UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UMA PROPOSTA DE API PARA DESENVOLVIMENTO DE APLICAÇÕES MULTIUSUÁRIO E MULTIDISPOSITIVO PARA TV DIGITAL UTILIZANDO O MIDDLEWARE GINGA

LINCOLN DAVID NERY E SILVA

JOÃO PESSOA - PB
AGOSTO - 2008

LINCOLN DAVID NERY E SILVA

UMA PROPOSTA DE API PARA DESENVOLVIMENTO DE APLICAÇÕES MULTIUSUÁRIO E MULTIDISPOSITIVO PARA TV DIGITAL UTILIZANDO O MIDDLEWARE GINGA

Dissertação apresentada ao programa de Mestrado em Informática da Universidade Federal da Paraíba, como requisito para a obtenção do título de Mestre em Informática (Sistemas de Computação).

Orientação: Guido Lemos de Souza Filho

JOÃO PESSOA - PB

AGOSTO - 2008

LINCOLN DAVID NERY E SILVA

Uma Proposta de API para Desenvolvimento de Aplicações Multiusuário e Multidispositivo para TV Digital Utilizando o Middleware Ginga

Dissertação apresentada ao programa de Mestrado em Informática da Universidade Federal da Paraíba, como requisito para a obtenção do título de Mestre em Informática.

Data de aprovação:

_____ / _____ / 2008.

Banca examinadora:

_____
Prof. Dr.
Instituição:

_____
Prof. Dr.
Instituição:

_____
Prof. Dr.
Instituição:

_____
Prof. Dr.
Instituição:

Dedicatória

Dedico esse trabalho a minha família, em especial a meus pais, Francisco Essenine e Silva e Leoneide Nery e Silva, que sempre me incentivaram e trabalharam para que, por meio dos estudos e esforços necessários, eu conseguisse alcançar objetivos cada vez maiores.

Agradecimentos

Agradeço primeiramente a Deus, por verdadeiramente reconhecer que sem Sua ajuda e cuidado as vitórias não aconteceriam;

a toda minha família, por sempre me incentivar e se alegrarem comigo a cada vitória;

aos meus mestres, em especial a meu orientador, Guido Lemos de Souza Filho, pelas oportunidades e por acreditar nos frutos de meu trabalho;

aos meus amigos, por serem honestamente amigos;

a Luiz Eduardo Cunha Leite, que me orientou durante o projeto FlexTV, quando se iniciou o desenvolvimento desse trabalho;

e um agradecimento especial a minha co-orientadora nesse trabalho, Tatiana Alves Tavares, que prestou a ajuda essencial para conclusão do mesmo.

Epígrafe

"A esperança não vem do mar, vem das antenas de TV"

Herbert Viana/ Bi Ribeiro

Resumo

O avanço das aplicações de TV Digital Interativa não ocorre na mesma velocidade que as aplicações para *WEB* ou *Desktop*. Tal fato se deve tanto por limitações encontradas no *hardware* e no *middleware* no qual as aplicações são executadas, quanto pela limitação do dispositivo usado na interação dos usuários com a TV. No panorama nacional, a especificação do *middleware* Ginga permite a incorporação de novas funcionalidades através da API de Integração de Dispositivos, alvo desse trabalho. Esta API que permite que aplicações de TVDI usem dispositivos móveis tanto como meio de interação, como para compartilhamento de seus recursos multimídia. Como resultado do uso da API proposta, as aplicações de TVDI passam a contar com novas possibilidades até então não disponíveis nos *middlewares* de TV Digital existentes; como a utilização de mais de um dispositivo simultaneamente, o suporte ao desenvolvimento de aplicações multiusuário e o acesso a recursos de captura de mídias contínuas disponíveis em aparelhos como celulares, que podem ser integrados aos aparelhos de TV. A API resultante desse trabalho foi implementada e utilizada no desenvolvimento de aplicações para TVDI voltadas a explorar os novos recursos avançados disponíveis.

Abstract

The Interactive Digital TV applications progress does not occur at the same speed we found at Web or Desktop applications. This fact is due to constraints encountered in both hardware and the middleware in which applications run, and also due to the limited way we have to interact with the TV: with the traditional remote control. In the Brazilian scene, the middleware Ginga specification allows the incorporation of new functionalities through the Device Integration API, which is target of this dissertation. The API allows TVDI applications to use mobile devices both as a means of interaction, and to share its multimedia resources. As a result of the API use, TVDI applications are able to employ new possibilities not available in others existing Digital TV middlewares, like the use of multimedia resources and multiuser support. The new API has been implemented and applied to develop TVDI applications aiming to explore the new advanced features available.

Sumário

Lista de ilustrações

Lista de tabelas

ABREVIATURAS

Sigla	Significado
ABNT	**Associação Brasileira de Normas Técnicas**
ACAP	**Advanced Common Application Platform**
API	**Application Programming Interface (Interface de Programação da Aplicação)**
ARIB	**Association of Radio Industries and Businesses**
ATSC	**Advanced Television Systems Committee**
BML	**Broadcast Markup Language**
CA	**Conditional Access (Acesso Condicional)**
CENELEC	**European Committee for Electrotechnical Standardization**
DASE	**Digital TV Applications Software Enviroment**
DAVIC	**Digital Audio-Visual Council**
DIBEG	**The Digital Broadcasting Experts Group**
DVB	**Digital Video Broadcasting**
EBU	**European Broadcasting Union**
EPG	**Eletronic Program Guide (Guia de Programação Eletrônico)**
ETSI	**European Telecommunications Standards Institute**
GEM	**Globally Executable MHP**
GPRS	**General Packet Radio Service**
GPS	**Global Positioning System**
HAN	**Home Area Network**
HAVI	**Home Audio Video Interoperability**
IP	**Internet Protocol**
ISDB	**Integrated Services Digital Broadcasting**
JMF	**Java Media Framework**
JVM	**Java Virtual Machine**
Lavid	**Laboratório de Aplicações de Vídeo Digital**
LCD	**Liquid Crystal Display**
MHP	**Multimedia Home Platform**
OCAP	**OpenCable Application Platform**
PDA	**Personal Digital Assistants**
SBTVD	**Sistema Brasileiro de TV Digital**
SI	**Service Information**
SO	**Sistema Operacional**
STB	**Set-top Box**
TCP	**Transmission Control Protocol**

Sigla	Significado
TS	**Transport Stream**
TV	**Televisão**
TVD	**TV Digital**
TVDI	**TV Digital Interativa**
UML	**Unified Modeling Language**
VOD	**Video over Demand (Vídeo sob Demanda)**

1 INTRODUÇÃO

Um sistema básico de Televisão Digital (TVD) consiste de uma estação transmissora, um meio físico sobre o qual o sinal é transmitido, que pode ser o ar ou meios físicos guiados (cabo coaxial, fibra óptica etc.), e um receptor (o STB) responsável por receber o sinal transmitido, decodificá-lo e exibi-lo. A Figura 1 ilustra um cenário simplificado de televisão digital, incluindo os principais componentes.

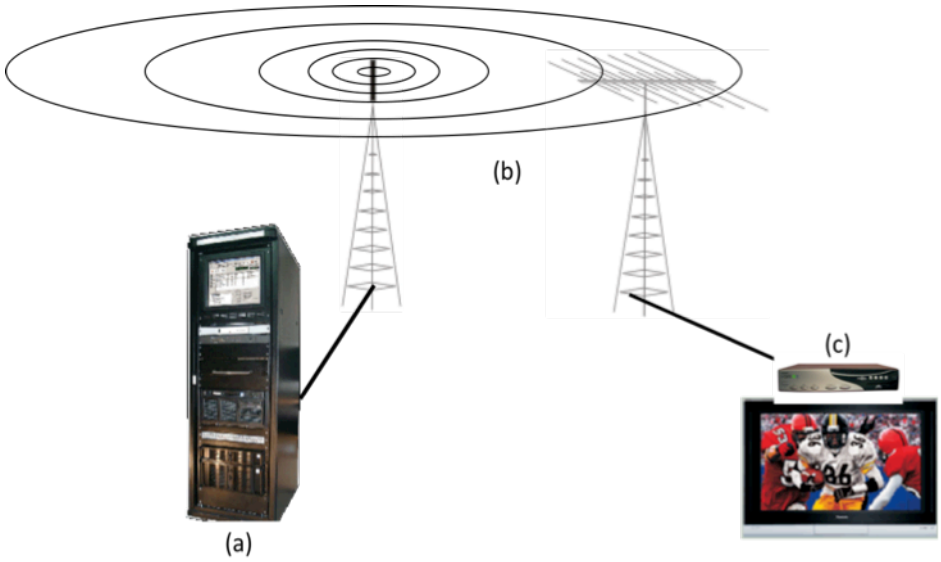

Figura 1: Sistema de TV Digital e Interativa. Perceba os vários componentes integrantes do sistema: a estação transmissora (a), a transmissão do sinal por difusão (b) e o receptor (c) que converte e exibe o conteúdo.

Como a transmissão é feita através de um fluxo de *bits*, há a possibilidade de se transmitir uma maior quantidade de informação, em comparação ao sistema analógico. Isso é possível principalmente graças ao desenvolvimento de técnicas de compressão, através das quais se pode produzir vídeos com taxas em *bits* de 1/4 a 1/10 do original puro (vídeo digital não comprimido) (1).

A aplicação de técnicas de multiplexação no sinal de TV Digital permite, que adicionalmente aos fluxos de áudio e vídeo tradicionais do ambiente de televisão, sejam incluídos um ou mais fluxos de dados[1]. Dentre as possibilidades para fluxos de dados podemos citar: teletexto, informações sobre a programação do canal, além código executável

[1] No formato analógico há transmissão de dados através da utilização do intervalo VBI (*Vertical blank interval*) para transmissão de *closed-caption* entre outras informações

(aplicações). É esta possibilidade de transmitir dados binários que codificam aplicações, que torna possível a concepção e funcionamento da TV Digital Interativa.

O conceito de TV Interativa não é novo; ele começou a ser explorado com o teletexto, nos 80. O grupo americano Warner-Qube ainda não conhecia o termo, mas no começo dos anos 70 já realizava experimentos interativos em um sistema de vídeo sob demanda (VOD). Infelizmente esses experimentos foram sufocados pela dificuldade de se conseguir uma rede de duas vias com qualidade até as residências dos clientes (2). Hoje em dia, com o avanço da tecnologia, o termo ganhou maior abrangência, e a TV Interativa se refere a uma gama bem maior de serviços, tais como: informativos meteorológicos, participação de enquetes, compras pela TV e até mesmo navegação na Internet.

A possibilidade de executar a mesma aplicação em dispositivos diferentes, de diferentes fabricantes e com diferentes capacidades de processamento, é viabilizada pela adoção de um *middleware* comum, que pode ser definido como uma camada de *software* que abstrai as características específicas de *hardware* da plataforma. Além de definir as possibilidades para cada sistema interativo de TV, o *middleware* é responsável pela padronização dessas capacidades. Ao ser executado, o *middleware* disponibiliza para os desenvolvedores uma máquina virtual que pode ser programada através de interfaces de programação padronizadas (APIs, do inglês *Application Programming Interfaces*). Essa padronização só é conseguida com a adoção de padrões abertos de *middlewares*.

Os principais projetos de middleware aberto são o *Multimedia Home Plataform* (MHP), do europeu *Digital Video Broadcast* (DVB), além dos americanos Digital TV Applications Software Enviroment (DASE), *OpenCable Application Platform* (OCAP), *Advanced Common Application* (ACAP), e o japonês ARIB (*Association of Radio Industries and Businesses*). Mais recentemente o Brasil decidiu por especificar e usar um *middleware* próprio, o Ginga (3) (2).

Neste trabalho são investigadas estratégias que contribuem para melhorar a experiência de interação entre o usuário e as aplicações de televisão interativa, especialmente, estratégias complementares ou alternativas ao uso do controle remoto. A solução encontrada apresenta melhorias em relação tanto a usabilidade quanto no enriquecimento da experiência de uso das aplicações para televisão.

O uso do controle remoto tradicional como meio de interação com as aplicações de televisão interativa é questionado nas referências (5) e (7). Jääskeläinen (2001) em seu trabalho de análise do estado da arte e dos problemas encontrados no desenvolvimento de aplicações interativas para televisão, levanta os seguintes pontos:

- Os desenvolvedores de aplicações de TVDI devem levar em consideração com quais tipos de dispositivos a audiência pode interagir.
- Os desenvolvedores de aplicações de TVDI devem levar em consideração de que maneira a audiência poderia interagir por meio de dispositivos móveis.

Fazemos dos questionamentos levantados por Jääskeläinen as perguntas centrais para o presente trabalho. Nesse caso, buscamos uma solução que permita aos desenvolvedores de aplicações de TVDI fazer uso de novos recursos de interação, no caso dispositivos móveis, os quais poderão ser utilizados para aumentar as possibilidades de interação do usuário com a televisão.

O projeto do Ginga teve como meta refletir as ansiedades e necessidades requeridas pela sociedade brasileira, no que tange seu ambiente de televisão digital. Este projeto foi conduzido com base no levantamento dessas necessidades e na idealização de novas funcionalidades. Uma dessas idéias é justamente a motivação e principal contribuição desse trabalho: que é uma API que permite a interação entre o usuário e a televisão por meio de dispositivos móveis. Esta API torna possível o desenvolvimento de aplicações interativas multidispositivo e multiusuário. Uma aplicação com estas características permite que mais de um usuário, através de dispositivos distintos, interajam simultaneamente com um receptor de TV Digital.

