveis localmente. Com as implementações Java com XML-RPC não é diferente, para disponibilizar um serviço é preciso fazer um *bind* entre um identificador para o serviço e o nome da classe que o representa. Com isso, utiliza-se o mecanismo de reflexão do Java para verificar se o método invocado existe para aquela classe ou não.

Isso permite, por exemplo, que uma definição de um serviço possa ser enviada em forma de um arquivo .class ou .jar pela rede através do serviço de transferência de dados e carregado dinamicamente. Entretanto, para manter um padrão nas tarefas e criar um contrato padrão entre as tarefas que vão ser escalonadas no *grid*, foi criada uma interface chamada *GridTask* que deve ser implementada por classes de serviços.

A Figura 4.3 mostra a estrutura da interface. Para facilitar ainda a implementação dos serviços, foi criada uma classe abstrata chamada *GringaTask* que implementa de forma básica essas funções da interface *GridTask*.

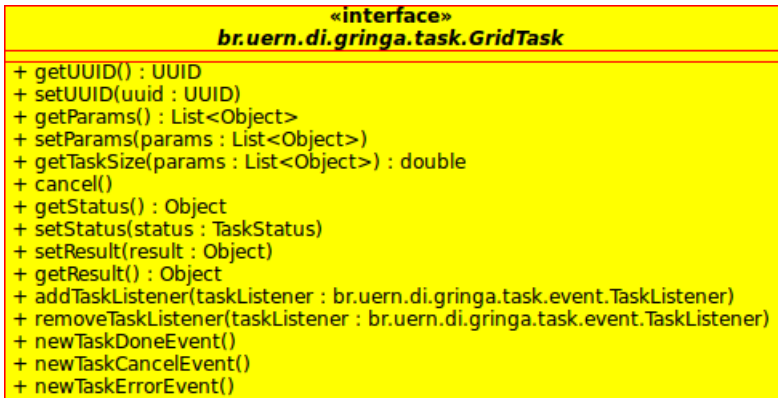

Figura 4.3: Estrutura da classe GridTask

Basicamente, a interface *GridTask* herda das interfaces *Runnable* e *Serializable* de forma que as tarefas possam ser executadas de forma concorrente, permitindo

a execução de várias tarefas por vez, e que os dados recebidos como parâmetros e enviados como resultado possam ser transformados em vetores de bytes e enviados pela rede, através do processo de serialização. Cada tarefa tem um identificador único implementado como objeto da classe *universally unique identifier* (UUID), e métodos para acesso e alteração(*get/set*) para manipular os parâmetros de entrada, o resultado e o estado atual da tarefa. Além disso, também é importante mencionar os métodos que manipulam os chamados *TaskListeners*, que representam objetos monitores de execução de tarefas, e que devem ser informados quando uma tarefa foi cancelada, concluída ou teve falha na execução.

Por fim, temos o método *getTaskSize* que é usado pelo algoritmo de escalonamento para tentar atribuir as maiores tarefas, isto é, tarefas que necessitam de maior carga de processamento (medidas em número de operações), aos nós com maior desempenho. Ele deve ser implementado pelo desenvolvedor do serviço de forma que devolva uma estimativa da quantidade de operações que serão executadas para os parâmetros especificados, como uma espécie de medição da complexidade do algoritmo de resolução da tarefa. Caso não seja possível medir ou o programador do serviço não saiba implementar esta função, ela devolverá como valor padrão o tamanho dos parâmetros recebidos.

A Figura 4.4 mostra a implementação do serviço que é solicitado no código da Figura 4.2.

```
1  package br.uern.di.gringa.samples;
2  // Importando classes necessárias
3  import br.uern.di.gringa.task.GringaTask;
4  import java.util.List;
5  // Classe herda de GringaTask ou implementa GridTask
6  public class HelloWorld extends GringaTask{
7      // É obrigatório a implementação do método run
8      @Override
9      public void run() {
10         // Pega os parâmetros enviados na chamada
11         List parameters = getParams();
12         // Testa se só há um parâmetro e ele é "hello"
13         // caso não seja retorna um evento de erro
14         if ( parameters.size()!=1 || !parameters.get(0).equals("hello"))
15             newTaskErrorEvent();
16         // Caso seja coloca-se "world" como resultado e diz que a tarefa
17         // foi concluída
18         else{
19             setResult("world");
20             newTaskDoneEvent();
21         }
22     }
23 }
```

Figura 4.4: Implementação de um exemplo de serviço Gringa.

Na linha 8 vemos o código em que a classe herda de *GringaTask*. A implementação do método *run* é obrigatória por causa da interface *Serializable*. Na linha 14 usa-se o método *getParams* para obter os parâmetros enviados na chamada e posteriormente testá-lo na linha 17, verificando se só há um parâmetro e ele tem o valor "hello". Caso isso não ocorra é chamado o método *newTaskErrorEvent* que altera o estado da tarefa para erro e notifica os interessados do erro. Se tudo estiver correto, é atribuído o valor "world" como retorno e é chamado o método *newTaskDoneEvent* para notificar que a tarefa está concluída.

4.4 Implementação dos Serviços Básicos e de Coordenação

A implementação dos nós de execução e de coordenação foi realizada conforme a arquitetura descrita na subseção 3.1.2 e organizada em vários pacotes. Esses pacotes, com classes, enumerações e interfaces abstraem as chamadas aos serviços XML-RPC entre os dispositivos. O apêndice C traz um catálogo com os serviços implementados mais utilizados no Gringa, contendo sua identificação e sua função.

A seguir estão relacionados os pacotes que são comuns tanto aos clientes, nós e coordenador, sendo eles:

- *br.uern.di.gringa.console* - Contém aplicativos com interface gráfica com o usuário que servem para inicializar nós coordenadores e de execução, acompanhar sua execução e configurar o gringa para o equipamento;
- *br.uern.di.gringa.util* - Classes úteis para os mais diversos fins, como manipulação de identificadores UUID por exemplo;
- *br.uern.di.gringa.task* e *br.uern.di.gringa.task.event* - Usadas para requisitar e monitorar tarefas do *grid*;
- *br.uern.di.gringa.xmlrpc* - Classes que usam as funcionalidades do XML-RPC implementadas para dar vida ao *grid*, usando o *broker* de serviços XML-RPC para codificar/decodificar chamadas de serviços.

A seguir os pacotes que possuem classes usadas nos nós de execução de serviços. São eles:

- *br.uern.di.gringa.benchmarks* - Possui as classes usadas para fazer os testes de medição de desempenho dos equipamentos. Atualmente o Gringa usa como padrão o algoritmo de Linpack (DONGARRA et al., 2003) para testes de processamento e o Ping-pong (LUSZCZEK et al., 2005) para medir taxa de transferência pela rede;

- *br.uern.di.gringa.nodo* - Possui classes que auxiliam e controlam a execução das tarefas no nó (*desktop*);

- *br.uern.di.gringa.nodome* - Tem a mesma funcionalidade do anterior, só que foi projetado para dispositivos móveis que usam Java ME;

- *br.uern.di.gringa.nodo.monitoring* - Conta com classes para realizar o monitoramento de nível de bateria utilizadas pelos aparelhos celulares, notebooks e outros dispositivos com autonomia de energia limitada;

- *org.kxml2* e *org.kxmlrpc* - São adaptações realizadas no código KXML-RPC, para adicionar o suporte à execução de serviços em nó móvel, que não estão disponíveis na versão original do pacote.

- *br.uern.di.gringa.kbxmlrpc.task* - Possui classes para dar suporte à submissão de tarefas em nós que usam o *Gringa* com conexões *Bluetooth*.

Os seguintes pacotes são usados apenas por nós coordenadores. Suas funcionalidades são:

- *br.uern.di.gringa.kbxmlrpc* - dão suporte ao acesso do coordenador a dispositivos que ofertam serviços através do protocolo *Bluetooth*;

- *br.uern.di.gringa.nclbridge* - Conforme explicado na seção 2.1, a norma Ginga-NCL traz um subconjunto da linguagem Lua para dar suporte à execução de scripts junto ao NCL. Por este padrão a conectividade do NCL se faz apenas através de eventos da classe *tcp*. Esse pacote possui classes para dar suporte a essa comunicação a mais baixo nível, funcionando como uma ponte de acesso para que aplicações NCLua padrão acessem o grid.

- *br.uern.di.gringa.naming* - Contém classes usadas pelo serviço de diretórios. Para esta implementação do Gringa, o serviço de diretórios foi implementado usando o Java Naming and Directory Interface (JNDI[5]) armazenando os dados

[5]http://www.oracle.com/technetwork/java/jndi/index.html

dos nós na memória, por estarem em constante processo de mudança, e em arquivo a configuração/descrição dos serviços.

- *br.uern.di.gringa.server* - por fim, este pacote congrega os principais serviços oferecidos pelo nó coordenador como o modelo de criação de instâncias de monitoramento das aplicações, algoritmo de escalonamento e monitoramento de nós.

No pacote *server*, há uma classe principal chamada *GringaServer* que instancia os demais serviços administrativos. Alguns serviços, como o de escalonamento, por exemplo, podem ser facilmente alterados, pois funcionam como o padrão de projeto *Command*, isto é, devem implementar um método de execução implementando uma interface de contrato. No caso do escalonamento, isto é feito através da interface *Scheduler* que possui o método *submit* abstrato para implementação, devendo receber como parâmetro uma lista de tarefas e dar suporte à notificação de outros componentes através da interface *TaskListener*.