Em termos de arquitetura, o Ginga é composto por dois ambientes para processamento de aplicações distintos: uma máquina de apresentação e uma máquina de execução. A máquina de apresentação é responsável pelas aplicações declarativas que fazem uso da API Ginga-NCL (4). A máquina de execução é responsável pelas aplicações procedurais, que são desenvolvidas utilizando a API Ginga-J (5), que é a API baseada em Java, onde o presente trabalho se insere.

1.1 OBJETIVOS

1.1.1 Objetivo geral

O objetivo geral do trabalho foi especificar e validar API (*Application Program Interface*) que adicionou ao *middleware* Ginga a capacidade de interagir com dispositivos móveis. Esta API tornou possível o desenvolvimento e a execução de aplicações interativas multidispositivo e multiusuário em aparelhos receptores de TV.

1.1.2 Objetivos específicos

Para que fosse possível especificar uma API Ginga que permitisse a integração do receptor de TVDI com dispositivos móveis, foi necessário:

- Estudar os ambientes de TVD e TVDI.
- Estudar os *middlewares* de TVDI existentes e suas APIs.
- Avaliar as APIs dos *middlewares* existentes, com foco no suporte a interação com dispositivos móveis.
- Estudar as tecnologias para implementação de *software* para dispositivos móveis, com o objetivo de entender os recursos disponíveis em cada uma delas em termos de linguagens de programação, protocolos e limitações específicas.
- Especificar a API Ginga-J para integração de dispositivos.
- Especificar uma estratégia para integração da API Ginga-J com dispositivos móveis, que contemplasse tanto o lado do receptor fixo (*set-top box*), quanto o dispositivo móvel.
- Implementar a API Ginga-J com suporte a interação com dispositivos móveis.
- Implementar um componente do Ginga (instalado e executado no dispositivo móvel) integrado com a API proposta.
- Testar a estratégia proposta através do desenvolvimento de uma implementação de referência.
- Validar a estratégia como um todo através da implementação de aplicações de teste.

1.2 CONTRIBUIÇÕES

A principal contribuição deste trabalho foi a especificação da API incluída no texto do padrão de *middleware* do Sistema Brasileiro de TV Digital "*Televisão digital terrestre - Codificação de dados e especificações de transmissão para transmissão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais*", que foi aprovado em consulta pública promovida pela ABNT (Associação Brasileira de Normas Técnicas) e aguarda aprovação na Comissão Especial de TV Digital da mesma.

Esta API viabiliza o suporte a recursos multimídia e interação simultâneas de múltiplos usuários em aplicações de TVDI (12). Nas seções que seguem estas contribuições são mais bem explicadas.

1.2.1 Recursos Multimídia

Alguns dispositivos móveis, como os celulares e PDAs, trazem consigo muitos recursos de captura e exibição de diferentes tipos de mídia. Dentre estes recursos podemos citar:

- Microfone: usado para entrada de áudio
- Tela: usado para exibição de conteúdo, texto ou vídeo
- Alto falantes: usado para reprodução de som
- Teclado: para entrada de comandos

Devido à forma como foi especificada a API de Integração de Dispositivos, cada interação do dispositivo móvel com o Ginga é recebida e tratada da mesma forma como é feita com o controle remoto. Percebeu-se então que a comunicação entre o dispositivo e o *middleware* poderia trazer dados binários que representassem tanto teclas como som, vídeo, imagem, posicionamento GPS ou outros tipos de informação. Essa possibilidade viabilizou a integração de dispositivos capazes de capturar diferentes tipos de mídia com as TVs.

1.2.2 Aplicações Multiusuário

A API de Integração não restringe a quantidade de dispositivos móveis conectados ao aparelho de TV, dessa forma as aplicações contam com a possibilidade de se comunicar com

mais de um dispositivo simultaneamente. Adicionalmente, a aplicação é capaz de identificar de qual dispositivo cada interação foi realizada.

Essa característica torna possível que aplicações de TVDI interajam simultaneamente com mais de um usuário, viabilizando o desenvolvimento de aplicações multiusuário. É importante ressaltar que não estamos falando de aplicações multiusuário que contam com um servidor de *backend* (ou retaguarda) para receber a interação de cada receptor, mixar, e enviar o resultado para eles; estamos falando de aplicações multiusuário executando localmente em cada receptor de televisão.

Essa contribuição é importante no ambiente residencial, uma vez que assistir televisão é normalmente uma experiência coletiva (5), como conseqüência, é interessante que cada membro da família possa expressar sua opinião ao interagir com a aplicação individualmente. Por exemplo num Quiz, cada participante pode votar individualmente e receber sua pontuação separadamente. Podemos também tomar como exemplo o Big Brother (13), neste caso, a aplicação poderia contabilizar mais de um voto por aparelho de TV, caso cada telespectador utilize um dispositivo distinto para interagir.

1.2.3 Perfis de Usuários

Devido a possibilidade de identificar individualmente a interação proveniente de cada dispositivo móvel, é possível usar essa característica para implementar um comportamento baseado em perfis.

Podemos imaginar, por exemplo, que ao utilizar seu celular para trocar de canal, o receptor perceba qual dispositivo fez a interação e, automaticamente, configure o receptor de TV com preferências de cor, volume e outras gravadas a partir de interações anteriores do dispositivo identificado. Essa característica também foi observada por Roibás (7), uma vez que os dispositivos móveis são quase sempre pessoais, os sistemas que com eles se comunicam podem identificar usuários através da identificação de dispositivos, podendo assim oferecer serviços personalizados.

1.3 ESTRUTURA DA DISSERTAÇÃO

A presente dissertação foi dividida em cinco capítulos

No Capítulo 1 fazemos a introdução ao tema do trabalho, apresentando a motivação que nos levou a decidir explorar o tema proposto. Apresentamos ainda uma breve justificativa da relevância do tema em relação ao atual estado da arte ao qual o problema se insere. Além disso, introduzimos as principais contribuições conseguidas com o desenvolvimento do trabalho.

No Capítulo 2 apresentamos um breve histórico sobre a TV Digital e a TV Digital Interativa, apresentando os principais componentes que compõem as duas tecnologias e a evolução do conceito em termos de soluções existentes. O foco do capítulo são os *middlewares* abertos de TVDI existentes atualmente, apresentando as principais características das iniciativas mais importantes conhecidas.

O Capítulo 3 é integralmente dedicado ao Ginga, *middleware* do Sistema Brasileiro de Televisão Digital. Apresentamos um breve histórico e apresentamos as principais características técnicas do padrão, como arquitetura e o contexto no qual se insere. O objetivo desse capítulo é apresentar o Ginga em maior profundidade, uma vez que a API proposta nesse trabalho se integra as especificações desse *middleware*.

No Capítulo 4 fazemos a apresentação completa da API proposta. Mostrando seu contexto e as funcionalidades que ela propõe. Apresentamos os diagramas da especificação e detalhes da API com comentários sobre todas as suas classes e métodos.

No Capítulo 5 apresentamos algumas aplicações pilotos desenvolvidas com o objetivo de validar e testar a API proposta. As aplicações foram desenvolvidas especificamente para explorar características da API. Como ilustração são mostradas imagens de suas interfaces sendo executadas tanto em receptores de TV quanto nos dispositivos móveis.

Por fim, no Capítulo 6 apresentamos as considerações finais do trabalho.

2 MIDDLEWARE PARA TV DIGITAL

As tecnologias de TVDI permitem a fabricação de STBs com diferentes arquiteturas de hardware, cujas capacidades de processamento, armazenamento e comunicação são bastante variáveis. Neste cenário de hardware e software heterogêneos, os desenvolvedores de aplicações deveriam escrever diferentes versões dos programas para cada combinação de hardware e sistema operacional dos diversos tipos de STBs. Conseqüentemente, a heterogeneidade das plataformas tornaria o desenvolvimento de aplicações para TVDI uma atividade ineficiente e de custo elevado, que poderia inviabilizar sua adoção em larga escala.

Para tornar mais eficiente o desenvolvimento de aplicações, bem como reduzir os custos associados, favorecendo assim a consolidação da TVDI, os fabricantes e provedores de conteúdo perceberam que a solução é adotar mecanismos que tornem portáveis as aplicações e os serviços nos diversos tipos de STBs.

Neste sentido, para atender ao requisito de portabilidade, os STBs devem ser equipados com um *middleware* comum, que, como vimos, é responsável por prover às aplicações uma API genérica, padronizada e bem definida, conforme ilustrado na Figura 2.

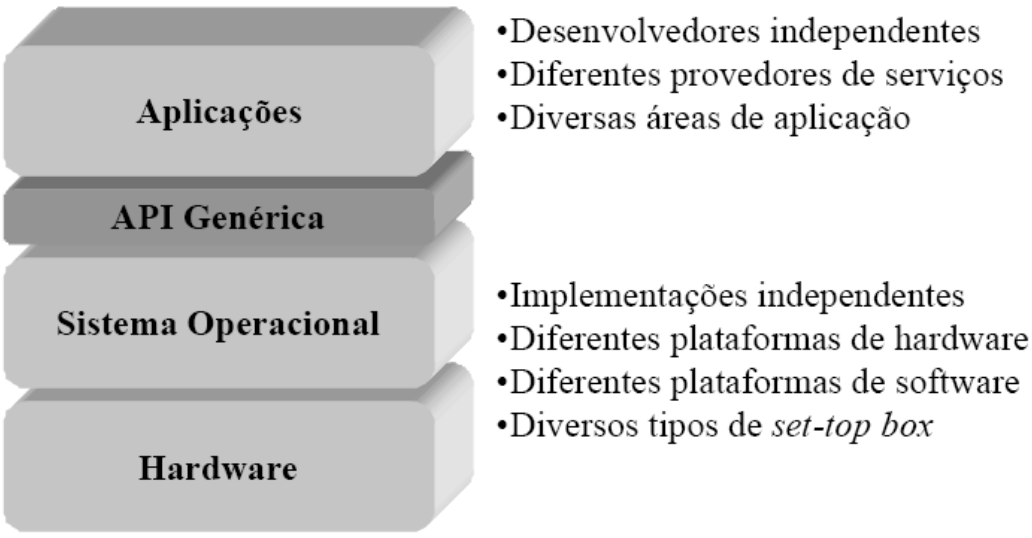

Figura 2: Portabilidade baseada em API genérica, abstraindo as especificidades e heterogeneidades de hardware e software dos diversos tipos de dispositivos receptores.

Foi em função dos benefícios da adoção de uma camada de *middleware*, que diversos órgãos de padronização concentraram esforços na especificação de padrões de *middleware*. Como resultado destes esforços surgiram os padrões, apresentados na seção 2.2.

É importante frisar que, apesar da existência de padrões, para ser suportado nos diversos tipos de STBs, um determinado padrão de *middleware* deve ser embarcado em cada plataforma de hardware e sistema operacional. Embora as aplicações se tornem portáveis entre diferentes plataformas de hardware e sistema operacional, elas se tornam dependentes do *middleware* adotado. Ou seja, uma aplicação desenvolvida para o *middleware* MHP não é diretamente portável para o DASE e o ARIB.

2.1 BLOCOS FUNDAMENTAIS

Embora os diversos padrões de *middleware* não sejam compatíveis entre si, estes padrões adotam versões modificadas, reduzidas ou estendidas de determinados componentes. Dentre estes blocos fundamentais comuns, podemos destacar os seguintes componentes: DAVIC - *Digital Audio-Visual Council* (14); HAVi - *Home Audio Video Interoperability* (15) e Java TV (16). Em função da ampla adoção destes blocos fundamentais, antes de descrever os padrões de televisão digital, apresentamos uma breve descrição das principais funcionalidades do DAVIC, HAVi e Java TV.

2.1.1 DAVIC

DAVIC é uma associação que representa diversos setores da indústria audiovisual, que foi iniciada em 1994, mas extinta após 5 anos de atividade, conforme já previsto no seu estatuto. O principal objetivo da associação DAVIC foi especificar um padrão da indústria para interoperabilidade fim-a-fim de informações audiovisual digital interativa e por difusão.

Para obter esta interoperabilidade, as especificações DAVIC (14) definem interfaces para diversos pontos de referência, onde cada interface é definida por um conjunto de fluxos de informações e protocolos. As especificações DAVIC especificam formatos de conteúdos para diversos tipos de objetos (fonte, texto, hipertexto, áudio e vídeo) e também incluem facilidades para controlar a língua adotada no áudio e na legenda.

Além disso, as especificações DAVIC definem diversas APIs relacionadas a informações de serviços, filtragem de informações, notificação de modificações nos recursos, sintonização de canais de transporte (*tuning*) e controle de acesso:

- *Service Information API*: provê às aplicações uma interface de alto nível para acessar informações de serviços presentes em fluxos MPEG-2. Esta API define métodos para acessar todas as informações presentes nas tabelas de serviços (*SI Tables*), permitindo, por exemplo, que um guia de programação eletrônico (*EPG – Electronic Program Guide*) possa identificar o escalonamento dos programas de cada serviço.
- *MPEG-2 Section Filter API*: permite as aplicações identificarem a ocorrência de determinados padrões nos dados mantidos em seções privadas MPEG-2.
- *Resource Notification API*: define um mecanismo padrão para aplicações registrarem interesse em determinados recursos e serem notificadas de mudanças nestes recursos.
- *Tuning API*: especifica uma interface de alto nível para fisicamente sintonizar os diferentes fluxos de transporte.
- *Conditional Access API*: provê uma interface básica para o sistema de controle de acesso. Por exemplo, esta API permite a aplicação verificar se o usuário possui direito de acesso a um dado serviço ou evento.
- *DSM-CC User-to-Network API*: define mecanismos para que as aplicações possam controlar as sessões DSM-CC.

Para apresentação de saída gráfica, as especificações DAVIC adotam um subconjunto do pacote AWT de interface com o usuário da API Java. Para apresentar fluxos de áudio e vídeo, as especificações DAVIC adotam o JMF - *Java Media Framework* (17) e definem algumas extensões para características específicas de televisão digital. Por exemplo, as especificações definem APIs para sincronizar aplicações em um instante específico de tempo de um determinado conteúdo e gerenciar eventos incluídos no conteúdo ou início da apresentação de uma determinada mídia.

2.1.2 HAVi

HAVi é uma iniciativa das oito maiores companhias de produtos eletrônicos que especifica um padrão para interconexão e interoperação de dispositivos de áudio e vídeo digital. A especificação (15) permite que os dispositivos de áudio e vídeo da rede possam interagir entre si, como também define mecanismos que permitem que as funcionalidades de

um dispositivo possam ser remotamente usadas e controladas a partir de outro dispositivo. No entanto, para essa comunicação, o HAVi especifica que uma rede IEEE-1394 – *Firewire* (18) deva ser utilizada.

A especificação HAVi é independente de plataforma e linguagem de programação, podendo ser implementada em qualquer linguagem para qualquer processador e sistema operacional. Desta forma, a especificação HAVi permite que os fabricantes projetem dispositivos interoperáveis e os desenvolvedores de aplicações possam escrever aplicações Java para estes dispositivos usando a API provida pelo HAVi.

No contexto de televisão digital interativa, o STB pode ser conectado em uma rede HAVi, podendo compartilhar seus recursos com outros dispositivos e usar os recursos de outros dispositivos para compor aplicações mais sofisticadas. Por exemplo, um STB pode gerar um menu completo que permite ao usuário acessar funcionalidades de qualquer dispositivo ou uma combinação de dispositivos HAVi, usando somente o controle remoto da televisão e apresentando o sistema de forma consistente para o usuário. Como outro exemplo, um STB pode automaticamente programar o aparelho de vídeo cassete a partir das informações obtidas do guia de programação eletrônico (*EPG – Electronic Program Guide*).

HAVi adota o padrão de rede *Firewire*, que atualmente suporta uma taxa de transmissão de até 400Mbps e é capaz de suportar comunicação isócrona, tornando-o adequado para tratamento simultâneo de múltiplos fluxos de áudio e vídeo digital em tempo real.

A especificação HAVi define uma arquitetura de software distribuída cujos elementos de software asseguram a interoperabilidade e implementam serviços básicos tais como: gerenciamento da rede, comunicação entre dispositivos e gerenciamento da interface com os usuários. HAVi define um conjunto serviços distribuídos que suportam APIs Java padronizadas, permitindo que aplicações distribuídas possam transparentemente acessar os serviços através da rede. Para assegurar a interoperabilidade, todos os elementos de software se comunicam usando um mecanismo de passagem de mensagem que adota formatos de mensagens e protocolos padronizados pelo HAVi. Os elementos de software da arquitetura HAVi são:

- *1394 Communication Media Manager*: coordena a comunicação assíncrona e isócrona em uma rede IEEE-1394.
- *Messaging System:* responsável pela passagem de mensagens entre os diversos elementos de software.
- *Registry*: define um serviço de diretório que permite localizar os diversos elementos de software na rede e identificar suas funcionalidades e propriedades.
- *Event Manager*: implementa um serviço de notificação de eventos, que sinaliza mudanças no estado dos elementos de software ou na configuração da rede HAVi.
- *Stream Manager*: gerencia a transferência em tempo real de fluxos de áudio e vídeo entre elementos de software.
- *Resource Manager*: controla o compartilhamento de recursos e realiza o escalonamento de ações.
- *Device Control Module (DCM)*: representa um dispositivo da rede HAVi e expõe as APIs deste dispositivo. Cada dispositivo da rede HAVi possui um DCM associado.
- *DCM Manager*: coordena a instalação e remoção de DCMs.

2.1.3 Java TV

Java TV (16) é uma extensão da plataforma Java que permite a produção de conteúdo para televisão interativa. O principal objetivo de Java TV é viabilizar o desenvolvimento de aplicações interativas portáveis, que são independentes da tecnologia de rede de difusão.

Java TV consiste de uma máquina virtual Java JVM – *Java Virtual Machine* (19) e várias bibliotecas de códigos reusáveis e específicos para televisão digital interativa. A JVM é hospedada e executada no próprio STB. Java TV foi desenvolvido sobre o ambiente J2ME (20), que foi projetado para operar em dispositivos com restrições de recursos. Neste contexto, determinadas características da plataforma Java foram excluídas, pois são consideradas desnecessárias ou inadequadas para tais ambientes. No entanto, o J2ME não define funcionalidades específicas de televisão. Tais funcionalidades são incluídas no Java TV.

Java TV permite níveis avançados de interatividade, gráficos de qualidade e processamento local no próprio STB. Estas facilidades oferecem um amplo espectro de

possibilidades para os desenvolvedores de conteúdo, mesmo na ausência de um canal de retorno. Por exemplo, EPGs podem oferecer uma visão geral da programação disponível, permitindo a mudança para o canal desejado pelo usuário. Através de mecanismos de sincronização, aplicações específicas podem ser associadas a um determinado programa de televisão. Além disso, aplicações isoladas podem executar de forma independente do programa de televisão.

Em Java TV, programas de televisão tradicionais e interativos são caracterizados como um conjunto de serviços individuais. Um serviço é uma coleção de conteúdo para apresentação em um STB. Por exemplo, um serviço pode representar um programa de televisão convencional, com áudio e vídeo sincronizados, ou um programa de televisão interativa, que contém áudio, vídeo, dados e aplicações associadas.

Uma aplicação Java TV é denominada *Xlet*. *Xlets* não precisam estar previamente armazenados no STB, pois podem ser enviados pelo canal de difusão quando necessários. Ou seja, o modelo *Xlet* é baseado na transferência de código executável pelo canal de difusão para o STB e posterior carga e execução do mesmo, de forma automática ou manual. Um *Xlet* é bastante similar a um *Applet* na *Web* ou *MIDlet* em celulares e outros dispositivos móveis.

2.2 PADRÕES DE MIDDLEWARE PARA TVDI

Nesta seção serão descritas as características principais dos padrões abertos de *middleware* para televisão digital terrestre definidos até a presente data.

2.2.1 DVB/MHP

Na camada de *middleware*, o padrão DVB adota o MHP, cuja especificação é denominada DVB-MHP (*Digital Video Broadcasting – Multimedia Home Platform*) (21). A plataforma MHP começou a ser especificada pelo projeto DVB em 1997. No entanto, a primeira versão (MHP 1.0) foi oficialmente lançada em junho de 2000. Após um ano do lançamento da primeira versão, em junho de 2001, foi lançada uma nova especificação (MHP 1.1). Em junho de 2003, foi lançada a versão 1.1.1 do MHP.

O MHP define uma interface genérica entre as aplicações e o *set-top Box* (*hardware* e sistema operacional), no qual as aplicações são executadas. Além disso, o MHP define o

modelo e o ciclo de vida das aplicações, como também os protocolos e os mecanismos para distribuição de dados em ambientes de televisão pseudo-interativa e interativa.

Nas versões 1.1 e 1.1.1, o MHP provê funcionalidades adicionais em relação à versão inicial, incluindo, por exemplo, a possibilidade de carregar programas interativos através do canal de retorno e o suporte opcional a aplicações desenvolvidas usando uma linguagem declarativa.

A partir da versão 1.1, o MHP adota modelos de aplicações baseados em linguagens procedural e declarativa. No modelo procedural, o MHP suporta a execução de aplicações Java TV, denominadas DVB-J. No modelo declarativo, opcionalmente, o MHP suporta a execução de aplicações desenvolvidas com tecnologias relacionadas à linguagem HTML, denominadas DVB-HTML.

O MHP foi utilizado como base para a especificação do GEM; *middleware* para TVDI que tem o objetivo de prover um ambiente interoperável global, harmonizando os padrões existentes.

2.2.2 DASE/ATSC

Na camada de *middleware*, o padrão ATSC, da própria ATSC (*Advanced Television System Committee*), adota o DASE (*DTV Application Software Environment*) (22), definindo uma camada de software que permite a programação de conteúdo e aplicações. O DASE adota modelos de aplicações baseados em linguagens procedural e declarativa. No modelo procedural, o DASE suporta a execução de aplicações Java TV. No modelo declarativo, o DASE suporta a execução de aplicações desenvolvidas em uma versão estendida da linguagem HTML. O DASE foi descontinuado, sendo substituído no ATSC pelo ACAP.

2.2.3 OCAP e ACAP

O OCAP (*OpenCable Application Platform*) (23) e ACAP (*Advanced Common Application Platform*) (22) foram *middlewares* que surgiram no mercado americano como alternativa ao DASE. Os dois *middlewares* foram definidos com base na iniciativa GEM, cujo objetivo foi de gerar um núcleo comum de APIs para o cenário das aplicações interativas para televisão.

O ACAP e OCAP são então *middlewares* compatíveis com o GEM. Além das APIs definidas no GEM, esses middlewares introduzem ainda algumas APIs próprias, específicas do sistema americano. O ACAP é usado para transmissão terrestre e o OCAP para transmissão em TV a Cabo.

2.2.4 ARIB/ISDB

Na camada de *middleware*, o padrão ISDB (*Integrated Services Digital Broadcasting*) adota a plataforma padronizada pelo ARIB (*Association of Radio Industries and Businesses*) (25), definindo uma camada de software que permite a programação de conteúdo e aplicações. O ARIB adota um modelo de aplicação baseado na linguagem declarativa denominada BML (*Broadcast Markup Language*), que, por sua vez, é baseada na linguagem XML (*Extensible Markup Language*). A parte procedural do ARIB foi baseada no GEM, junto com a definição de APIs específicas do ISDB.

2.3 RECOMENDAÇÕES ITU

Com o intuito de promover a interoperabilidade, a ITU (*International Telecommunication Union*) publicou o conjunto de recomendações ITU-T: J.200 (26), J.201 (27) e J.202 (28). Essas recomendações visam harmonizar os *middlewares* dos sistemas de TV Digital em diferentes níveis: arquitetura e definições para *middleware* declarativo e procedural (2).

Tais recomendações incluem, na arquitetura proposta para o receptor, uma divisão em dois ambientes para execução de aplicações interativas: um declarativo e outro procedural. Conforme ilustrado na Figura 3, a arquitetura recomendada posiciona esses ambientes de execução logo acima dos serviços especializados disponibilizados pelo receptor de TV. Os dois ambientes não são necessariamente independentes, uma vez que a recomendação inclui uma "ponte" que deve disponibilizar APIs para intercomunicação entre os esses dois ambientes. Essa API de ponte permite que as aplicações declarativas utilizem serviços disponíveis nas aplicações procedurais, e vice-versa. Considerando que as aplicações são executadas na camada acima da camada dos ambientes de execução, seguindo a recomendação ITU, é possível que as aplicações possuam uma arquitetura híbrida, com partes declarativas e procedurais.

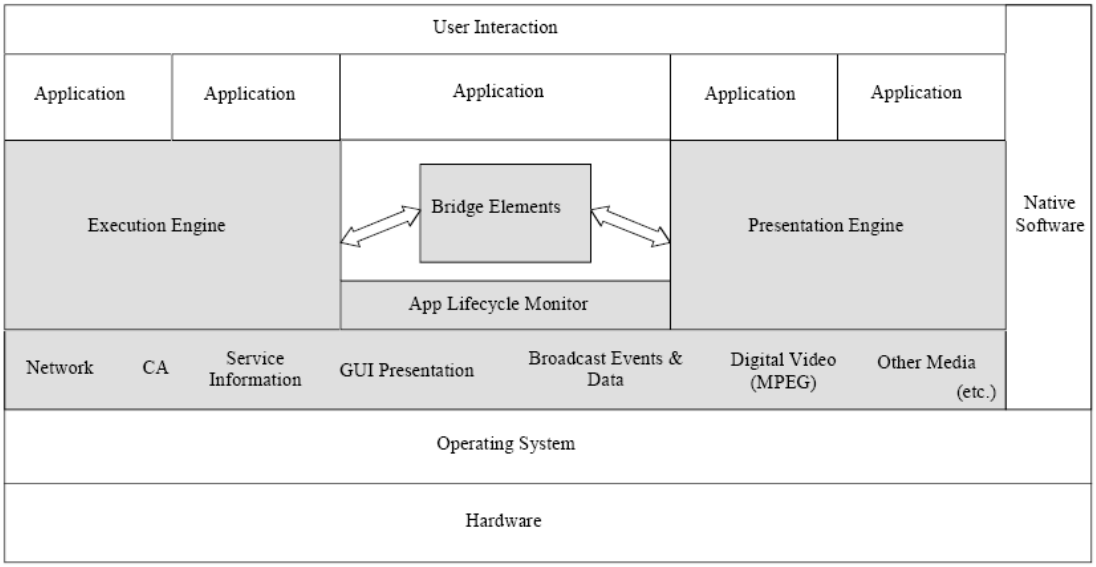

Figura 3: Arquitetura do ambiente de execução de aplicações, segundo recomendação ITU J.200 (22)

Seguindo as recomendações da ITU, os *middlewares* de TVD existentes possuem os dois ambientes de execução, o procedural e o declarativo. Estes ambientes aderem as recomendações ITU J.200, incluindo as APIs de harmonização definidas no documento para cada ambientes. Adicionalmente as APIs definidas na recomendação, cada *middleware* dispõe de um conjunto de adicional de APIs que são disponibilizadas para cada ambiente de execução. A Figura 4 mostra os *middlewares* de TVDI divididos em suas partes declarativa e procedural, relacionando em cada um as APIs adicionais definidas para os ambientes distintos. Devemos observar que as recomendações ITU estão localizadas na base da figura, indicando que estes blocos funcionais formam as bases para os ambientes de execução em cada padrão *middleware*.