4.4.1 Escalonamento no Gringa

O algoritmo de escalonamento padrão do Gringa é o algoritmo *Power-aware User-preference Task-size based Schedulling* (PUTS) proposto no trabalho de (DANTAS, 2011). Ele faz modificações no algoritmo de escalonamento DFPLT (*Dynamic Fastest Processor to Largest Task*) que atribui a maior tarefa ao processador mais rápido naquele momento, acrescentando ao tempo estimado para a conclusão de cada tarefa a taxa de transferência da tarefa na rede. Além disso, o algoritmo aplica penalidades de acordo com um fator de perda de nível de energia (medidos em dispositivos como celulares e notebooks) e parâmetros de preferência do usuário pela execução de tarefas naquele nó, ou até mesmo de serviços específicos. Com isso, o custo computacional de uma tarefa(CT) para um determinado nó é calculado de acordo com a seguinte fórmula:

$$CT = (TTD + TT * \frac{1}{NBE}) * FPU$$

TT representa o tempo da tarefa que é dado pela soma do tempo de transferência (levando em consideração o tamanho da entrada) e o tempo estimado de execução (baseado no tamanho da tarefa que é definido pelo desenvolvedor do serviço conforme descrito na seção 4.3 e pela estimativa de capacidade de processamento dada pelo algoritmo de benchmark). Esse tempo sofre uma penalidade calculada de acordo com

o nível de bateria estimado (NBE), calculado de acordo com o tempo de execução da tarefa e o fator de perda de energia que é calculado ao longo do tempo. Por exemplo, se o nível de bateria for de 40% a razão 1/0.4 multiplicará o tempo da tarefa por 2,5 fazendo com que o nó seja menos requerido, mesmo que tenha melhor desempenho.

Esse tempo da tarefa é somado ao tempo para tornar-se disponível (TTD) que é uma variável que acumula o tempo estimado de cada tarefa que o nó já está executando. Por fim a soma desses tempos é multiplicada por um fator de preferência do usuário (FPU) calculado com base em duas medidas informadas na configuração do nó: a disponibilidade do nó (de 0 a 100%) e a disponibilidade de uma certa tarefa naquele nó (também de 0 a 100%). Quanto menor o fator de disponibilidade, maior será a penalidade que vai multiplicar o tempo total para completar a tarefa(CT) e, consequentemente o nó terá menos chances de ser escolhido para a executar a tarefa.

Capítulo 5

Resultados e Discussão

Neste capítulo falaremos sobre os testes realizados para aferir a eficiência do Gringa. Foram feitos vários testes com aplicações com características distintas tanto em comunicação quanto em carga computacional. Para simular um ambiente doméstico foram montados cenários utilizando um STB de TV Digital, dois *smartphones*, dois *notebooks* e dois *desktops*. Eles foram interligados através de rede cabeada (STB e *desktops*) e *Wifi* (*smartphones* e *notebooks*) de forma que eles pudessem ser acessados diretamente através de um endereço IP. As configurações de *hardware* de cada dispositivo estão listados na Tabela 5.1.

Foram realizados diversos testes de comunicação com os equipamentos nas mais diversas situações, como: queda da rede, alto uso de CPU nos *desktops* durante execução de tarefas, recebimento de ligações telefônicas enquanto haviam tarefas executando nos *smartphones*, entre outros. O Gringa comportou-se conforme o esperado, excluindo nós que não atualizavam informações e reescalonando tarefas que estavam neles. Já quando há a queda do coordenador, e com isso o estouro do tempo de resposta, os clientes e os nós cancelam automaticamente suas solicitações, tentando, posteriormente, conectar o coordenador novamente. Caso não obtenham resposta após um número determinado de tentativas, os nós e a aplicação cliente finalizam sua execução.

Outro aspecto relevante a se comentar é que, como o STB utilizado não possuía conexão *bluetooth*, houve o acréscimo de um serviço adicional em alguns nós para que os mesmos atuassem como *proxy*. Esses nós recebiam tráfegos via *bluetooth*, e usavam a "API de Acesso a Aplicações de TVDI Ginga", conforme já explicado através da Figura 3.4.

Tabela 5.1: Dispositivos utilizados nos experimentos de desempenho do Gringa

Dispositivo	Descrição	Recursos de hardware
Settop Box (STB)	Conversor Digital Zinwell	Processador Bcrm 295MHz, 128Mb DDR - USB-Disk 4Gb
Smartphone 1 (S1)	Nokia N85	Processador ARM 369MHz, 128Mb SDRAM, Micro SD 8Gb
Smartphone 2 (S2)	Nokia 5530 Xpress-Music	Processador ARM 434MHz, 128 Mb SDRAM, Micro SD 4Gb
Notebook 1 (N1)	Netbook HP Mini 110 1150BR	Processador Intel Atom 1.6GHz, 2Gb DDR3, HD Sata2 - 250Gb
Notebook 2 (N2)	Notebook HP Probook 6445b	Processador AMD Turion M2 2.3GHz, 2Gb DDR2, HD Sata2 - 250Gb
Desktops (D1 e D2)	Desktop HP	Processador AMD Athlon II X2 2.8GHz, 2Gb DDR2, HD Sata2 - 250Gb

Nas próximas seções serão mostrados os dois experimentos realizados para medir o desempenho de *grids* usando o Gringa. Para isso, foram usadas aplicações de uma classe denominada por (PENG et al., 2004) como *Embarrassingly Distributed* e também conhecida como *Bag of Tasks*. Nesses tipos de aplicações pode-se dividir o processamento em várias tarefas independentes, isto é, que não trocam dados entre si e nem dependem dos resultados das outras para realizar sua execução.

5.1 Experimento 1: Multiplicação de Matrizes

Esse é um experimento clássico para medir o desempenho de aplicações paralelas. Para realizar a multiplicação de duas matrizes, uma delas, geralmente a maior, é dividida em grupos de linhas. Cada grupo é enviado com a segunda matriz inteira e o resultado de cada multiplicação resulta em submatrizes com linhas da matriz resultado.

No exemplo mostrado na equação 5.1 a primeira matriz é dividida em três conjuntos, com uma, duas e três linhas, respectivamente. No caso, em cada tarefa um desses conjuntos é enviado junto com a segunda matriz inteira, para que a multiplicação seja feita pelos nós do *grid* e os resultados são devolvidos ao cliente

para compor o resultado final. A aplicação cliente Gringa testada foi desenvolvida através de um *script* Lua, que pode ser chamado através de um documento NCL, que é mostrado no apêndice A.

$$\left(\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \end{array}\right) \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix} \tag{5.1}$$

As matrizes utilizadas nos experimentos tinham dimensões de 400x64 e 64x400. Para cada cenário foi usado o mesmo caso de teste composto por 10 arquivos com linhas da primeira matriz, tendo em cada um número diferente de linhas que variava de 1 a 122. Essa divisão em arquivos com tamanhos diferentes impactou diretamente na distribuição das tarefas pelos nós, uma vez que o cálculo do custo da tarefa é dado pela fórmula:

$$CustoMMatriz = Linhas_{M1} * Colunas_{M2} * (2 * Colunas_{M1} - 1) \tag{5.2}$$

O custo é estimado em termo de número de operações de ponto flutuante. Isso significa que o algoritmo de escalonamento poderia encontrar o melhor balanceamento de carga e atribuir tarefas menores a nós com menor capacidade de hardware.

Este caso de teste é classificado por (FALAVINHA JUNIOR et al., 2007) como uma tarefa com pequena carga computacional e pequeno volume de dados. Mesmo assim, para dispositivos com baixo poder de hardware, como um STB ou um *smartphone*, elas podem demandar um tempo muito grande. A Tabela 5.2 mostra uma média de tempo de execução local dessa multiplicação a partir da leitura de dois arquivos texto com os conteúdos das matrizes a serem multiplicadas e a criação de um arquivo com o resultado. Nela podemos perceber como é extremamente grande a diferença de desempenho entre um *desktop* e os demais. Apesar de terem processadores com frequências relativamente próximas, a diferença entre o STB e os dois *smartphones* também foi considerável.

Sendo assim, para mostrar melhoria no tempo da execução dessa tarefa de multiplicação de matrizes de acordo com a capacidade dos dispositivos, foram montados

Tabela 5.2: Tempo de execução de uma multiplicação de matrizes localmente

Dispositivo	Tempo médio (segundos)	Implementação
STB	1623,3	Script Lua chamado a partir de documento NCL
S1	155,2	Java ME CLDC 1.1 MIDP 2.0
S2	31,7	Java ME CLDC 1.1 MIDP 2.0
N1	1,6	Java SE 6
N2	0,5	Java SE 6
D1/D2	0,2	Java SE 6

5 cenários. Em todos o *notebook* N1 assumiu o papel de coordenador do *grid* e o STB de cliente, enquanto os demais nós continham apenas o serviço necessário para a multiplicação de submatrizes, conforme mostra a Figura 5.1. Os cenários de testes foram montados da seguinte forma:

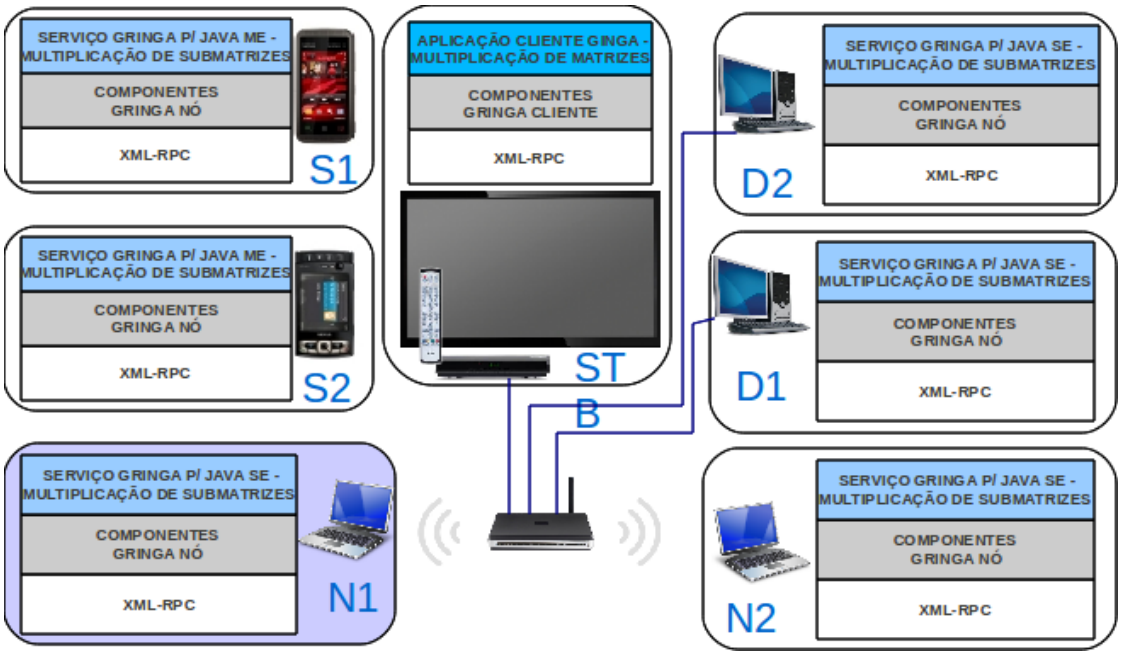

Figura 5.1: Disposição de componentes e papéis para cada dispositivo no experimento 1.

- Cenário 1: o *netbook* N1 foi iniciado como coordenador e os dois *smartphones* (S1 e S2) como nó oferecendo o serviço;
- Cenário 2: além da coordenação, N1 passou a oferecer também o serviço;

- Cenário 3: o *notebook* N2 foi adicionado;
- Cenário 4: o *desktop* D1 foi adicionado;
- Cenário 5: o *desktop* D2 foi adicionado;

5.2 Experimento 2: Conversão de Vídeo

Aplicações de conversão de vídeo são conhecidas por necessitarem uma grande quantidade de recursos computacionais, especialmente processamento. Nesse experimento temos uma aplicação que faz a conversão do formato de um vídeo do padrão H.264 em HDTV para um MPEG em SDTV, reduzindo o seu tamanho. O MPEG foi escolhido por ser popular e possuir suporte à reprodução em vários dispositivos de execução de arquivos multimídia. Já o padrão H.264 foi escolhido por duas razões.

Em primeiro lugar por ele ser o formato usado nas transmissões terrestres do SBTVD, o que possibilita que um vídeo possa ser extraído do sinal recebido pela TV possa ser convertido pelo *grid*. Além disso, o vídeo convertido poderia ser copiado para um computador ou um *smartphone* que não possuíssem suporte a reprodução de vídeos nesse formato, ou cuja restrição de espaço em disco induzisse o usuário a optar por reduzir o tamanho do arquivo.

A segunda razão é a forma como um vídeo H.264 é organizado. Conforme fala (RICHARDSON, 2010), cada sequência de um vídeo codificado em H.264 é separada em unidades de dados do tipo VCL (*Video Coding Layer*). Essas unidades VCL são empacotadas em unidades do tipo NAL (*Network Abstraction Layer*), que contém informações necessárias para a transmissão pela rede e armazenamento dos dados.

Um ponto interessante é que cada conjunto de unidades NAL que armazena uma sequência de vídeo não depende de informações das anteriores para ser decodificada. Com isso, um arquivo de vídeo H.264 pode ser decodificado de forma distribuída, dividindo-se o vídeo como mostrado na Figura 5.2, onde para cada conjunto o cabeçalho, em verde, é copiado antes do início dos dados.

No trabalho de (MORAIS, 2011) foi desenvolvida uma aplicação que separa o vídeo conforme citado, e também um serviço Gringa que converte o vídeo H.264 em MPEG. Para a implementação desse serviço de conversão foram usados o software

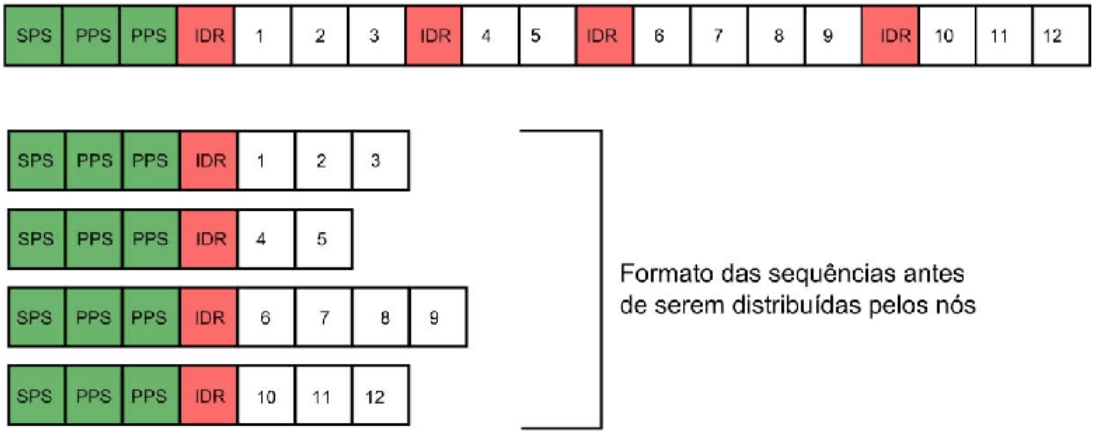

Figura 5.2: Separação de sequências de vídeo para conversão. (MORAIS, 2011)

livre FFMpeg[1] e a biblioteca JAVE[2].

Para esse experimento foi usado um vídeo de 113,6 MB que foi dividido em 69 sequências de tamanho variando entre 705 KB e 4,2 MB, o que (FALAVINHA JUNIOR et al., 2007) considera como tarefas de tamanho médio em relação tamanho dos dados e complexidade da tarefa.

Como os *smartphones* não possuíam o FFMpeg, eles não foram utilizados nesses testes. Dessa forma, foram montados apenas três cenários, nos quais o *desktop* D2 foi o coordenador do *grid* e o STB foi o cliente em todos, como mostra a Figura 5.3. A aplicação cliente foi desenvolvida em Lua e está no apêndice B.

Os cenários de testes foram montados da seguinte forma:

- Cenário 1: o *Desktop* D2 foi iniciado como coordenador e os dois *notebooks* (N1 e N2) como nó oferecendo o serviço;
- Cenário 2: o *desktop* D1 foi adicionado;
- Cenário 3: além da coordenação, neste cenário D2 passou a oferecer também o serviço;

Como não foi possível determinar com exatidão o número de operações realizadas pelo algoritmo de conversão, o tamanho de cada tarefa para o escalonador foi definido como o tamanho do vídeo a ser convertido.

[1] www.ffmpeg.org
[2] www.sauronsoftware.it/projects/jave/

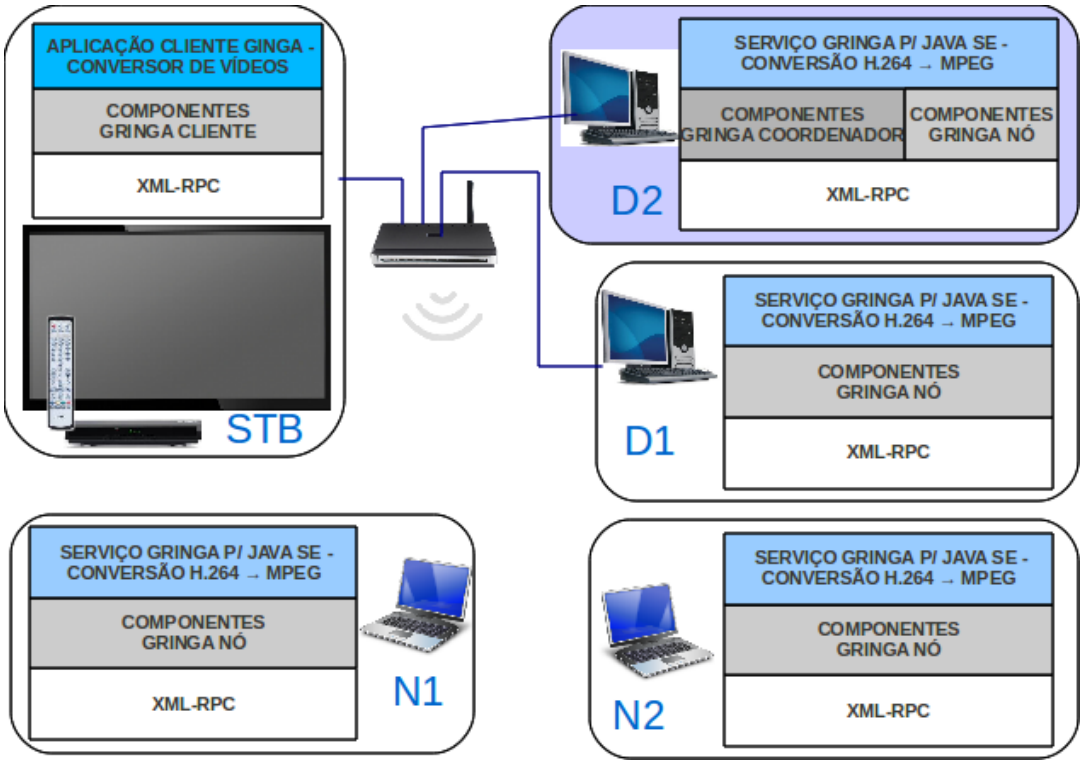

Figura 5.3: Disposição de componentes e papéis para cada dispositivo no experimento 2.

A Tabela 5.3 mostra os tempos obtidos para a conversão sequencial dos vídeos em cada dispositivo.