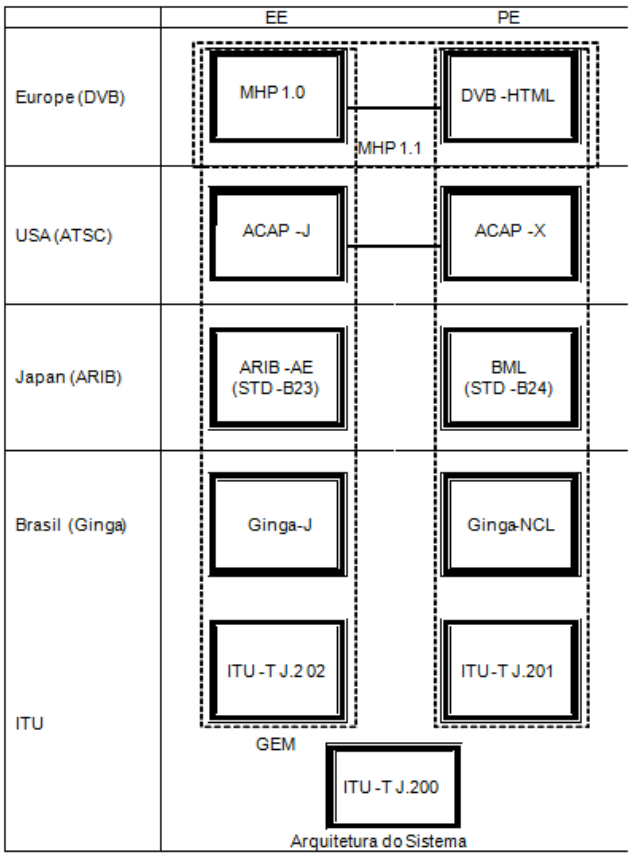

Figura 4: Visão geral das partes procedural e declarativas dos *middlewares* de TVDI.

Na arquitetura da parte procedural, a recomendações ITU J.202 (24) inclue as APIs do bloco comum já apresentado: DAVIC, JavaTV e HAVi, conforme ilustrado na Figura 5. A parte HAVi incluída, no entanto, abrange apenas os componentes e não a especificação completa; tal API é conhecida como *HAVi Level 2 User Interface*. Todas as APIs definidas para harmonização do ambiente procedural dos padrões de *middleware* de TVDI especificadas na recomendação J.202 (24) são mostradas na Figura 6.

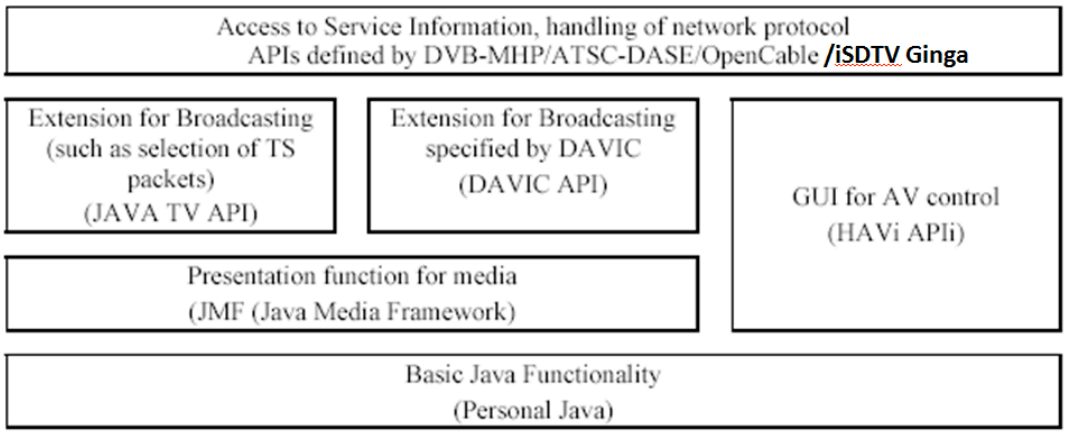

Figura 5: Arquitetura da máquina de execução proposta para os *middlewares* de TV Digital, segundo recomendação da ITU (26).

java.awt
java.awt.event
java.awt.image
java.beans
java.io
java.lang
java.lang.reflect
java.net
java.security
java.security.cert
java.util
java.util.zip
javax.media
javax.media.protocol
javax.tv.graphics
javax.tv.locator
javax.tv.media
javax.tv.media.protocol
javax.tv.net
javax.tv.service
javax.tv.service.guide
javax.tv.service.navigation
javax.tv.service.selection
javax.tv.service.transport
javax.tv.util
javax.tv.xlet
org.davic.media
org.davic.resources
org.havi.ui
org.havi.ui.event
java.math
java.rmi
java.security.spec
javax.net
javax.net.ssl
javax.security.cert
org.davic.mpeg

org.davic.mpeg.sections
org.davic.net
org.davic.net.dvb
org.davic.net.tuning
org.dvb.application
org.dvb.dsmcc
org.dvb.event
org.dvb.io.ixc
org.dvb.io.persistent
org.dvb.lang
org.dvb.media
org.dvb.net
org.dvb.net.tuning
org.dvb.net.rc
org.dvb.test
org.dvb.ui
org.dvb.user

Figura 6: APIs definidas na recomendação ITU J.202 (24) para harmonização dos ambientes de execução das aplicações procedurais dos *middlewares* de televisão digital.

Como já foi dito, cada padrão de *middleware* dispõe de um conjunto adicional de APIs ao bloco comum. Estas APIs provêem funcionalidades relacionadas às características particulares de cada padrão. Elas são necessárias para permitir que as aplicações possam controlar aspectos específicos de cada sistema. Por exemplo, um receptor DVB-S (que sintoniza canais via satélite) precisa ter, no MHP instalado nesse receptor, APIs que controlam a sintonização de satélite, uma vez que essa sintonização tem peculiaridades que a diferem dos sistemas terrestre e cabo. As APIs específicas dos *middlewares* DASE, MHP, OCAP, ARIB e Ginga-J são listadas na Figura 7.

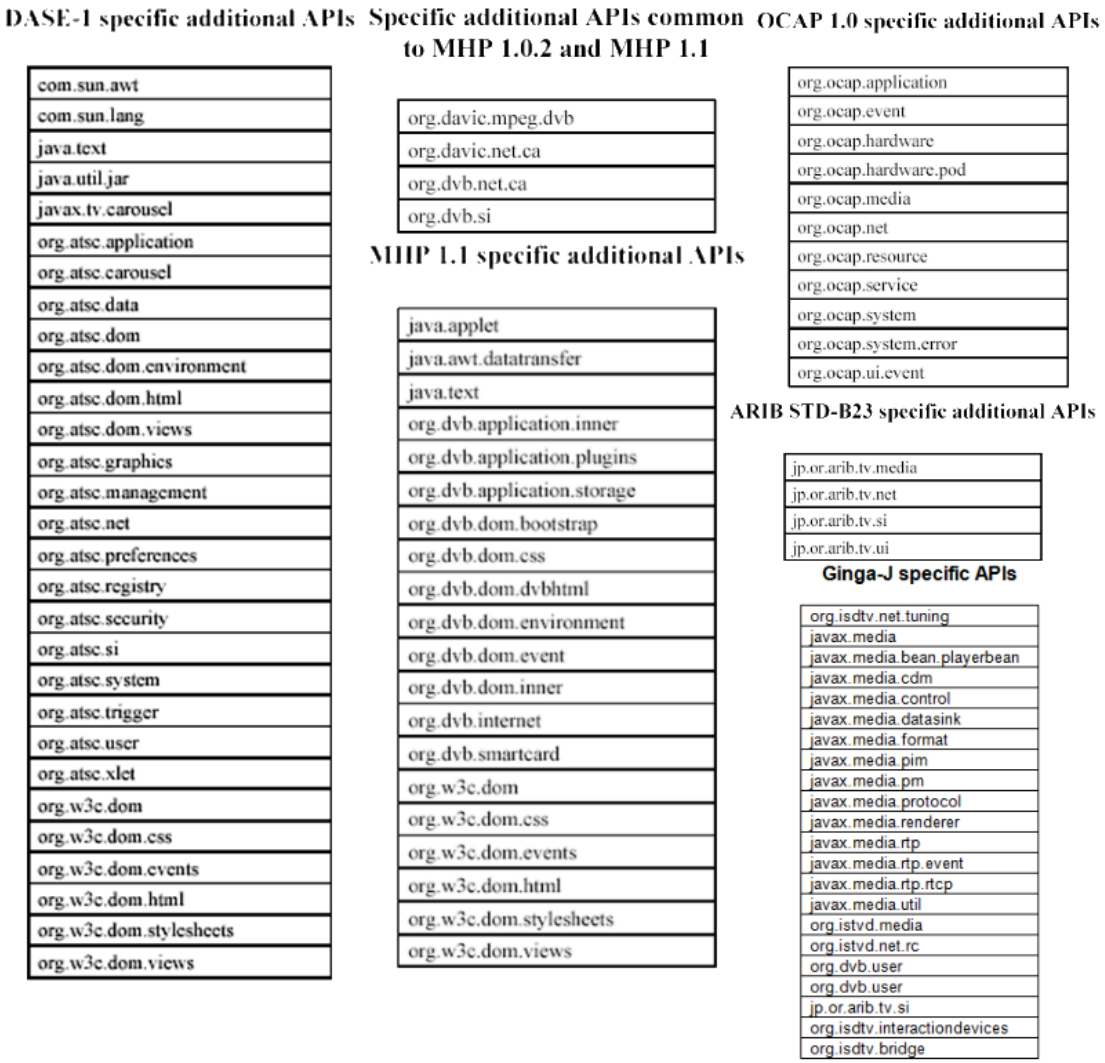

Figura 7: APIs específicas dos padrões de *middleware* DASE, MHP, OCAP, ARIB e Ginga.

2.4 DISCUSSÃO DAS FUNCIONALIDADES DISPONÍVEIS NOS MIDDLEWARES DE TV DIGITAL

De uma maneira geral,os *middlewares* existentes apresentam um conjunto de APIs que permitem a realização das mais diversas operações, tais como:

- Acesso de baixo nível do MPEG;
- Acesso a dados transmitidos no fluxo multiplexado;
- Controle de mídia e reprodução;
- Controle do ciclo de vida das aplicações;

- Controle dos gráficos e interface com usuário;
- Comunicação com servidores de *back-end* e outras aplicações;
- Acesso ao *hardware* do receptor e outros periféricos, como *smart cards;*
- Segurança.

Muitas dessas funcionalidades estão disponíveis nas APIs dos blocos comuns definidas no padrão GEM (25), apresentados na seção 2.1.

Dentre essas APIs, as APIs Havi são as mais próximas em termos de funcionalidades das que foram incluídas na API de Integração de Dispositivos do Ginga. O HAVi, no entanto, requer que todos dispositivos HAVi possuam uma interface de rede *Firewire*, por onde a comunicação entre os vários dispositivos é realizada. Esse pré-requisito exclui, inclusive, grande parte dos receptores de TV atualmente disponíveis no mercado. Esses receptores não são considerados dispositivos HAVi, e apenas usam a API de componentes gráficos, que implementa uma parte desta especificação.

Além da limitação em relação ao padrão de rede utilizado, a API de troca de mensagens e verificação de recursos e dispositivos é complexa e não faz parte das especificações de *middlewares* de TVDI conhecidos.

O HAVi não inclui suporte a integração com dispositivos móveis genéricos, como o celular.

No estudo realizado durante a realização do presente trabalho, verificou-se que uma boa estratégia para especificar a API de Integração de Dispositivos Ginga seria estender a API HAVi, como será explicado no Capítulo 5.

Para superar as limitações do HAVi, a API proposta permite a conexão entre dispositivos através de diferentes protocolos e redes, de forma transparente. Além disso, inclui suporte a verificação de quais dispositivos estão disponíveis e quais são suas funcionalidades.

3 MIDDLEWARE GINGA

Nesse capítulo apresentamos o detalhamento do *Middleware* GINGA, uma proposta que agrega características já existentes em outros *middlewares* com outras inovadoras. A apresentação enfatiza a arquitetura do GINGA, o GINGA-J, porção procedural do GINGA, na qual se insere a API resultante do trabalho aqui descrito.

O Laboratório de Aplicações de Vídeo Digital da UFPB (LAVID) participou ativamente da fase de estudos, escolha e especificação da proposta para o padrão brasileiro de televisão digital. O LAVID foi responsável pela coordenação do grupo de *middleware* do projeto FlexTV (29). Nessa ocasião, o grupo pôde identificar deficiências e pontos fortes de cada padrão de TV Digital existente. O trabalho no FlexTV, somado ao realizado no projeto MAESTRO (30), coordenado pela PUC-RIO, levou a especificação de um novo padrão de *middleware*, o Ginga, que foi adotado no Sistema Brasileiro de Televisão Digital.

3.1 ARQUITETURA

Segundo, Soares (2006), o universo das aplicações para televisão digital pode ser dividido em dois conjuntos: o das aplicações declarativas e o das aplicações procedurais. Uma aplicação declarativa é aquela em que sua entidade “inicial” é do tipo “conteúdo declarativo”. Analogamente, uma aplicação procedural é aquela em que sua entidade “inicial” é do tipo “conteúdo procedural” (30).