Tabela 5.3: Tempo de execução de uma conversão de vídeos localmente

Dispositivo	Tempo médio (segundos)	Implementação
N1	97,5	Java SE 6
N2	36,4	Java SE 6
D1/D2	22,6	Java SE 6

5.3 Métricas de Desempenho

Antes de mostrar os resultados veremos as métricas que foram usadas para verificar o desempenho dos *grids*. Para testarmos um *middleware* para *grids* de serviços podemos levar em conta diversos aspectos diferentes. O trabalho de (TRUONG et al., 2006), por exemplo, classificou métricas de Qualidade de Serviço (QoS) que

poderiam ser monitoradas e mensuradas. Essas métricas são dispostas em 4 eixos principais:

- *Performance*: indica o quão bem o serviço executou, baseando-se em medidas de tempo e taxas;
- Dependabilidade: diz se o *grid* é confiável, incluindo conceitos como disponibilidade, acurácia, gerenciabilidade e segurança, por exemplo;
- Configuração: mostra como o *grid* está organizado, que padrões usa, como é realizada a localização de serviços etc.;
- Métricas customizadas: que são específicas para cada serviço.

Alguns desses critérios não são relevantes ou aplicáveis em um cenário de TV Digital Interativa em um ambiente de rede local como é a proposta da dissertação. As métricas de configuração, por exemplo, são mais aplicáveis a *grids* de maior tamanho entre diferentes corporações, onde há um conceito forte de organizações virtuais.

Já para os requisitos de dependabilidade temos a seguinte situação com relação a cada um dos itens:

- Disponibilidade: Não foi realizada a medição de disponibilidade pois foram criados cenários diferentes, e em cada um assumimos que os equipamentos estariam conectados ao *grid* o tempo todo;
- Acessibilidade: como o gerenciador de nós verifica a permanência de um nó no *grid* e o gerenciador de aplicações reescalona tarefas cujos nós executores, a acessibilidade de um determinado serviço é garantido e testado no próprio processo de escalonamento das tarefas;
- Acurácia: a corretude do resultado devolvido depende do serviço implantado no nó. Como dois serviços não podem ter nomes e assinaturas iguais, essa possibilidade só ocorrerá se houverem serviços implantados em nós distintos que tenham o mesmo nome e a mesma assinatura e que tenham implementações diferentes;
- Confiabilidade: Como cada serviço, no caso dos implementados em Java, são *threads*, as falhas de execução de uma instância de tarefa não afetarão

diretamente as outras, e tem sua execução finalizadas quando ultrapassam o limite de tempo determinado nas configurações do Executor de Tarefas do Nó;

- Capacidade: a capacidade do *grid* ou de um nó é determinada pelo algoritmo de escalonamento. No caso do PUTS, algoritmo padrão do Gringa, a variável TDD da fórmula 4.4.1, que armazena uma estimativa de tempo para que o nó tenha processamento disponível, é definida em um parâmetro de configuração e cada vez que supera esse valor o nó é definido como indisponível até a conclusão de uma tarefa;

- Gerenciabilidade: relativo à taxa de sucesso de recuperação de um serviço após uma falha, esse parâmetro não é medido nem usado no escalonamento para o Gringa, mas poderá vir a ser incorporado a depender do modelo de gerenciamento dos nós adotado;

- Segurança: com já explicado, a segurança do Gringa não é feita a nível de mensagem de aplicação, mas sim a nível de contêiner de serviço(transporte). No caso do XML-RPC, a segurança pode ser feita na troca de mensagens com o servidor web, usando autenticação e criptografia usando HTTPS. Pode-se acrescentar um componente de segurança na borda do *broker* de serviços, mas isso poderia afetar o desempenho em equipamentos sem tantos recursos como os *smartphones* e os próprios STB's de TV Digital.

No que diz respeito às métricas de desempenho, foi visto que em outros trabalhos elas não são tomadas individualmente, mas formando outras métricas compostas. No trabalho de (MA et al., 2008) as métricas de desempenho são combinadas em outras, de onde destacamos duas principais:

- Tempo de resposta da aplicação (*makespan*): tempo decorrido entre o início da primeira e o fim da última tarefa em uma aplicação. Também pode ser combinado com o tempo de espera , isto é, o tempo entre o pedido de execução da primeira tarefa e o momento em que ela é submetida para um nó, formando assim o *makespan* total;

- *Throughput*: representa a razão entre a quantidade de tarefa concluídas pelo intervalo de tempo decorrido. Essa métrica pode ser usada para medir a vazão de computação de tarefas realizada pelo *grid* como um todo, assim como de um nó específico.

Já o artigo de (ZHANG; YAN, 2005) apresenta métricas para desempenho de aplicações paralelas onde as máquinas são heterogêneas. Duas métricas são usadas para cenários onde a aplicação testada tem um tamanho fixo e a cada teste o número de máquinas é incrementado, aumentando a capacidade de processamento. A primeira delas, chamada *Speedup*, é usada para quantificar o ganho de performance da computação paralela da aplicação em relação à computação sequencial em uma única máquina. Ele é calculado de acordo com dois fatores: o grau de paralelismo e a heterogeneidade.

O grau de paralelismo representa o efeito do balanceamento de carga entre os nós e é definido pela fórmula 5.3, onde m representa o número de máquinas disponíveis para computação e $Tempo\_ativo_i$ representa o tempo que cada máquina i ficou ocupado executando tarefas.

$$Grau\_de\_paralelismo = \frac{\sum_{i=1}^{m} Tempo\_ativo_i}{Makespan} \tag{5.3}$$

Já a Heterogeneidade dos equipamentos tem por objetivo medir a diferença relativa entre o equipamento mais rápido e o demais que estão na rede. Ela é dada na fórmula 5.4, no qual $Peso\_relativo_i$ é uma razão entre o tempo necessário para computar todas as tarefas da aplicação A sequencialmente no melhor equipamento e o tempo obtido no nó i ($Tempo\_sequencial_{melhor}/Tempo\_sequencial_i$).

$$H = \frac{\sum_{i=1}^{m}(1 - Peso\_relativo_i(A))}{m} \tag{5.4}$$

Com isso, podemos calcular o *speedup* como mostrado na fórmula 5.5.

$$Speedup(A) = Grau\_de\_paralelismo * (1 - Heterogeneidade) \tag{5.5}$$

A outra métrica usada por (ZHANG; YAN, 2005) para aplicações de tamanho fixo é a Eficiência. Ela mede o percentual de tempo no qual as máquinas foram empregados de forma útil na computação paralela. A fórmula 5.6 mostra o cálculo da eficiência, onde $Tempo\_ocupado_i$ representa o tempo de sobrecarga da máquina i com outras tarefas que não são da aplicação testada, no caso em que o nó não é dedicado. Não haviam outros aplicativos com uso de processamento relevante executando durante os testes, os resultados mostrados nessa dissertação não levarão

em consideração esse fator.

$$Eficiencia = \frac{\sum_{i=1}^{m}(Peso\_relativo_i(A) * Tempo\_ativo_i(A))}{\sum_{i=1}^{m}(Makespan - Tempo\_ocupado_i) * Peso\_relativo_i(A))} \quad (5.6)$$

Sendo assim, as seguintes métricas serão usadas para aferir o desempenho do Gringa:

- *Throughput* do *grid*;
- *Makespan* total da aplicação;
- *Speedup*;
- Eficiência;

Além disso, também verificaremos como foi a distribuição de tarefas e carga para cada nó.

5.4 Resultados dos Experimentos

Para nossos testes, cada experimento foi executado 30 vezes para cada cenário, dos quais foram tiradas as médias que serão apresentadas nessa seção. No início dos testes, todos os *smartphones* e *notebooks* estavam com baterias totalmente carregadas e desconectados de fontes de energia. Começando pelos resultados do experimento das matrizes, a Tabela 5.4 mostra a divisão das tarefas entre os nós em cada cenário escolhido. Ela traz o número de tarefas recebidas para a execução em cada nó e a soma do tamanho delas.

Tabela 5.4: Quantidade e tamanho médio (em MFlops) das tarefas alocadas por dispositivo em cada cenário nas multiplicações de matrizes.

	Smartphone 1		Smartphone 2		Notebook 1		Notebook 2		Desktop 1		Desktop 2	
	Num.	Tam.	Num.	Tam.	Num.	Tam.	Num.	Tam.	Num.	Tam.	Num.	Tam.
Cen. 1	4,36	10,17	5,64	10,15	-	-	-	-	-	-	-	-
Cen. 2	1,23	0,58	0,93	4,66	7,84	15,08	-	-	-	-	-	-
Cen. 3	-	-	-	-	3,50	3,72	6,50	16,60	-	-	-	-
Cen. 4	-	-	-	-	2,40	1,28	3,27	8,26	4,33	10,78	-	-
Cen. 5	-	-	-	-	0,90	0,34	1,53	1,03	4,70	14,21	2,87	4,74

Podemos ver que no primeiro cenário as tarefas são distribuídas de forma equivalente entre os *smartphones*. Já no segundo, onde N1 passa a oferecer também o serviço, vemos que há uma diminuição tanto do número quanto do tamanho das tarefas nos telefones em detrimento da execução em N1, já que o seu processador é melhor. Quando o *notebook* N2 entra no terceiro cenário, os *smartphones* deixam de ser interessantes para o *grid* e passam a não executar mais nenhuma tarefa. Nos cenários seguintes os *desktops* passam a receber mais tarefas e de maior tamanho.

Percebemos em todos os cenários que, a partir dos dados enviados pelos gerenciadores de informações dos nós, o escalonador conseguiu distribuir as tarefas de maior tamanho para os nós com maior capacidade de processamento. É interessante também notar que quanto mais acrescentamos dispositivos mais velozes, menos tarefas e com menor carga são distribuídos para os equipamentos mais lentos.

Esse comportamento também é comprovado pelos resultados do experimento 2 que estão na Tabela 5.5. Podemos ver que o *notebook* N1, o que tem menor processamento, recebe a menor quantidade de vídeos e de tamanho menor que os demais, e essa quantidade vai diminuindo a medida que os *desktops*, com maior processamento, entram no *grid*.

Tabela 5.5: Quantidade e tamanho dos vídeos (em MBytes) das tarefas alocadas por dispositivo em cada cenário de conversão de vídeo.

	Notebook 1		Notebook 2		Desktop 1		Desktop 2	
	Num.	Tam.	Num.	Tam.	Num.	Tam.	Num.	Tam.
Cenário 1	22,80	38,42	46,20	75,19	-	-	-	-
Cenário 2	13,40	22,30	22,43	37,11	33,17	54,20	-	-
Cenário 3	3,73	5,81	10,16	16,72	30,57	51,71	24,54	39,37

Essa distribuição de um maior número de tarefas e de maior tamanho para dispositivos mais velozes impactam diretamente no tempo total de execução da aplicação (*makespan*) e da vazão de processamento (*throughput*) do *grid*. Os gráficos das Figuras 5.4 e 5.5, mostram o *makespan* e o *throughput* do *grid* em cada cenário de cada experimento. Podemos ver nitidamente um decréscimo no tempo e um aumento da vazão de processamento.

Dois fatos chamam a atenção no experimento das matrizes. O primeiro é que no cenário 2, o tempo total de 17,16 segundos foi bem maior que o tempo de 1,6 segundos que o *notebook* N1 gastou em sua execução sequencial. Nesse cenário, a

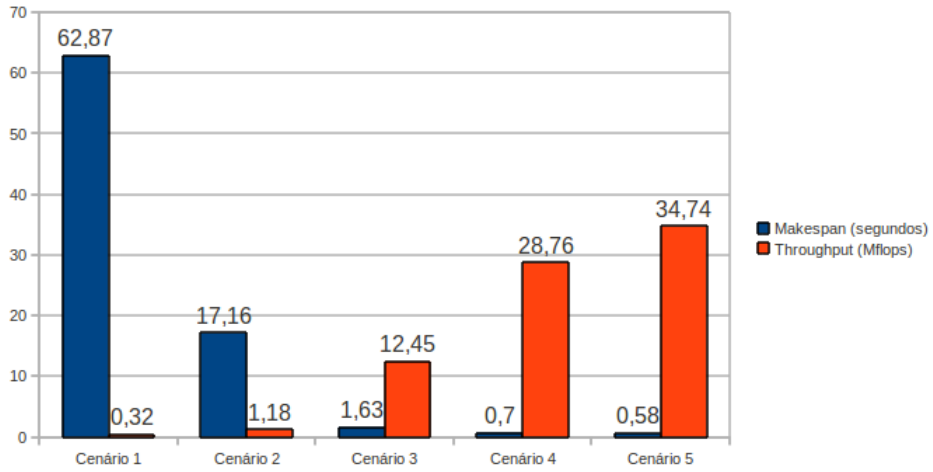

Figura 5.4: Resultados de *makespan* e *throughput* para o experimento 1.

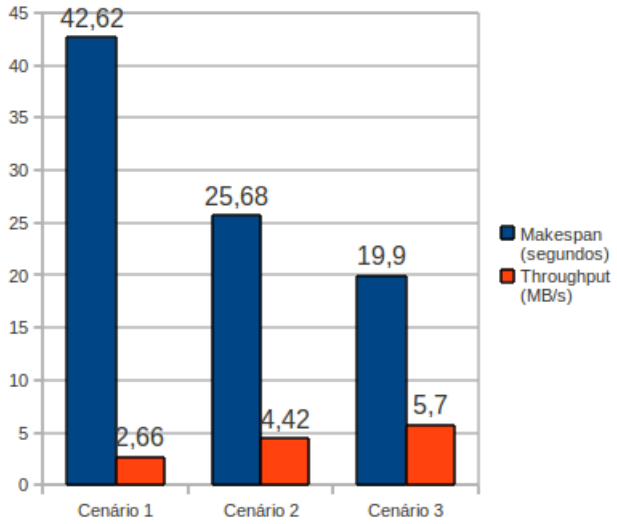

Figura 5.5: Resultados de *makespan* e *throughput* para o experimento 2.

distribuição de tarefas com os *smartphones* representou piora ao invés de melhoria. O segundo fato é que o menor tempo obtido no cenário 5 com 0,58 segundos foi maior que o tempo de execução sequencial nos *desktops*. Nesse caso, para a aplicação de tamanho pequeno gastou-se mais tempo com o overhead de comunicação e solicitação de serviços do que se gastou para executar a tarefa em si.

Já o resultado para a conversão de vídeo conseguiu obter tempos melhores que as conversões sequenciais, mesmo o *makespan* considerando também o tempo de envio do vídeo de entrada e o recebimento do vídeo convertido. Só essa transferência do vídeo original para os nós de conversão tomaria cerca de, no mínimo, 9 segundos do tempo total usando toda a capacidade da rede de 100 Mbits na qual os testes foram

realizados.

Já com relação às métricas de aceleração da execução (*speedup*) e eficiência, os gráficos das Figuras 5.6 e 5.7 mostram os resultados para os dois experimentos. No primeiro gráfico vemos que o primeiro cenário é o que traz o melhor resultado, com um aumento de 97% de velocidade com 85% de eficiência. Esse resultado demonstra que os *smartphones* passaram a maior parte do tempo processando o serviço da multiplicação e por isso usar os dois em paralelo saiu bem mais vantajoso que usar apenas um.

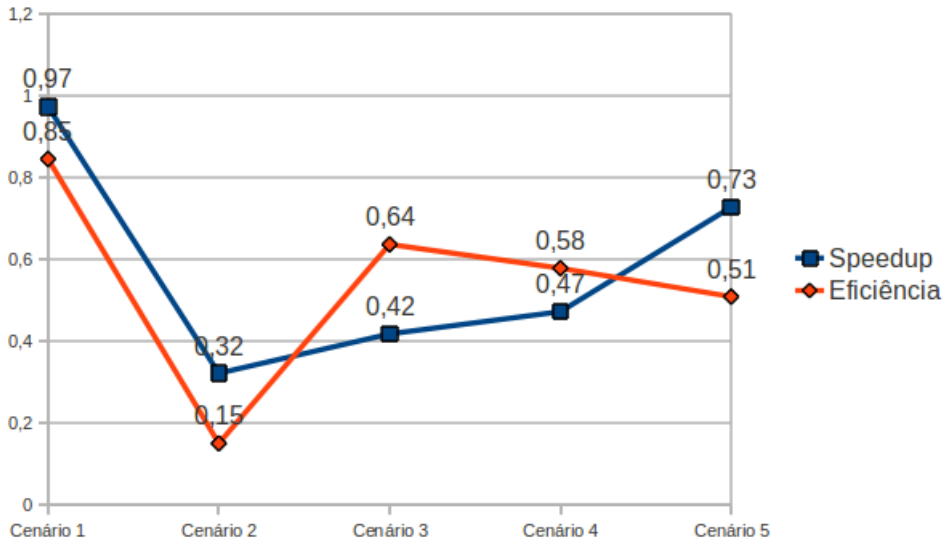

Figura 5.6: *Speedup* e eficiência para o experimento 1.

Em compensação, o segundo cenário foi o que obteve o pior resultado. Como comentado anteriormente, nesse caso o coordenador incluiu os *smartphones* para realizar o serviço de processamento quando o *notebook* N1 poderia fazer o serviço de maneira mais eficaz. Para os demais cenários, o *speedup* foi melhorando gradativamente a medida que novos equipamentos eram adicionados e a eficiência começou foi diminuindo numa faixa de 6% a cada dispositivo adicionado. Como a aplicação de multiplicação de matrizes não é uma tarefa de tamanho pequeno, o máximo de aceleração obtido foi de 73%, no cenário 5.

No gráfico com os resultados da conversão de vídeo, vemos que a eficiência teve valores semelhantes aos da multiplicação de matrizes, com uma redução de 6% a cada equipamento adicionado. Em compensação, como a conversão de vídeo é uma tarefa de grandeza média, com a mesma eficiência foram obtidos melhores resultados

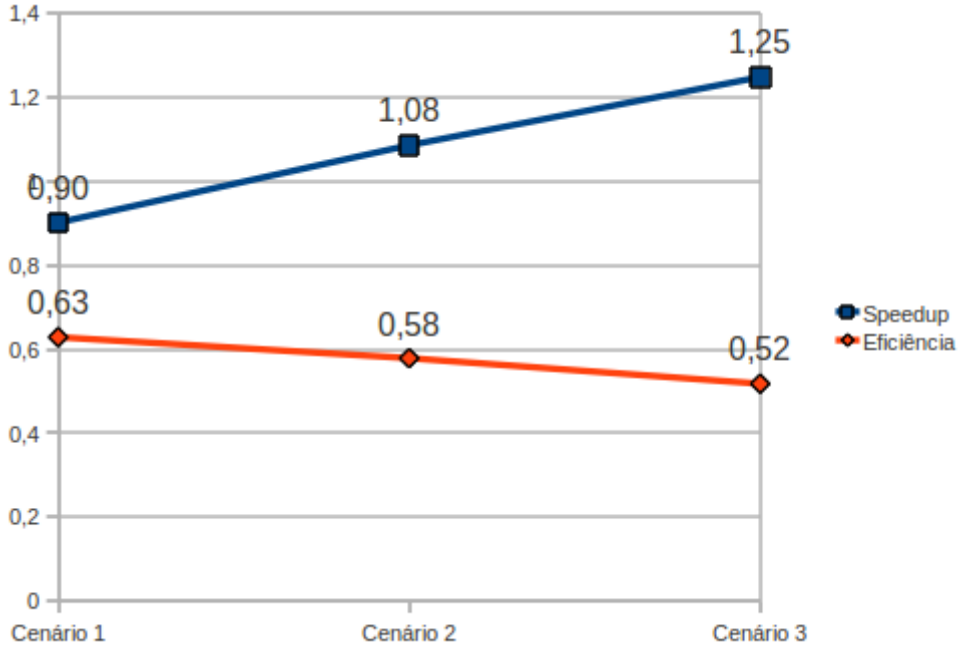

Figura 5.7: *Speedup* e eficiência para o experimento 2.

para o *speedup*, com a aceleração da aplicação aumentando cerca de 18% em cada cenário, chegando a uma melhoria de 125% no cenário 5.