Um conteúdo declarativo deve ser baseado em uma linguagem declarativa, isto é, em uma linguagem que enfatiza a descrição declarativa do problema, ao invés da sua decomposição em uma implementação algorítmica. Por outro lado, um conteúdo procedural deve ser baseado em uma linguagem não declarativa. Linguagens não declarativas podem seguir diferentes paradigmas. Tem-se assim, as linguagens baseadas em módulos, orientadas a objetos etc. A literatura sobre televisão digital, no entanto, utiliza o termo procedural para representar todas as linguagens que não são declarativas. Numa programação procedural, o computador deve obrigatoriamente ser informado sobre cada passo a ser executado. Pode-se afirmar que, em linguagens procedurais, o programador possui um maior poder sobre o código, sendo capaz de estabelecer todo o fluxo de controle e execução de seu programa -

como existem mais recursos disponíveis o grau de complexidade é maior. A linguagem mais usual encontrada nos ambientes procedurais de um sistema de TV digital é a linguagem Java[2].

Na Figura 8 podemos visualizar os principais componentes da arquitetura GINGA. Cabe ressaltar que as facilidades da especificação Ginga devem ser destinadas à aplicação em sistemas e receptores de transmissão para transmissão terrestre (*over-the-air*). Além disso, a mesma arquitetura, e facilidades, podem ser aplicadas a outros sistemas de transporte (como sistemas de televisão via satélite ou cabo).

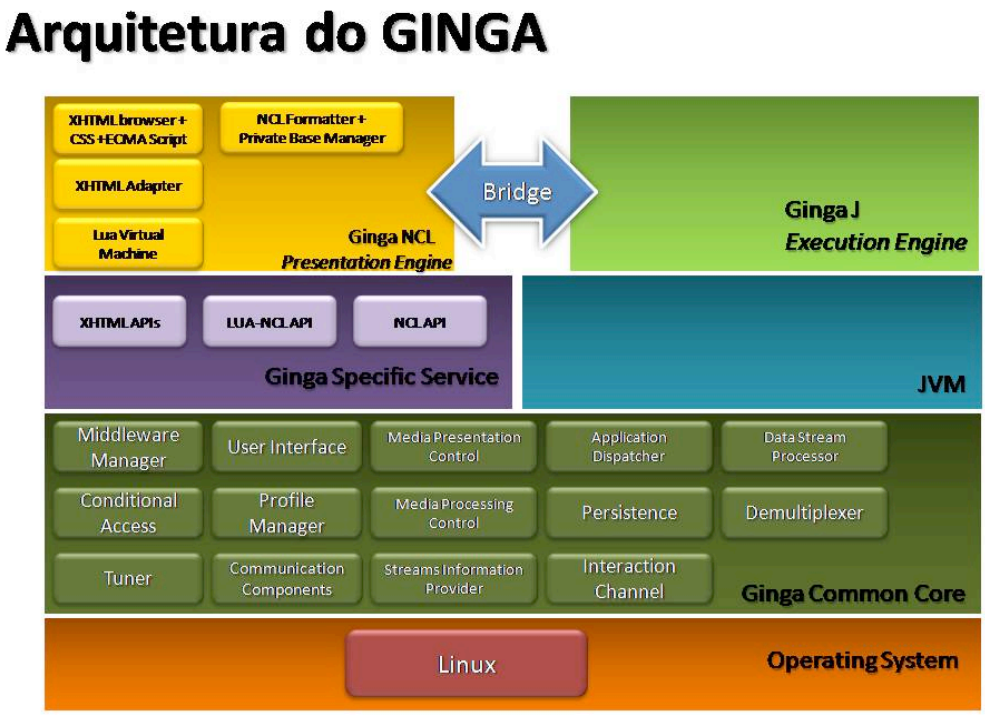

Figura 8:Arquitetura do GINGA

O Ginga-NCL (ou Máquina de Apresentação) é um subsistema lógico do Sistema Ginga que processa documentos NCL. Um componente-chave do Ginga-NCL é o mecanismo de decodificação do conteúdo informativo (NCL formatter). Outros módulos importantes são o baseado em XHTML, que inclui uma linguagem de estilo (CSS) e intérprete ECMAScript, e o mecanismo LUA, que é responsável pela interpretação dos scripts LUA.

[2] Java: HTTP:// java.sun.net

O Ginga-J (ou Máquina de Execução) é um subsistema lógico do Sistema Ginga que processa aplicações procedurais (Xlets Java). Um componente-chave do ambiente do aplicativo procedural é o mecanismo de execução do conteúdo procedural, que tem por base uma Máquina Virtual Java.

Decodificadores comuns de conteúdo devem servir para as necessidades de aplicativos tanto procedurais quanto informativos de decodificação e apresentação de conteúdos comuns do tipo PNG, JPEG, MPEG e outros formatos. O Ginga-Core é composto por decodificadores e procedimentos comuns de conteúdo para obter conteúdos transportados em fluxos de transporte MPEG-2 (TS) e através de um canal de retorno.

3.2 GINGA-J – A PARTE PROCEDURAL DO MIDDLEWARE BRASILEIRO

A Figura 9 apresenta o contexto em que o Ginga é executado. O Ginga é uma especificação de *middleware* distribuído, que reside em um dispositivo Ginga (dispositivo que embarque o *middleware* Ginga, como um receptor de televisão digital), com possibilidade de possuir componentes de *software* nos dispositivos de interação (celulares, PDA etc).

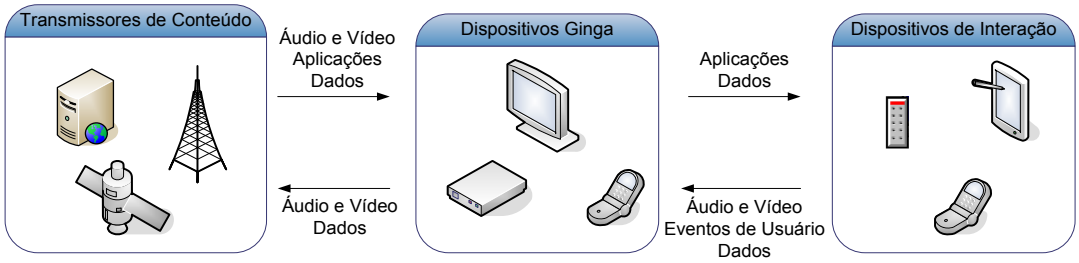

Figura 9: Contexto do GINGA

O dispositivo Ginga deve ter acesso a fluxos de vídeo, áudio, dados e outros recursos de mídia, que devem ser transmitidos através do ar, cabo, satélite ou através de redes IP. As informações recebidas devem ser processadas e apresentadas aos telespectadores.

O telespectador pode interagir com o dispositivo Ginga através de dispositivos de interação que podem conter componentes de *software* Ginga de forma que o dispositivo de interação possa enviar informações para o dispositivo Ginga utilizando as funcionalidades providas na especificação Ginga, ou seja, estes componentes de *software* que podem ser instalados nos dispositivos de interação permitem que as funcionalidades dos mesmos sejam exploradas, utilizando funcionalidades da API Ginga-J, por aplicações nos dispositivos Ginga

(receptores de televisão digital). Para que um dispositivo de interação possa ser utilizado, ele deve estar registrado com o dispositivo Ginga, e durante esse processo o dispositivo de interação pode receber o componente de *software* necessário para viabilizar a comunicação com o dispositivo Ginga.

Como resposta à informação enviada pelo telespectador, o dispositivo de Ginga deve apresentar a saída de vídeo e áudio utilizando seu próprio monitor e auto-falantes ou os dos dispositivos de interação. Um único dispositivo pode ter capacidade de entrada e saída simultâneas.

Por exemplo, um dispositivo de interação pode ser um PDA conectado à plataforma Ginga através de uma rede sem fio. Utilizando tal dispositivo de interação, um telespectador pode enviar comandos (eventos de usuário) à plataforma através do teclado PDA e os aplicativos da plataforma podem enviar conteúdo visual para ser apresentado na tela do PDA.

Um dispositivo de interação pode também ter capacidades de captura e reprodução de som, de forma que o telespectador possa enviar fluxos de áudio e vídeo para o dispositivo Ginga, utilizando os dispositivos de interação que dêem suporte à essa funcionalidade.

Vários telespectadores podem interagir com a plataforma Ginga simultaneamente. Nesse caso, cada telespectador pode ter um dispositivo de interação e a plataforma deve distinguir os comandos enviados por e para cada dispositivo. O dispositivo Ginga pode também enviar informações para os transmissores de conteúdo quando da existência de um canal de retorno (conexão com a Internet, por exemplo).

Como pode ser percebido, a interação com dispositivos diversos faz parte do contexto desejado para o Ginga. A utilização de tais recursos como meio de interação do usuário com o receptor Ginga foi pensada desde a concepção da arquitetura. Entretanto, para que as aplicações possam tirar vantagem dessa característica e oferecer funcionalidades nesse contexto de múltiplos dispositivos, uma API de Integração de Dispositivos precisa ser especificada. Essa API de Integração, alvo desse trabalho, será detalhada no próximo capítulo.

3.2.1 Arquitetura do Ginga-J

O modelo Ginga-J distingue entre as entidades e recursos de *hardware*, *software* do sistema e aplicativos conforme descrito na Figura 10.

As aplicações residentes (*native applications*) podem ser implementadas usando funções não padronizadas, fornecidas pelo Sistema Operacional (*real time OS*) do dispositivo de Ginga, ou por uma implementação particular do Ginga. Os aplicativos residentes também podem incorporar funcionalidades providas pelas API padronizadas Ginga-J. Aplicativos transmitidos (*Xlets*) sempre devem utilizar API padronizadas fornecidas pelo Ginga-J.

Em geral, o Ginga é alheio a quaisquer aplicativos residentes. Estas aplicações residentes incluem, mas não limitam-se a: *closed caption*, mensagens do sistema de acesso condicional (*Conditional Access* – CA), menus do receptor e guias eletrônicos de programação (*Eletronic Program Guide* – EPG) residente.

Os aplicativos residentes podem ter prioridade sobre os aplicativos Ginga. Como exemplo, o *closed caption* e mensagem de emergência devem ter prioridade no Sistema Ginga.

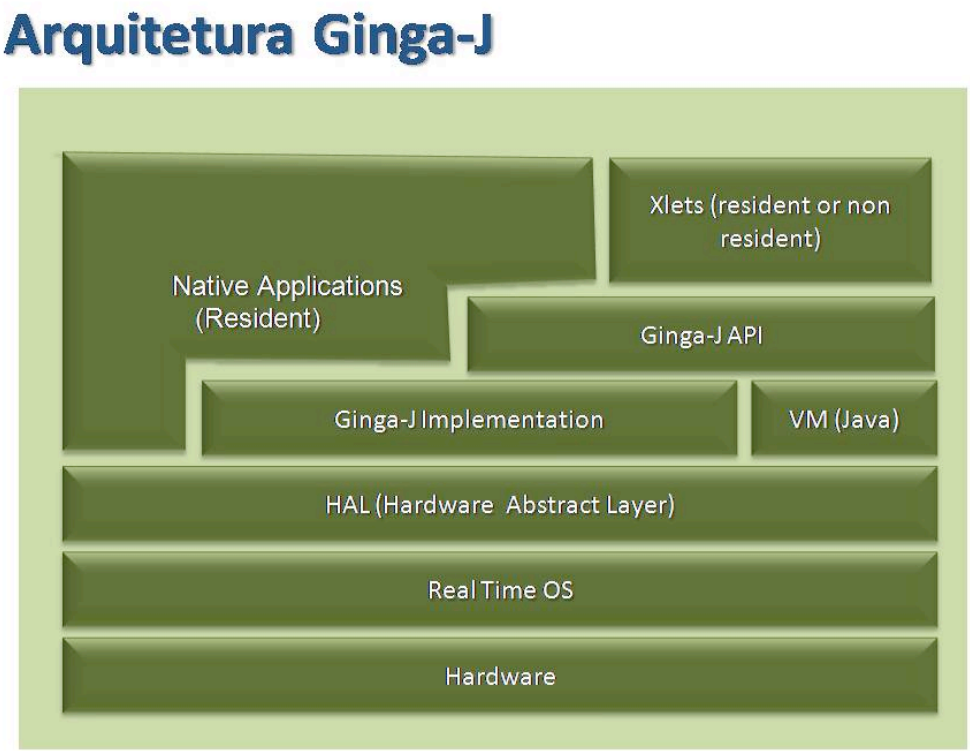

Figura 10: Arquitetura Ginga-J e ambiente de execução.

3.2.2 API Ginga-J

Alguns requisitos importantes identificados no contexto brasileiro não encontram funcionalidade correspondente nas definições estabelecidas internacionalmente para o *middleware*. Para poder preencher os requisitos específicos brasileiros e ainda manter compatibilidade internacional com as APIs do GEM, o Ginga baseia-se em três conjuntos de APIs, conforme ilustrado na Figura 11. As APIs são denominadas:

- APIs Verdes;
- APIs Amarelas;
- APIs Azuis.

As APIs Ginga Verdes são as APIs que são compatíveis com o GEM. As APIs Amarelas são extensões propostas para satisfazer os requisitos brasileiros específicos que podem ser implementados através do uso de um adaptador de *software* que use as APIs Verdes. As APIs Azuis são aquelas que não são diretamente compatíveis em *software* com as APIs do GEM.

Desta maneira, aplicações que utilizam apenas as APIs Verdes podem ser executadas em *middleware* Ginga, MHP, OCAP, ACAP e ARIB. As aplicações que utilizam APIs Verdes e Amarelas somente podem ser executadas apenas em MHP, ACAP, OCP e ARIB caso o adaptador de software necessário seja transmitido e executado junto com a aplicação. As aplicações que usem APIs Azuis somente serão executadas nos ambientes com *middleware* Ginga.