Capítulo 6

Conclusões e Trabalhos Futuros

A implantação da TV Digital no Brasil está abrindo um novo mercado de desenvolvimento de sistemas e proporcionando novas oportunidades para realização de pesquisas. Esta dissertação abordou a criação do *middleware* Gringa, que é voltado para a formação de *grids* de serviços focado em aplicações de TVDI em um ambiente residencial.

Os *grids* computacionais vem se tornando uma das tecnologias mais populares e relevantes de computação paralela. Aliado ao uso do paradigma orientado a serviços, os *grids* de serviços merecem destaque pois proveem um alto nível de abstração na comunicação e implementação de funcionalidades distribuídas pelos nós, permitindo que uma mesma tarefa possa ser implementada e executada em plataformas distintas de forma simples. Antes vistos apenas como plataformas para redes de volumosos recursos computacionais, voltados para uma demanda de experimentos científicos, ultimamente os *grids* estão se movendo em direção a tornar-se uma plataforma genérica para compartilhamento de vários tipos de recursos na rede.

O uso de um *middleware* é de fundamental importância para que isso seja possível, já que é função de um *middleware* abstrair a complexidade de interação entre componentes. Conforme mostrado aqui, vários são os aspectos a serem levados em consideração na construção de um *middleware* para *grids* orientados a serviços e vários também são os *middlewares* já implementados. Cada um usa técnicas distintas de forma a melhor atender os cenários para os quais foram projetados.

Através do uso do Gringa, mostrou-se que é possível construir, de uma maneira fácil, aplicações interativas para TV Digital que podem utilizar recursos computa-

cionais de equipamentos disponíveis em uma casa, como computadores e *telefones*, para realizar tarefas para o qual um STB não estaria apto ou demoraria muito tempo para executar. Além disso, o próprio STB pode ser um nó do *grid* e passar a oferecer serviços, como comunicação de outras aplicações com alguma que tenha sido enviada pela emissora de TV.

A arquitetura e implementação do Gringa foram projetadas para deixá-lo flexível, tornando fácil a substituição de seus componentes internos. Com isso, o Gringa abre novas possibilidades de pesquisas envolvendo *grids* e TVDI, tanto no aspecto de desenvolvimento de novos componentes que estendam a funcionalidade do *middleware*, quanto no desenvolvimento e comunicação de aplicações interativas para TV que utilizam o paradigma de *grid*.

Os componentes já desenvolvidos para o Gringa conseguiram um desempenho satisfatório, distribuindo as tarefas de forma a enviar um maior número de tarefas e de maior complexidade a nós com mais processamento. Isso fez com que houvesse uma maior vazão de tarefas no *grid* em um mesmo período, reduzindo assim o tempo total necessário para que as aplicações fossem executadas completamente.

As métricas de *speedup* e eficiência também demonstraram sucesso nos experimentos realizados. Na multiplicação de matrizes houve um ganho de *performance* entre 32 e 97%, com uma taxa de eficiência entre 15 e 85%. Já na conversão de vídeo houve um ganho entre 90 e 125%, com eficiência entre 52 e 63%. Se consideramos o tempo de multiplicação de matrizes somente com o STB e o fato de que o mesmo não possuía aplicação de conversão de vídeo, comparando com os resultados obtidos com o *grid*, podemos perceber ainda mais sua relevância e seus benefícios.

6.1 Limitações e Trabalhos Futuros

Pesquisando na literatura, podemos ver que existem diversas técnicas diferentes para a implementação de componentes de *middleware* de *grids* de serviços. Para sua implementação, o Gringa utilizou um conjunto restrito de algoritmos e padrões visando a melhor adequação ao cenário de aplicação proposto. Entretanto, é importante testar novos componentes a fim de encontrar configurações que visem melhorar o desempenho do Gringa, aumentando sua eficiência, e torná-lo compatível com outras tecnologias.

Um exemplo disso é o padrão de orientação a serviços utilizado. O XML-RPC tem uma especificação simples e é bastante leve em comparação com outras tecnologias, como *Web Services* e REST. Entretanto ele não possui suporte nativo à composição e orquestração de serviços e estas características podem ser necessárias em algumas aplicações. De maneira semelhante, temos diversos algoritmos de escalonamento, modelos de gerenciamento de serviços de diretórios, mecanismos de transferências de dados, gerenciamento de falhas (principalmente as que ocorrem com o coordenador), etc. Até mesmo para medir o desempenho do *grid* não existe um padrão ou consenso sobre quais as melhores métricas.

Uma forma de estimular o desenvolvimento de novos componentes e aplicações, além de realizar uma busca mais efetiva por falhas no código do Gringa é torná-lo um *software* livre, que é o intuito deste trabalho. Isto facilitaria também a construção de componentes que tornem possível a comunicação do Gringa com outros middlewares orientados a serviços. A atuação do Gringa como coordenador de um *grid* local residencial interligado a outros *grids* de escopo global poderá estimular a participação em iniciativas junto à *community grids* globais, ou ainda contribuir para um novo modelo de negócios de computação em nuvem mediado pela TV, por exemplo.

Outras tecnologias que vem ganhando espaço nos últimos tempos e que também podem ser desenvolvidos com o Gringa são os *grids* ubíquos e pervasivos. Eles podem ser agregados como serviços disponíveis para as aplicações interativas da TV, possibilitando uma maior imersão do usuário e uma comunicação com todos os equipamentos de uma casa através da TV usando um único padrão. Além disso, destacamos também a possibilidade de utilização de códigos móveis, de forma que os serviços pudessem ser distribuídos de forma dinâmica para os nós do *grid*, o que não ocorre atualmente.

Por fim, é importante dizer que não existe *middleware* finalizado, que não necessite de mudanças. A evolução das tecnologias e o surgimento de outras, aliada às novas demandas por parte dos usuários fazem com que as alterações sejam constantes e necessárias. É fundamental que um *middleware* esteja preparado para isto, e foi nesse sentido que o Gringa foi desenvolvido.

Referências Bibliográficas

ABNT. *NBR 15607-1 – Televisão digital terrestre – Canal de interatividade – Parte 1: Protocolos, interfaces físicas e interfaces de software*. 1. ed. Rio de Janeiro-RJ, 2008. Disponível em: <http://www.dtv.org.br/download/pt-br/ABNTNBR15607_2D1_2008Ed1.pdf>.

ABNT. *NBR 15606-2 – Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações*. 1. ed. Rio de Janeiro-RJ, 2009. Disponível em: <http://www.dtv.org.br/download/pt-br/ABNTNBR15606-2_2007Vc3_2008.pdf>.

ABNT. *NBR 15606-5 – Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 5: Ginga-NCL para receptores portáteis – Linguagem de aplicação XML para codificação de aplicações*. 1. ed. Rio de Janeiro-RJ, 2009. Disponível em: <http://www.dtv.org.br/download/pt-br/ABNTNBR15606_2D5_2008Vc2_2009Port.pdf>.

ABNT. *NBR 15606-4 – Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais*. 1. ed. Rio de Janeiro-RJ, 2010. Disponível em: <http://www.dtv.org.br/download/pt-br/ABNTNBR15606-4_2010Ed1.pdf>.

ABNT. *NBR 15606-6 – Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 6: Java DTV 1.3*. 1. ed. Rio de Janeiro-RJ, 2010. Disponível em: <http://www.dtv.org.br/download/pt-br/ABNTNBR15606-6_2010Ed1.pdf>.

ANDREOZZI, S.; GHISELLI, A.; HU, C.; JIANG, J.; KONYA, B.; RIEDEL, M.; VIRDEE, D. *Report on Grid Activities relevant to the identification of new services*. Bolonha-Itália, 2006.

AUGUSTIN, I.; FERREIRA, G. P.; YAMIN, A. Grade computacional como infra-estrutura para a computação ubíqua/pervasiva. In: *Escola Regional de Alto Desempenho*. Santa Cruz do Sul-RS: UCPel, 2008.

BATISTA, C. E. C. F.; ARAúJO, T. M. U. de; OMAIA, D.; ANJOS, T. C. dos; CASTRO, G. M. L. de; BRASILEIRO, F. V.; FILHO, G. L. de S. Tvgrid: A grid architecture to use the idle resources on a digital tv network. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid*. Rio de Janeiro-RJ: IEEE Press, 2007.

CELIK, T.; BOS, B.; HICKSON, I.; LIE, H. W. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. Cambridge-EUA, 2009. Http://www.w3.org/TR/2009/CR-CSS2-20090908.

CHINNICI, R.; WEERAWARANA, S.; MOREAU, J.-J.; RYMAN, A. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Cambridge-EUA, 2007. Http://www.w3.org/TR/2007/REC-wsdl20-20070626.

COMPUTERWORLD. *Para IDC, ultraportáteis dominaram mercado em 2010*. 2011. Disponível em: <http://computerworld.uol.com.br/negocios/2011/01-/14/para-idc-ultraportateis-dominaram-mercado-em-2010/>. Acesso em: 10 de março de 2011.

CORONATO, A.; PIETRO, G. D. Mipeg: A middleware infrastructure for pervasive grids. In: *Future Generation Computer System*. Napoli-Itália: Elsevier, 2008. v. 24.

COSTA, R.; BRASILEIRO, F.; FILHO, G. L.; SOUZA, D. M. Oddci: on-demand distributed computing infrastructure. In: *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. Nova Iorque-EUA: IEEE Press, 2009.

DANTAS, C. F. F. *Escalonamento de Tarefas em Grid no Cenário de TV Digital Interativa*. Mossoró-RN: Monografia (Graduação em Ciência da Computação). Universidade do Estado do Rio Grande do Norte, 2011.