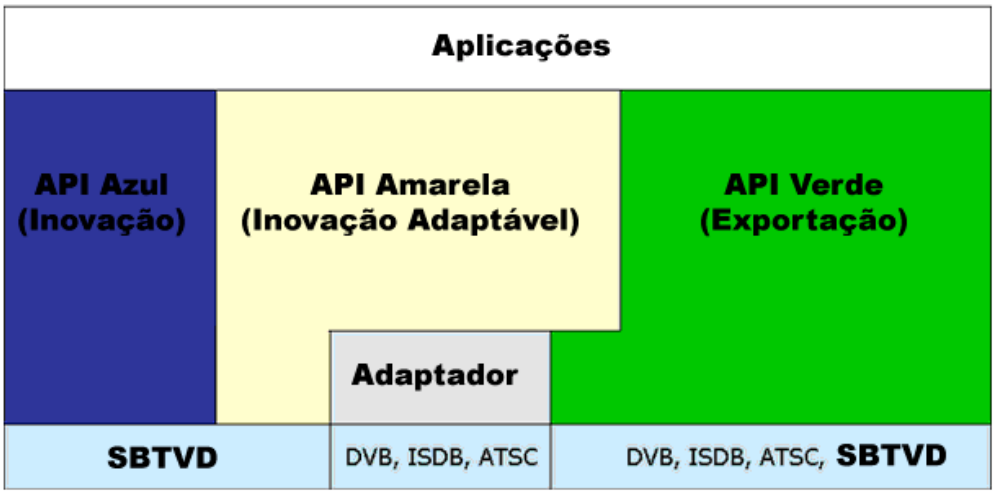

Figura 11: APIs Verdes, Amarelas e Azuis do Ginga-J.

O conjunto de APIs Verdes do Ginga-J é composto pelos pacotes Sun JavaTV, DAVIC, HAVi e DVB, todos incluídos na especificação da estrutura do GEM. O conjunto de APIs Amarelas do Ginga-J é composto pela API JMF 2.1, que é necessária para o desenvolvimento de aplicações avançadas - por exemplo com captura de som; por uma extensão à API de exibição do GEM, com funcionalidades para suportar as especificações do fluxo de vídeo definidas no padrão Ginga-J; por uma extensão à API do canal de retorno do GEM, que permite o envio de mensagens assíncronas; e por uma extensão à API do Serviço de Informação ISDB ARIB B.23.

O conjunto de APIs Azuis do Ginga-J é composto por uma API de Integração de Dispositivos (tema desse trabalho), que permite ao receptor de TV digital comunicar-se com qualquer dispositivo que possua uma interface compatível (sobre cobre, como Ethernet ou PLC; ou sem fio, como infravermelho ou *Bluetooth*), que possa ser utilizado como um dispositivo de entrada ou saída; por uma API Multiusuário, que utiliza a API de Integração de Dispositivos para permitir múltiplos usuários interagirem simultaneamente com aplicações de TV digital; uma API de Ponte NCL, que permite o desenvolvimento de aplicações Java contendo aplicações NCL.

Quando o governo brasileiro guiou as pesquisas no desenvolvimento do *middleware* de referência para a Televisão Digital Brasileira, ele determinou alguns requisitos importantes a serem preenchidos. Esses requisitos foram em sua maioria baseados em algumas particularidades do contexto social brasileiro.

Por exemplo, apenas 32,1 milhões de pessoas têm acesso à Internet, o que representa 21% da população brasileira - o governo brasileiro definiu então que a televisão digital deveria ser uma ferramenta para a inclusão digital, uma vez que a televisão está presente em 91% dos lares brasileiros (31).

Durante o desenvolvimento do *middleware* procedural de referência para o Sistema de TV Digital brasileiro foram conduzidos muitos estudos sobre as principais soluções de *middleware* para TV digital adotadas mundialmente e, uma vez que a maioria das especificações estava baseada nas especificações do GEM e do J.202, ficou claro que alguns requisitos não seriam alcançados, uma vez que o contexto europeu (que guiou o desenvolvimento do MHP) é muito diferente do brasileiro.

Atualmente o Brasil possui 79,5 milhões de telefones celulares (31). Um telefone celular pode ser utilizado como um canal de retorno para o ambiente de TV, ou usado como um controle remoto, ou usado como um dispositivo de interação (para responder a enquetes de maneira individual, por exemplo), etc. Uma vez que essas funcionalidades são todas implementadas utilizando protocolos comuns tais como *Bluetooth*, USB, *Wi-Fi*, etc., o Ginga é compatível com diversos dispositivos.

Dentro desse contexto, a inserção de mecanismos capazes de integrar a TVDI com dispositivos móveis é uma tendência de mercado. Aliado a esse fato as funcionalidades inovadoras do Ginga-J, providas por suas APIs, permitem o desenvolvimento de aplicações avançadas, através de extensões e da inserção de componentes adicionais. Dessa forma, considerou-se importante explorar a integração do *middleware* Ginga com dispositivos móveis tais como telefones celulares, PDAs, etc; o que foi feito através da API tema do proposto trabalho.

4 A API DE INTEGRAÇÃO DE DISPOSITIVOS DO GINGA-J

A API de Integração de Dispositivos agrega ao Ginga funcionalidades relacionadas à maneira como é feita a interação dos usuários com o receptor de TV Digital. A API faz parte da especificação do Ginga-J e foi completamente idealizada e especificada pelo grupo de trabalho responsável pelos estudos do *middleware* para o sistema brasileiro de TV Digital. Durante a fase de estudos, não foi identificada nenhuma especificação de API semelhante nos *middleware* de TV Digital conhecidos, sendo o Ginga o primeiro *middleware* que integra o receptor de TV Digital a estes tipos de dispositivos.

4.1 REQUISITOS

As funcionalidades inovadoras oferecidas pela API permitem o uso de diversos dispositivos de interação para comunicação com o receptor que hospeda o *middleware* Ginga e viabilizam que as aplicações interativas utilizem os recursos disponíveis nesses dispositivos. Tais dispositivos devem possuir um componente (módulo) do Ginga instalado – uma pequena parte móvel do Ginga que é responsável por gerenciar o protocolo de comunicação entre a instância Ginga no receptor de TV Digital e o componente do Ginga no próprio dispositivo. Dentre os possíveis dispositivos podemos citar celulares, PDAs, computadores portáteis e virtualmente qualquer outro dispositivo móvel com capacidade de processamento e comunicação; podemos imaginar "controles remotos avançados" compatíveis com o Ginga.

Para que os dispositivos possam ser efetivamente utilizados pelo Ginga eles precisam estar registrados junto ao *middleware*. O registro deve ser feito pela própria interface do sistema, que deve permitir a conexão de dispositivos através de diferentes redes: *Bluetooth*, *wi-fi*, infra-vermelho, USB entre outras, ficando a critério do desenvolvedor do receptor decidir que meios de conexão estarão disponíveis. O registro junto ao *middleware* só é possível para dispositivos que já possuem o módulo Ginga instalado, porém o processo de instalação também pode ser realizado pela interface do sistema: o componente móvel do Ginga pode ser um *Midlet*[3] num celular com tecnologia Java ou uma aplicação Symbian[4] para

[3] Midlet é nome dado a aplicações Java provenientes da plataforma J2ME para dispositivos com capacidade reduzida, como por exemplo, os celulares.

[4] Symbian é uma plataforma para desenvolvimento de aplicações para dispositivos móveis oriunda de um consórcio de empresas fabricantes de celulares, como: Nokia, Motorola e Sony Ericson, entre outras.

dispositivos com sistema operacional Symbian, e o módulo móvel do Ginga pode ser automaticamente transferido do receptor de TV Digital para o dispositivo onde o componente deve ser instalado.

Uma vez que um dispositivo esteja registrado junto ao *middleware* Ginga (conectado) sua interação com o mesmo pode ser feita de forma automática: os recursos presentes no dispositivo, como teclado, tela, microfone, câmera, auto-falantes e outros, estarão disponíveis para as aplicações através da API de Integração de Dispositivos do Ginga.

Como exemplo de uso simples, podemos conectar um celular ao Ginga e passar a usar o seu teclado como controle remoto e dispor de suas funcionalidades básicas – troca de canal, controle do volume do som, etc; em um outro caso uma aplicação interativa em execução poderá dispor de uma segunda tela de exibição (a tela do celular) para exibição de parte sua interface.

Os recursos dos dispositivos disponíveis para as aplicações dependem dos dispositivos conectados com o receptor – caso um celular possua câmera, será possível, por exemplo, que uma aplicação requisite a captura de uma imagem de tal dispositivo. A API dispõe de métodos para consulta de quais recursos estão disponíveis em cada dispositivo conectado.

A API de Integração de Dispositivos oferece a funcionalidade de identificar a origem de cada interação, relacionando-a a cada dispositivo individualmente. Essa característica viabiliza aplicações multiusuário.

Uma aplicação simples que pode explorar essa característica seria um Quiz (questionário). Tal aplicação comumente espera a resposta do usuário para cada pergunta por meio do clique no controle remoto (ou de qualquer dispositivo conectado, caso a aplicação esteja sendo executada no Ginga), e então exibe a próxima pergunta; ao final a pontuação do usuário é apresentada. No Quiz multiusuário, utilizando as APIs Ginga, é possível se identificar qual dispositivo originou cada resposta para o questionário, passando para as próximas perguntas apenas quando todos dispositivos individualmente respondessem a cada pergunta. Cada dispositivo representaria um usuário único da aplicação, e esta controla a pontuação de cada um desses usuários individualmente, e ao final do Quiz a pontuação de cada um seria exibida – inclusive nas telas dos dispositivos.

4.1.1 Compatibilidade com aplicações GEM

Um forte requisito considerado durante a fase de especificação e, posteriormente nas primeiras implementações de referência, era a de deixar sua interface a mais familiar possível, além de garantir que as aplicações existentes possam vir a utilizar a API Ginga com pouco impacto na sua reengenharia – poucas chamadas devem ser adicionadas ao código da aplicação.

A forma como os eventos de usuário são capturados pela aplicação é idêntica à utilizada pelos padrões de *middleware* compatíveis com o GEM. Seja usando AWT ou a classe *UserEvent* do sistema DVB-MHP, os Xlets existentes precisam apenas fazer um *downcasting*[5] para as classes equivalentes definidas na API e, assim, poderão identificar qual dispositivo gerou eventos para a aplicação. Para captura de eventos da forma convencional, nenhuma chamada no código precisa ser alterada – apenas para identificação do dispositivo teremos duas linhas de código adicionadas (com relação a uma aplicação GEM), como pode ser visualizado no código fonte de exemplo presente no capítulo 5.

Ainda na parte de dispositivos, a notificação de atividade dos mesmos é feita através do uso de ouvintes (*listeners*), o que facilita o trabalho do desenvolvimento já que essa estratégia é familiar aos desenvolvedores Java.

4.2 ESPECIFICAÇÃO

A especificação foi incorporada à Norma Ginga-J, aprovada em consulta pública da ABNT (Associação Brasileira de Normas Técnicas), porém ainda não publicada. Apesar das funcionalidades avançadas presentes na API, ela é bastante simples e apresenta uma interface familiar as interfaces já utilizadas para a construção de Xlets. Algumas funcionalidades foram baseadas no Havi e os desenvolvedores provavelmente não encontrarão problemas para usar a nova API.

A API especifica apenas um pacote, o pacote *br.org.sbtvd.interactiondevices*, que pode ser observado na Figura 12, cujas classes componentes são as seguintes:

[5] *Downcasting*: No campo das linguagens orientadas a objetos, chamamos de *downcasting* o ato de trocar a classe de um objeto para uma subclasse da classe referenciada originalmente. As linguagens só permitem o *downcasting* se o objeto puder ser representado pela subclasse requerida, ou seja, se o objeto realmente for pertencente a hierarquia da classe.

- Classe GRemoteDeviceManager;
- Classe GRemoteDevice;
- Interface GRemoteDeviceActionListener;
- Classe GRemoteEvent;
- Classe GRemoteKeyEvent;
- Classe GRemoteUserEvent.

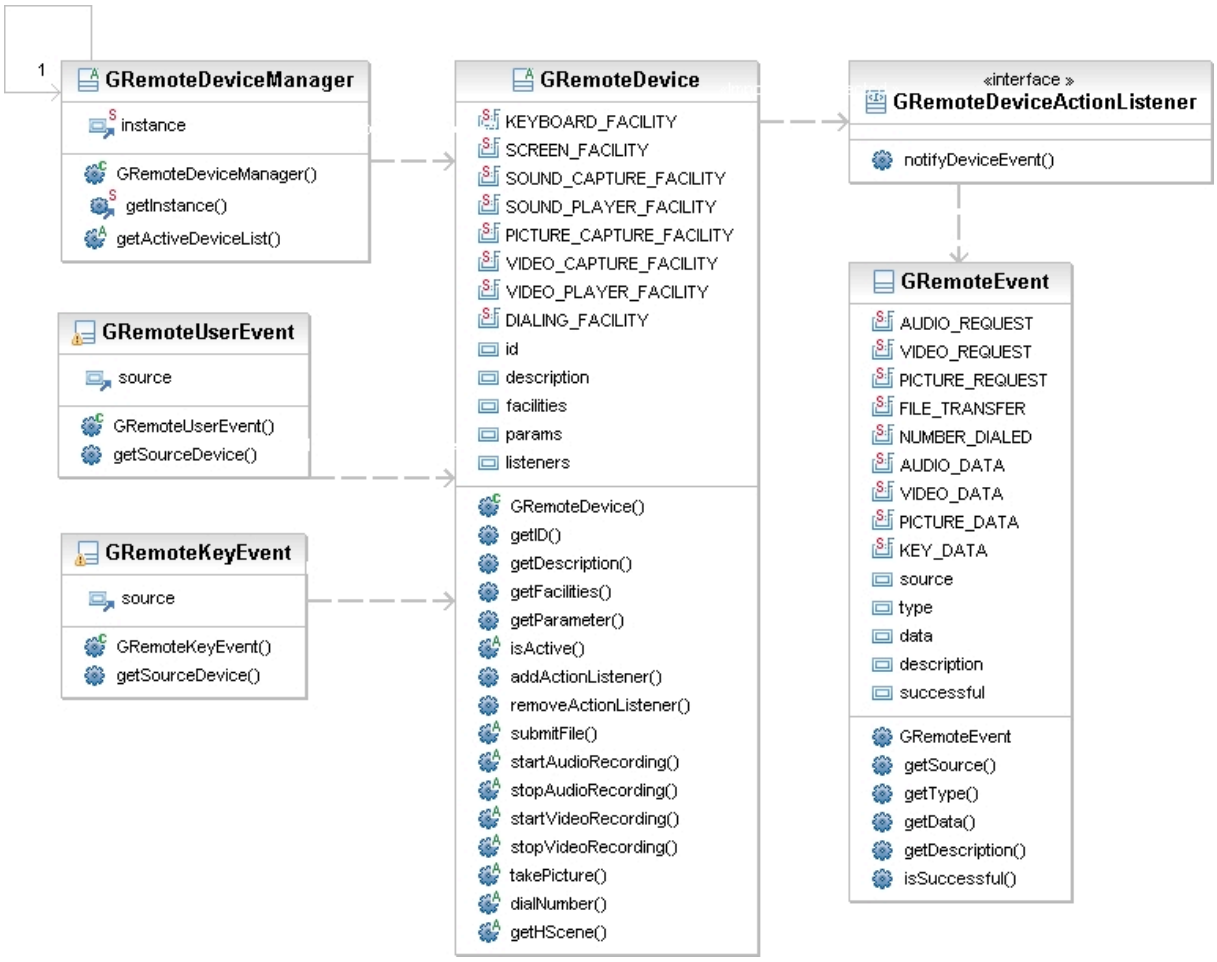

Figura 12: Diagrama de Classes UML da API de Integração de Dispositivos.

A classe GRemoteDeviceManager define um objeto que deve administrar todas as conexões com os dispositivos de interatividade registrados com o Ginga.

Os métodos públicos disponíveis são:

- static GRemoteDeviceManager getInstance () : este método retorna um objeto GremoteDeviceManager;

- GRemoteDevice[] getActiveDeviceList() : este método retorna um *array* contendo objetos GRemoteDevice referenciando cada um dos dispositivos de interatividade registrados com o Ginga.

A classe GRemoteDevice define um objeto que deve ser uma representação abstrata de um dispositivo de interação. Esta classe oferece métodos que possibilitam a recuperação de informações acerca dos dispositivos registrados (tipo do dispositivo, funcionalidades disponíveis etc), bem como explorar as funcionalidades disponíveis do mesmo (utilizar recursos de gravação de áudio, vídeo, captura de imagens etc).

As constantes públicas estáticas presentes na classe GRemoteDevice são detalhadas na Tabela 1.

Constantes	**Descrição**
int KEYBOARD_FACILITY	Um valor inteiro que representa a funcionalidade de teclado de um dispositivo de interação
int SCREEN_FACILITY	Um valor inteiro que representa a funcionalidade de tela de um dispositivo de interação.
int SOUND_CAPTURE_FACILITY	Um valor inteiro que representa a funcionalidade de captura de áudio de um dispositivo de interação.
int SOUND_PLAYER_FACILITY	Um valor inteiro que representa a funcionalidade de reprodução de áudio de um dispositivo de interação.
int PICTURE_CAPTURE_FACILITY	Um valor inteiro que representa a funcionalidade de captura de imagens de um dispositivo de interação.
int VIDEO_CAPTURE_FACILITY	Um valor inteiro que representa a funcionalidade de captura de vídeo de um dispositivo de interação.
int VIDEO_PLAYER_FACILITY	Um valor inteiro que representa a funcionalidade de reprodução de vídeo de um dispositivo de interação.
int DIALING_FACILITY	Um valor inteiro que representa a funcionalidade de efetuar ligações telefônicas de um dispositivo de interação.

Tabela 1: Constantes públicas estáticas da classe GRemoteDevice

Os métodos públicos da classe GRemoteDevice são:

- int getID() : este método retorna um inteiro representando o identificador único de um dispositivo de interação.

- String getDescription() : este método retorna uma string (cadeia de caracteres) contendo a descrição do dispositivo de interação.
- int[] getFacilities() : este método retorna um vetor de inteiros onde cada item representa uma funcionalidade suportada pelo dispositivo, de acordo com a Tabela 1.
- String getParameter(String name) : Este método recebe uma string contendo um mnemônico para um parâmetro (característica) relacionada a um dispositivo de interação (por exemplo, a área da tela em pixels representado pelo mnemônico "screen_area") e retorna o valor correspondente em uma String (por exemplo "640x480"). Estes parâmetros (e seus respectivos mnemônicos) dependem do dispositivo de interação em questão.
- boolean isActive() : Este método retorna um valor booleano (verdadeiro ou falso) representando o estado do dispositivo: verdadeiro (true) se o dispositivo estiver ativo, falso (false) se o dispositivo estiver inativo ou não registrado com o Ginga.
- void addActionListener(GRemoteDeviceActionListener lis) : este método registra no dispositivo um *Listener* do tipo GRemoteDeviceActionListener (passado como parâmetro), que será notificado das atividades deste dispositivo.
- void removeActionListener(HRemoteDeviceActionListener lis) : este método desregistra no dispositivo um *Listener* do tipo GRemoteDeviceActionListener (passado como parâmetro).
- int submitFile(java.io.File file) throws java.io.IOException : este método envia um arquivo (passado como parâmetro) para o dispositivo. O método retornará um valor inteiro igual ao tamanho do arquivo em bytes em caso de sucesso ou -1 (menos um) em caso de falha. Este método pode lançar uma exceção do tipo java.io.IOException.
- void startAudioRecording() throws IOException : este método inicia a captura de áudio no dispositivo. Este médoto pode lançar uma exceção do tipo java.io.IOException.
- void stopAudioRecording() throws IOException : este método finaliza a captura de áudio no dispositivo. Este método pode lançar uma exceção do tipo java.io.IOException.

- void startVideoRecording() throws IOException : este método inicia a captura de vídeo no dispositivo. Este método pode lançar uma exceção do tipo java.io.IOException.
- void stopVideoRecording() throws IOException : este método finaliza a captura de vídeo no dispositivo. Este método pode lançar uma exceção do tipo java.io.IOException.
- int takePicture() throws java.io.IOException : este método dispara a captura de uma fotografia no dispositivo. Este método pode lançar uma exceção do tipo java.io.IOException.
- void dialNumber(String number) : este método faz com que o dispositivo efetue uma ligação telefônica para o número passado como parâmetro em uma cadeia de caracteres (string). Este método pode lançar uma exceção do tipo java.io.IOException.
- org.havi.HScene getHScene(): este método retorna um objeto do tipo org.havi.HScene, relativo ao dispositivo, de forma que elementos de interface possam ser manipulados no objeto HScene de cada dispositivo de interação.

A interface GRemoteDeviceActionListener deve conter métodos que devem ser implementados por qualquer objeto que deva ser notificado sobre eventos relacionados às atividades dos dispositivos de interação registrados com o Ginga.

O método público que deverá ser implementado é o void notifyDeviceEvent(GRemoteEvent event). Este método notifica o objeto que implementa a interface GRemoteDeviceActionListener acerca de eventos que ocorreram nos dispositivos, através da passagem de um objeto GRemoteEvent como parâmetro.

A classe GRemoteEvent descreve um objeto que representa um evento relacionado a um dispositivo de interação registrado com o Ginga. Este objeto encapsula dados relacionados ao evento.

As constantes públicas estáticas presentes na classe GRemoteEvent são detalhados na Tabela 2.

Constante	Descrição
int AUDIO_REQUEST	Um valor inteiro que define que o evento está relacionado a uma requisição por áudio e contém o estado dessa requisição encapsulado.
int VIDEO_REQUEST	Um valor inteiro que define que o evento está relacionado a uma requisição por vídeo e contém o estado dessa requisição encapsulado.
int PICTURE_REQUEST	Um valor inteiro que define que o evento está relacionado a uma requisição por imagem e contém o estado dessa requisição encapsulado.
int FILE_TRANSFER	Um valor inteiro que define que o evento está relacionado a uma requisição por transferência e contém o estado dessa requisição encapsulado.
int NUMBER_DIALED	Um valor inteiro que define que o evento está relacionado ao estabelecimento de uma ligação telefônica (por parte do dispositivo de interação) e contém o estado dessa requisição encapsulado.
int AUDIO_DATA	Um valor inteiro que define que o evento possui dados de áudio encapsulados.
int VIDEO_DATA	Um valor inteiro que define que o evento possui dados de vídeo encapsulados.
int PICTURE_DATA	Um valor inteiro que define que o evento possui uma imagem encapsulada.
int KEY_DATA	Um valor inteiro que define que o evento possui um evento de teclas encapsulado.

Tabela 2: Constantes públicas estáticas da classe GRemoteEvent

Os métodos públicos da classe GRemoteEvent são:

- Object getSource() : este método retorna um Object referenciando o dispositivo de interação (GRemoteDevice) que foi a fonte do evento em questão.
- int getType() : este método retorna um valor inteiro de acordo com o tipo do evento em questão (de acordo com as constantes estáticas definidas na mesma classe).
- byte[] getData() : este método retorna os dados relacionados ao evento em questão.
- String getDescription() : este método retorna uma cadeia de caracteres (String) relacionada a uma descrição do evento em questão.

- boolean isSuccessful() : este método retorna um valor booleano (verdadeiro ou falso) representando o estado do evento: se verdadeiro (*true*) o evento foi bem sucedido, falso (*false*) em caso contrário.

A classe GRemoteKeyEvent estende java.awt.event.KeyEvent, e é relacionada a eventos de teclas originados em dispositivos de interatividade. Sua utilização é feita da mesma maneira como nos eventos AWT normais, bastando fazer um *downcasting* quando necessário. O método público da classe GRemoteKeyEvent é o GRemoteDevice getSourceDevice() que retorna um objeto GRemoteDevice referenciando o dispositivo de interação que foi a fonte do evento em questão.

A classe GRemoteUserEvent estende org.dvb.event.UserEvent, e é relacionada a eventos do usuário originados em dispositivos de interatividade. O método público da classe GRemoteUserEvent é o GRemoteDevice getSourceDevice() que retorna um objeto GRemoteDevice referenciando o dispositivo de interação que foi a fonte do evento em questão.

4.3 IMPLEMENTAÇÃO DE REFERÊNCIA

Durante o projeto FlexTV (29) foi desenvolvida uma implementação de referência do *middleware* Ginga e da API descrita anteriormente. Tal implementação foi realizada inicialmente através de uma extensão ao XletView[6] (emulador de um ambiente de TVDI) e posteriormente incorporado a implementação de referência do Ginga desenvolvida pelo LAVID.

O XletView possui uma implementação reduzida do Havi, o que foi bastante útil tanto para a análise quanto para a implementação da API especificada nesse trabalho. O componente gráfico principal da API, a classe GRemoteHScene é representado como uma extensão do HScene do Havi; uma vez que já tínhamos a implementação Havi de referência, a implementação se tornou mais simples.

No XletView temos ainda disponível um ambiente completo de testes, ainda que em *software*. Sua interface, conforme mostrado na Figura 13, possui um controle remoto para

[6] XletView: http://xletview.sourceforge.net/

interação com o usuário, cujos cliques são traduzidos para eventos idênticos aos gerados pelo clique do controle remoto no STB.

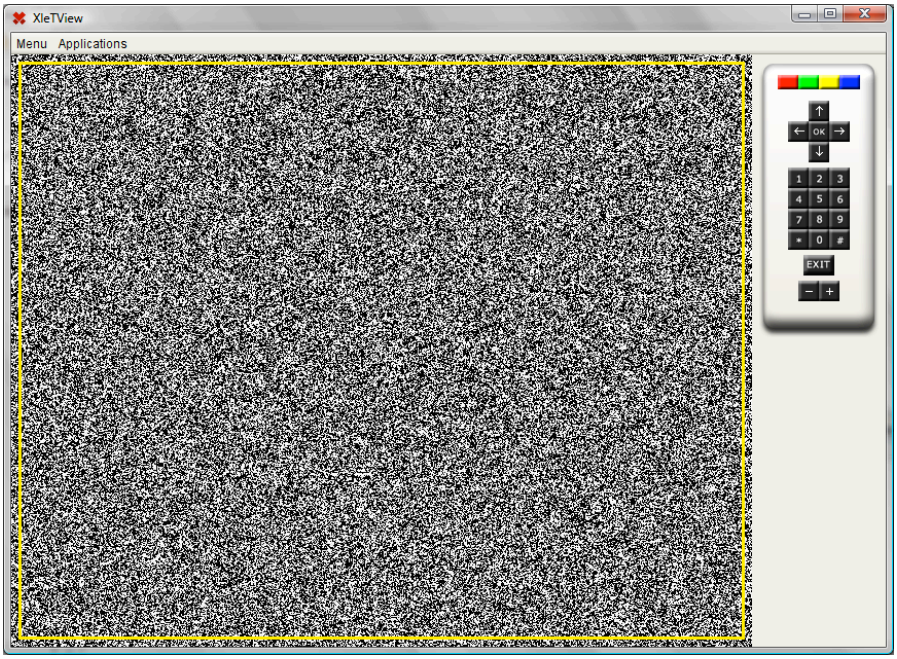

Figura 13: Interface do XletView

Foi adicionado a implementação do XletView, que tem código aberto, o código da API de Integração de Dispositivos proposta pelo trabalho. Uma vez que a API representa apenas uma interface de programação aos desenvolvedores, código adicional foi adicionado para implementar as funcionalidades da API acima a camada de *middleware*. Esse código adicional trata, por exemplo, da conexão dos dispositivos e da geração dos eventos do usuário, por meio das classes GRemoteKeyEvent e GRemoteUserEvent.