DASHOFY, E.; ASUNCION, H.; HENDRICKSON, S.; SURYANARAYANA, G.; GEORGAS, J.; TAYLOR, R. Archstudio 4: An architecture-based meta-modeling environment. In: *Companion to the proceedings of the 29th International Conference on Software Engineering*. Washington, DC, EUA: IEEE Computer Society, 2007. (ICSE COMPANION '07), p. 67–68. ISBN 0-7695-2892-9. Disponível em: <http://dx.doi.org/10.1109/ICSECOMPANION% -.2007.21>.

DASHOFY, E. M. Issues in generating data bindings for an xml schema-based language. In: *In Proceedings of the Workshop on XML Technologies and Software Engineering (XSE2001)*. Nova Iorque-EUA: ACM, 2001.

DONGARRA, J.; LUSZCZEK, P.; PETITET, A. The linpack benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, John Wiley & Sons, v. 15, n. 9, 2003.

ECMA, I. *EcmaScript Language Specification*. Genebra-Suiça, 2011. Http://www.w3.org/TR/2009/CR-CSS2-20090908.

ERL, T. *SOA - Princípios de design de serviços*. Boston-EUA: Prentice Hall, 2009.

ERL, T. *SOA Design Patterns*. Boston-EUA: Prentice Hall, 2009.

ETSI. *Digital Video Broadcasting (DVB), Implementation guidelines for Data Broadcasting*. Nice-França, 2003.

FALAVINHA JUNIOR, J. N.; JúNIOR, A. M.; OLIVEIRA, L. J. de; BOCCARDO, D. R. Avaliação de algoritmos de escalonamento em grids para diferentes configurações de ambiente. In: *WPerformance - V Workshop em Desempenho de Sistemas Computacionais e de Comunicação*. Rio de Janeiro-RJ: Sociedade Brasileira de Computação, 2007.

FERNANDES, J.; FILHO, G. L. de S.; SILVEIRA, G. E. Introdução à televisão digital interativa: Arquitetura, protocolos, padrões e práticas. In: *Anais da Jornada de Atualização em Informática do Congresso da SBC*. Salvador-BA: Sociedade Brasileira de Computação, 2004.

FOSTER, I. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci & Technol.*, v. 21, n. 4, p. 513–520, 2006.

FOSTER, I.; GANON, D.; KISHIMOTO, H.; REICH, J. J. V. *The Open Grid Services Architecture Use Cases*. Chicago-EUA, 2004.

FOSTER, I.; KESSELMAN, C. *The Grid: Blueprint for a new Computing Infrastructure*. San Francisco-EUA: Morgan Kaufmann Publishers, 1999.

FOSTER, I.; KISHIMOTO, H.; SAVVA, A. *The Open Grid Services Architecture, Version 1.0*. Chicago-EUA, 2005.

G1. *Brasil fecha 2010 com 202,9 mi de celulares; TIM ganha mercado*. 2011. Disponível em: <http://g1.globo.com/tecnologia/noticia/2011/01/brasil-fecha-2010-com-2029-mi-de-celulares-tim-ganha-mercado-1.html>. Acesso em: 10 de março de 2011.

GATIS, I. de A. L. *Um Middleware para Construção de Aplicações de TV Digital Distribuídas baseadas no Modelo P2P*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2006.

GEER, D. Chip makers turn to multicore processors. *Computer*, v. 38, n. 5, p. 11 – 13, 2005. ISSN 0018-9162.

GHISI, B. C.; LOPES, G. F.; SIQUEIRA, F. Integração de aplicações para tv digital interativa com redes sociais. In: *Anais do Simpósio Brasileiro de Sistemas Multimídia e Web – WebMedia*. Belo Horizonte-MG: UFMG, 2010.

GOLDFARB, C. F.; PRESCOD, P. *The XML Handbook*. Upper Saddle River, NJ 07458, USA: Prentice-Hall PTR, 1998.

IBGE. *Pesquisa nacional por amostra de domicílios*. 2009. Disponível em: <http:// www.ibge.gov.br/home/estatistica/populacao/trabalhoerendimento /pnad2009/>. Acesso em: 08 de Janeiro de 2011.

IERUSALIMSCHY, R.; FIGUEIREDO, L. H. de; HENRIQUE, L.; WALDEMAR, F.; FILHO, W. C. Lua - an extensible extension language. *Software: Practice & Experience*, v. 26, p. 635–652, 1995.

JAGANNADHAM, D.; RAMACHANDRAN, V.; KUMAR, H. Java2 distributed application development (socket, rmi, servlet, corba) approaches, xml-rpc and web services functional analysis and performance comparison. In: *Communications and Information Technologies, 2007. ISCIT '07. International Symposium on*. Sidney-Austrália: IEEE Press, 2007. p. 1337 –1342.

JUCÁ, P. M.; LUCENA, U. Experiências no desenvolvimento de aplicações para televisão digital interativa. In: *III Fórum de Oportunidades em Televisão Digital Interativa*. Poços de Caldas-MG: Pontifícia Universidade Católica de Minas Gerais, 2005.

KHARE, R.; GUNTERSDORFER, M.; OREIZY, P.; MEDVIDOVIC, N.; TAYLOR, R. xadl: enabling architecture-centric tool integration with xml. In: *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. Wailea-EUA: University of Hawaii at Manoa, 2001.

KURNIAWAN, D.; ABRAMSON, D. An integrated grid development environment in eclipse. In: *e-Science and Grid Computing, IEEE International Conference on*. Los Alamitos: IEEE Press, 2007. p. 491 –498.

LEITE, L. E. C.; BATISTA, C. E. C. F.; FILHO, G. L. de S.; KULESZA, R.; ALVES, L. G. P.; BRESSAN, G.; RODRIGUES, R. F.; SOARES, L. F. G. Flextv – uma proposta de arquitetura de middleware para o sistema brasileiro de tv digital. *Revista de Engenharia de Computação e Sistemas Digitais*, n. 2, 2005.

LI, G.; SUN, H.; GAO, H.; YU, H.; CAI, Y. A survey on wireless grids and clouds. In: *Grid and Cooperative Computing, 2009. GCC '09. Eighth International Conference on*. Lanzhou-China: IEEE Press, 2009. p. 261 –267.

LUSZCZEK, P.; DONGARRA, J.; KOESTER, D.; RABENSEIFNER, R.; LUCAS, B.; KEPNER, J.; MCCALPIN, J.; BAILEY, D.; TAKAHASHI, D. Introduction to the hpc challenge benchmark suite. *SC2005*, Citeseer, Seatle-EUA, 2005.

MA, S.; XIAOSHEDONG; MEI, Y.; WANG, Z.; ZHU, Z. Taxonomy and an ontology for grid metrics. In: *Proceedings of the The Third ChinaGrid Annual Conference (chinagrid 2008)*. Washington-EUA: IEEE Press, 2008. (CHINAGRID '08), p. 300–307. ISBN 978-0-7695-3306-3. Disponível em: <http://dx.doi.org/10.1109/ChinaGrid.2008.48>.

MADWAY, G. *Vendas de computadores crescem 13,6% em 2010*. 2011. Disponível em: <http://br.reuters.com/article/internetNews-/idBRSPE70C02Z20110113?sp=true>. Acesso em: 10 de março de 2011.

MANHÃES, M. A. R.; SHIEH, P. J.; LAMAS, A. da C.; MACEDO, P. E. de O. Canal de interatividade em tv digital. *Caderno CPqD Tecnologia*, v. 1, n. 1, p. 8, 2005.

MANVI, S. S.; BIRJE, M. N. A review on wireless grid computing. *International Journal of Computer and Electrical Engineering*, 2010.

MORAIS, A. D. L. *DESENVOLVIMENTO DE UMA APLICAÇÃO PARALELA PARA CONVERSÃO DE VÍDEO H.264*. Mossoró-RN: Monografia (Graduação em Ciência da Computação). Universidade do Estado do Rio Grande do Norte, 2011.

NASCIMENTO, J. P. L. do. *Desenvolvendo aplicações utilizando a API Ginga-J: um estudo de caso com clipes interativos*. Mossoró-RN: Monografia (Graduação em Ciência da Computação). Universidade do Estado do Rio Grande do Norte, 2008.

NUNES, D. *Celular trocado a cada 14 meses*. 2011. Disponível em: <http://www.correiodopovo.com.br/Impresso-/?Ano=116&Numero=264&Caderno=0&Noticia=307532>. Acesso em: 16 de setembro de 2011.

OLIVEIRA, M.; CUNHA, P. R. F.; SANTOS, M. E. S.; BEZERRA, J. C. C. Implementing home care application in brazilian digital tv. In: *Proceedings of the Second international conference on Global Information Infrastructure Symposium*. Piscataway-EUA: IEEE Press, 2009.

ORACLE. *Overview (Connected Device Configuration)*. 2011. Disponível em: <http://download.oracle.com/javame/config/cdc/ref-impl/cdc1.1.2/jsr218-/index.html>. Acesso em: 30 de abril de 2011.

ORACLE. *Overview (Foundation Profile)*. 2011. Disponível em: <http://download-.oracle.com/javame/config/cdc/ref-impl/fp1.1.2/jsr219/index.html>. Acesso em: 30 de abril de 2011.

ORACLE. *Overview (Personal Basis Profile)*. 2011. Disponível em: <http://download.oracle.com/javame/config/cdc/ref-impl/pbp1.1.2/jsr217-/index.html>. Acesso em: 30 de abril de 2011.

PAGANI, M. *Multimedia and Interactive Digital TV: Managing the Opportunities Created by Digital Convergence*. Hershey-EUA: IRM Press, 2003.

PENG, L.; SEE, S.; SONG, J.; STOELWINDER, A.; NEO, H. K. Benchmark performance on cluster grid with ngb. *Parallel and Distributed Processing Symposium, International*, IEEE Press, Los Alamitos-EUA, v. 18, p. 275a, 2004.

QUOCIRCA. *Business Grid Computing*. 2003. Quocirca report. Disponível em: <http://www.quocirca.com/pages/analysis/reports/view/store250-/item1515>. Acesso em: 30 de abril de 2011.

RICHARDSON, I. E. G. *The h.264 advanced video compression standard*. 2nd. ed. Nova Iorque-EUA: John Wiley & Sons, 2010.

RIVERA, G. G. de; RIBALDA, R.; COLáS, J.; GARRIDO, J. A generic software platform for controlling collaborative robotic system using xml-rpc. In: *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Monterey-EUA: IEEE Press, 2005.

SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society*, v. 12, n. 4, 2007.

SOUZA FILHO, G. L.; LEITE, L. E. C.; BATISTA, C. E. C. F. Ginga-j: The procedural middleware for the brazilian digital tv system. *Journal of the Brazilian Computer Society*, v. 12, n. 4, 2007.

STANOEVSKA-SLABEVA, K.; WOZNIAK, T.; RISTOL, S. (Ed.). *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Nova Iorque-EUA: Springer, 2010.

SUBRAMANIAN, R.; GOODMAN, B. D. *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*. Hershey-EUA: Idea Group Publishing, 1999.

TAY, B. H.; ANANDA, A. L. A survey of remote procedure calls. *SIGOPS Oper. Syst. Rev.*, ACM, Nova Iorque-EUA, v. 24, p. 68–79, 1990. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/382244.382832>.

TOTVS. *Sticker BB*. 2011. Disponível em: <https://www.stickercenter.com.br-/StickerWeb/pt_BR/sticker.html?stickerName=BB&stickerId=10>. Acesso em: 20 de maio de 2011.

TRUONG, H.-L.; SAMBORSKI, R.; FAHRINGER, T. Towards a framework for monitoring and analyzing qos metrics of grid services. In: *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*. Washington-EUA: IEEE Press, 2006. (E-SCIENCE '06), p. 65–. ISBN 0-7695-2734-5. Disponível em: <http://dx.doi.org/10.1109/E-SCIENCE.2006.142>.

VIANA, N.; MAIA, O.; LUCENA, V. de; PINTO, L. A convergence proposal between the brazilian middleware for idtv and home network platforms. In: *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*. Las Vegas-EUA: IEEE, 2009. p. 1–5.

VöLTER, M.; KIRCHER, M.; ZDUN, U. *Remoting Patterns : Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. Chichester-Inglaterra: John Wiley & Sons, 2005.

WESLEY, A. (Ed.). *Documenting Software Architecture: Views and Beyond*. Boston,EUA: Paul Clements and Felix Bachmann and Len Bass and David Garlan and James Ivers and Reed Little and Robert Nord and Judith Stafford, 2003.

XIAO, Y.; DU, X.; ZHANG, J.; HU, F.; GUIZANI, S. Internet protocol television (iptv): The killer application for the next-generation internet. *Communications Magazine, IEEE*, v. 45, n. 11, p. 126 –134, 2007. ISSN 0163-6804.

ZHANG, X.; YAN, Y. Modeling and characterizing parallel computing performance on heterogeneous networks of workstations. In: *Proceedings of Seventh IEEE Symposium on Parallel and Distributed Processing*. San Antonio-EUA: IEEE Press, 2005.

Apêndices

Apêndice A

Aplicação Gringa de Multiplicação de Matrizes em Lua

```lua
-- Importando módulo com as funções do middleware Gringa
require 'gringa-client'

server = '10.6.0.151'
local result = ''

local ok1 = connect(server)
if ok1 then
    -- Arquivo que contem descrição dos arquivos das matrizes
    arq = io.open('./files/list.txt','r')
    mmlines = arq:read("*num") -- linhas
    mmcols = arq:read("*num")  -- colunas
    arq:read("*line")
    mmfilename = arq:read("*line") -- nome do arquivo da matriz completa
    mmfile = io.open('./files/' .. mmfilename, 'r')
    m2 = mmfile:read("*all")       -- lê completamente
    mmfile:close()

    local tasks ={}

    numfiles = arq:read("*num")  -- lê em quantas submatrizes foi dividido
    for i=1, numfiles do
        numl = arq:read("*num")  -- número de linhas
        arq:read("*line")
        f = arq:read("*line")    -- nome do arquivo
        mfile = io.open('./files/' .. f,'r')
        local m1=mfile:read("*all")   -- lê o arquivo completamente

        -- Chama o serviço de multiplicação de matrizes
        local ok2, tmp2 = callService('MMatriz', numl, mmlines, mmcols, m1, m2)
        if ok2 then
           tasks[i]=tmp2  -- guarda o identificador da tarefa colocando em uma tabela
           result = 'Task added'
        else
           result = 'Error on remote procedure call'
        end
    end
    print (result)
    arq:close()
    final = io.open('result.txt','w') -- cria o arquivo de result
     for i=1, numfiles do
        result = tasks[i]
        local ok3, tmp = getTaskResultBlocking(tasks[i]) -- espera pelo result da tarefa
        if ok3 then
            final:write(tostring(tmp[1]))
            result = 'Result #' .. i .. ' retrieved'
        else
            result = 'Error on retrieving result'
        end
     end
     print (result)

    result = 'Done'
    final:close()

else
   result = 'Falha na conexão ao grid'
end

print (result)
```

Apêndice B

Aplicação Gringa de Conversão de Vídeo em Lua

```
require 'gringa-client'

local result = -1
local chunksize = 1024
local buffsize= 128*chunksize -- Definindo o tamanho do buffer de leitura
local tasks = {}
local server = '10.6.0.151'

-- Função auxiliar compor os nomes dos arquivos a serem lidos
function three(num)
   if num < 10 then
       return '00' .. tostring(num)
   else
       if num <100 then
          return '0' .. tostring(num)
       else
          return num
       end
   end
end

-- Envia um arquivo para a conversão
function sendFile(i)
      local myfile = io.open('./h264/b64-arq' .. three(i) .. '.h264' ,'rb')
      if myfile then
      local f1 = myfile:read(buffsize)
      local s1 = { }
      while f1 do      -- Lendo arquivo no buffer
        s1[#s1+1] = f1
        f1 = myfile:read(buffsize)
      end
      -- Chamando o serviço de conversão
      local ok2, tmp = callService('GringaConverter', toBase64(table.concat(s1)))
        myfile:close()
        if ok2 then
                tasks[i] = tmp
                result = 'File sended'
        else
           result = 'Failed to call service'
           return false, result
        end
      else
                result = 'File not found'
                return false, result
      end

      return true, tasks
end
```

```
49
50 local ok1 = connect(server)
51 local inicio = os.time()
52 if ok1 then
53     local arq = io.open('./list.txt','r')
54     local numfiles = arq:read("*num") -- Lendo o número de arquivos a converter
55     arq:close()
56     -- Chamando a função de envio
57     for i=1, numfiles do
58          enviaArquivo(i)
59     end
60     -- Reunindo em um único arquivo
61     for i=1, numfiles do
62         local final = io.open('./mpeg/arq' .. i ..'.mpeg','wb')
63         local ok3, tmp = getTaskResultBlocking(tasks[i]) -- Obtendo arquivo convertido
64         if ok3 then
65             final:write(fromBase64(tmp[1]))
66             result = 'File received'
67         else
68             result = 'Error while receiving file'
69         end
70         print (result)
71         final:close()
72      end
73     result = 'done'
74 else
75    result = 'Falha na conexão ao grid'
76 end
77 print (result)
```

Apêndice C

Catálogo dos principais serviços XML-RPC para o Gringa

Tabela C.1: Serviços XML-RPC para o Gringa

Serviço	Descrição
TaskExecutor.receiveResult	Solicita de um nó o recebimento do resultado de uma execução realizada nele
TaskExecutor.getStatus	Obtém o status da execução de uma tarefa em um nó
TaskExecutor.putResultTask	Envia ao coordenador o resultado da tarefa para que seja armazenada
TaskExecutor.execute	Solicita a um nó a execução de uma tarefa
TaskExecutor.containsRequest	Verifica se uma requisição está sendo atendida
TaskExecutor.cancel	Solicita o cancelamento de uma execução de uma tarefa no nó
NodoInfor.getStateResource	Solicita informações sobre algum tipo de recurso (número de tarefas executando, processamento, nível de bateria, etc) de um nó
NodoInfor.pong	Usado para retornar o resultado do Ping-Pong, *benchmark* para medir a velocidade da rede
AppManager.getStatusTask	Usado pelo cliente para obter uma tarefa enviada ao Gerenciador de aplicações
AppManager.getTasksNoDone	Devolve a lista de tarefas ainda não concluídas para uma aplicação
AppManager.constainsApp	Verifica se uma aplicação está registrada
AppManager.getTasks	Obtém a lista de todas as tarefas de uma aplicação
AppManager.getDoneTasks	Devolve a lista de tarefas já concluídas na aplicação
AppManager.newApp	Registra uma nova aplicação
AppManager.finalize	Finaliza uma aplicação
AppManager.receiveResult	Usado pelo cliente para obter o valor do resultado de uma tarefa ou valor nulo caso ela não tenha sido concluída
AppManager.cancel	Cancela a execução de uma tarefa
AppManager.addTask	Solicita a inclusão da execução de uma nova tarefa
AppManager.remTask	Remove uma instância da lista de tarefas em uma aplicação
AppManager.waitForResult	Representa uma espera bloqueada pelo resultado de uma tarefa
AppManager.getNumTasks	Obtém o número de tarefas registradas para a aplicação
NodeManager.updateAttributs	Atualiza as informações de um nó no Gerenciador de nós
NodeManager.register	Registra a entrada de um nó no *grid*
NodeManager.remService	Usado pelos nós para remover um serviço da lista dos disponíveis
NodeManager.ping	Envia a solicitação de um benchmark de rede
NodeManager.addService	Usado pelos nós para adicionar um serviço da lista dos disponíveis
NodeManager.remove	Registra a saída de um nó no *grid*