Uma vez que tínhamos o ambiente necessário completo: o *middleware* para teste (no caso o emulador XletView) e .a implementação da API, foi iniciado o desenvolvimento e testes das aplicações piloto que são apresentadas, posteriormente, nesse trabalho. Uma vez que elas utilizam apenas a API, a elas não importam se estão sendo executadas no XletView ou na implementação do Ginga no STB; valendo mencionar que a transferência para o STB é direta e transparente.

Por ser uma implementação de teste, nem todos os recursos previstos na especificação foram implementados. Por exemplo, a implementação só permite a conexão por dispositivos móveis via TCP, sendo possível a utilização por meio de conexão *Wi-Fi* (quando disponível nos dispositivos), ou mesmo GPRS.

Para emulação dos dispositivos, uma aplicação Java foi desenvolvida, implementando o protocolo usado para comunicação com a API. Esse dispositivo emulado possuía teclado e capacidade para reprodução de imagens.

Para que pudéssemos validar tanto a API quanto as aplicações, um protocolo de comunicação entre os dispositivos e o receptor precisou ser especificado. Esse protocolo não faz parte da especificação da API, que é o principal objetivo desse trabalho; a API é capaz de funcionar com qualquer protocolo que venha a ser utilizado, sendo a escolha do protocolo devida às implementações da API que venham a ser feitas. O protocolo que utilizamos é simples e baseado em mensagens de texto. O protocolo pode ser brevemente descrito como:

Formato geral das mensagens:

comando <parâmetro 1> ... <parâmetro n>'\n'

Onde 'comando' representa o objetivo principal da mensagem, e os 'parâmetros' podem variar em número; cada comando tem uma quantidade de parâmetros específica. Toda mensagem termina com uma quebra de linha ('\n'), e cada parte da mensagem é separada por espaço.

Para conectar-se, o *handshake* da conexão é iniciado pelo dispositivo, como se segue:

1: [dispositivo] **ID <identificação única do dispositivo>**

* A indentificacao única do dispositivo é atribuída pelo receptor na primeira conexão, e deve ser armazenada no dispositivo para futuras conexões. Na primeira conexão o dispositivo envia o identificador '?'

2: [dispositivo] **DESCRIPTION <descrição textual do dispositivo >**

3: [dispositivo] **FACILITIES <capacidade 1> <capacidade n>**

* Cada capacidade do dispositivo é representada por uma palavra, como: **SCREEN**, **KEYBOARD** ou **SOUND_CAPTURE**, por exemplo.

4: [receptor] **CONNECTED <identificação única do dispositivo>**

Algumas mensagens de comunicação utilizam o primeiro parâmetro para informar o tamanho em *bytes* do parâmetro seguinte. Por exemplo, para informar ao receptor sobre uma tecla pressionada o dispositivo envia:

[dispositivo] **KEY 1 <byte representando a tecla>**

Para enviar uma imagem que deve ser exibida no dispositivo, o receptor envia:

[receptor] **IMG <tamanho em bytes> <bytes da imagem>**

Para iniciar e parar a captura de áudio pelo dispositivo, o receptor envia:

[receptor] **START AUD OUT**

[receptor] **STOP AUD OUT**

No entanto, para uma melhor validação e melhor qualidade da prova de conceito, era necessário implementar o componente do Ginga num receptor móvel real. Escolhemos utilizar celulares Nokia disponíveis, mais especificamente os modelos N80 (30) e o E61 (31).

Inicialmente foi implementado uma versão utilizando a tecnologia J2ME, que suporta conexões *Bluetooth* ou *Wi-Fi*, de acordo com os recursos disponíveis no celular. No entanto, ao utilizar essa tecnologia Java, os recursos disponíveis para o desenvolvedor, em termos de utilização das funções do aparelho, são muito limitados. Utilizando a versão corrente do J2ME só é possível, por exemplo, exibir imagens, capturar as teclas digitadas dos usuários, e tocar clipes de mídia previamente armazenados. Não é permitido o *streaming* de mídia; o que inviabiliza a captura de áudio e envio simultâneo para o receptor.

Para sobrepor as limitações encontradas com o J2ME, partimos para o estudo da plataforma *Symbian*, presente nos celulares Nokia disponíveis. O Symbian oferece um controle muito maior dos recursos disponíveis no aparelho; a plataforma oferece uma interface de programação entendida do C++, e oferece o controle quase que total de todos os dispositivos internos do aparelho. Com o Symbian foi possível capturar o áudio bruto do microfone do celular e enviar para o receptor; também é possível a captura de imagens, vídeo e realizar chamadas telefônicas por aplicações *Symbian*.

Atualmente a API também está sendo portada e disponibilizada no projeto OpenGinga[7].

[7] OpenGinga: http://www.openginga.org/

5 ESTUDO DE CASOS

Com a API Ginga-J ora proposta, tornou-se possível o desenvolvimento de aplicações interativas de TV Digital executadas em mais de um dispositivo e com suporte para múltiplos usuários interagindo simultaneamente.

Tais aplicações se destacam por apresentarem maior grau de interatividade, proporcionando ao usuário de TVDI uma experiência mais rica, incluindo o uso de recursos multimídia, quando disponíveis nos dispositivos móveis.

Nesse capítulo abordamos três cenários diferentes, cada um deles explorando uma funcionalidade nova adicionada pelo uso da API Ginga-J proposta. Cada um desses cenários é demonstrado através de aplicações piloto, desenvolvidas como parte do trabalho. As aplicações foram desenvolvidas utilizando a API Ginga-J com as modificações necessárias para integração de dispositivos móveis, e são executadas numa implementação de referência do *middleware* desenvolvida pelo LAVID no contexto do esforço do SBTVD - o FlexTV (29).

Nos estudos de casos foram utilizados celulares Nokia com tecnologia *Wi-fi* e plataforma Symbian. A parte móvel do Ginga executando em cada celular foi desenvolvida em Symbian como parte do trabalho, e serviu para a validação dos resultados da API proposta. Uma implementação para o iPhone, utilizando o SDK disponível e a linguagem Objective-C está em andamento.

O detalhamento das aplicações piloto desenvolvidas não pertencem ao escopo do trabalho. No entanto, para descrever de forma mais didática o contexto dessas aplicações utilizamos diagramas de casos de uso UML (34). É importante salientar que os diagramas apresentados não representam as aplicações em sua plenitude, uma vez que procuramos enfatizar as funcionalidades oferecidas que utilizam a API proposta.

5.1 CENÁRIO 1: USO DE DISPOSITIVOS MÓVEIS COMO CONTROLE-REMOTO

A funcionalidade básica e direta conseguida com a API proposta é a utilização dos dispositivos móveis como meio de interagir com o receptor de TV Digital. Ao conectar o dispositivo ao receptor, cada tecla pressionada é interpretada como uma tecla de controle remoto.

Devido a forma como foi especificada essa parte da API, apenas com a adição de novos métodos a classes estendidas do HAVi, e sem nenhuma modificação ao modo de tratamento de eventos das APIs Havi e DVB User, presentes no núcleo comum J.201 (23), o GEM, é possível garantir que as aplicações já existentes permaneçam compatíveis com essa nova funcionalidade de maneira transparente. Ou seja, basta conectar um dispositivo móvel ao receptor e controlar qualquer aplicação já existente com as teclas desse dispositivo de forma automática, nenhum código da aplicação precisa ser modificado para usar a funcionalidade.

No entanto, apesar da forma transparente com a qual a API de Integração de Dispositivos trata as aplicações já existentes, devemos observar que a API provê métodos para identificação de qual foi o dispositivo móvel que gerou cada interação, caso a aplicação necessite dessa informação.

De acordo com MIRANDA (5) a utilização de dispositivos móveis, especialmente celulares, como interfaces de acesso as aplicações em TV Digital é uma tendência para substituição do atual controle remoto. Na Figura 14 podemos observar um comparativo dessas duas interfaces de acesso às aplicações.

(a) (b) (c)

Figura 14: Utilização do Controle-remoto de TVDI e do Celular como interface de acesso.

5.1.1 TOV e HOMEBANKING

Como já foi dito, qualquer aplicação pode lançar mão dessa funcionalidade, seja de forma transparente, seja verificando de qual dispositivo as interações são originadas.

No intuito de ilustrar esse primeiro cenário apresentamos duas aplicações-piloto desenvolvidas no LAVID: a TOV-Torcida Virtual (35) e o Homebanking para TV Digital[8].

.A aplicação TOV é uma aplicação que recebe comandos de controle remoto de forma transparente, pois ela não diferencia, em sua implementação corrente, qual dispositivo originou o chamado da interação. Sendo assim, a interação com usuário é feita da mesma forma como qualquer implementação utilizando o Havi. Para tanto, basta implementar o seguinte método:

keyPressed(KeyEvent < tecla>) da Interface KeyListener

A aplicação Homebanking, entretanto, precisa diferenciar qual dispositivo está gerando quais interações, pois a aplicação pode gerenciar mais de uma sessão de Homebanking por vez. Uma sessão é associada a cada dispositivo móvel conectado. Por questões de segurança e de confidencialidade de dados, é um requisito natural da aplicação perceber qual dispositivo gerou o comando para, por exemplo, exibir o saldo em conta, ou executar um comando para transferência de valores.

Para identificar qual dispositivo interagiu com a aplicação, apenas algumas linhas de código precisam ser adicionados no método *keyPressed(KeyEvent <tecla>)*. Essas linhas de código tem o objetivo de verificar se o objeto KeyEvent é, na verdade, uma implementação da extensão GRemoteKeyEvent proposta; caso positivo, a aplicação saberá que a interação foi originada por um dispositivo móvel, e terá na classe GRemoteKeyEvent um método que identifica o dispositivo. Caso a implementação seja um KeyEvent original, a interação foi originada pelo controle remoto comum.

Como pode ser observado no trecho de código ilustrado na Figura 15, é muito simples identificar qual dispositivo interagiu com a aplicação. Após a identificação, a aplicação pode continuar com o tratamento especifico ao qual se propõe.

[8] A apresentação e descrição mais completa dessas aplicações é feita nos próximos cenários, pois cada uma utiliza outros recursos mais avançados da API proposta e especificamente relacionados ao propósito ao qual elas foram concebidas

```
if ((tecla instanceof HRemoteKeyEvent)){
      HRemoteKeyEvent teclaDisp = (HRemoteKeyEvent) tecla;
      int idDispositivo = teclaDisp.getOriginDevice().getID();
}
```

Figura 15: Trecho de código para identificação do dispositivo onde uma tecla foi pressionada.

Ao especificar essa API para o Ginga-J, uma preocupação sempre presente foi a de garantir simplicidade para o desenvolvedor. Pelo trecho de código mostrado na Figura 15, fica ilustrado que o código para utilização desse novo recurso é muito simples e fácil de usar. Apesar de simples, a identificação de qual dispositivo interage com a aplicação oferece ao desenvolvedor a capacidade de desenvolver novos tipos de aplicações.

5.2 CENÁRIO 2: USO DE RECURSOS MULTIMÍDIA DOS DISPOSITIVOS MÓVEIS

Outra contribuição do trabalho foi possibilitar que as aplicações utilizem recursos multimídia presentes nos dispositivos móveis. Durante a fase de especificação da API Ginga-J, percebeu-se que seria possível tratar um dado multimídia de forma semelhante a uma tecla capturada do controle remoto. A API provê meios para que o desenvolvedor identifique os tipos dos dados capturados nos dispositivos integrados ao Ginga.

Nesse cenário – de recursos multimídia e dispositivos móveis – podemos destacar alguns recursos existentes em dispositivos móveis, que podem ser utilizados no contexto das aplicações interativas, são eles:

- Captura de áudio: celulares e PDAs possuem microfones e capacidade de capturar áudio.
- Tela: celulares e PDAs possuem telas, essas telas podem ser aproveitadas pelas aplicações para exibição de conteúdo adicional.
- Alto-falantes: também presentes em celulares.
- Câmera: muito comum em celulares hoje em dia, pode capturar imagens ou vídeos a serem utilizados por aplicações interativas.
- Rede Telefônica: uma aplicação pode gerar, por meio de um celular, uma ligação para um número de *Call Center*, ou para um provedor de Internet.

Outros recursos não muito comuns ainda, mas podem ser utilizados dados que identifiquem localização e movimentação. No caso da localização muitos dispositivos já estão sendo equipados com GPS, o que pode ser útil para oferecer facilidades que utilizam esse tipo de informação como localizadores e guias automáticos pela tela da TV. A movimentação é uma característica proveniente dos acelerômetros que pode ser utilizada para disponibilizar controles de acesso facilitados para TVD, como apontadores de tela (implementando uma espécie de *mouse* para TV Digital). Esse recurso pode ser utilizadao também no desenvolvimento de jogos.

A API proposta possui métodos para identificar que recursos cada dispositivo móvel possui. Essa identificação é possível coletando a lista de *Facilities* de um dispositivo especifico.

public int[] getFacilities() disponível na classe GRemoteDevice

Cada *Facility* é representada por um inteiro, e tem sua correspondência entre as várias constantes definidas na própria classe GRemoteDevice. A classe também possui os métodos para iniciar o uso de cada recurso multimídia.

Algumas aplicações foram desenvolvidas para testar a API proposta. As seções seguintes apresentam algumas delas.

5.2.1 TOV

A Torcida Virtual, ou simplesmente TOV, é uma aplicação multimídia. O conceito da TOV é preencher um estádio de esportes virtualmente. Uma vez preenchido o estádio, as pessoas participantes poderão interagir entre si como se estivessem num estádio real. A idéia é dar suporte a um ambiente virtual com espaço acústico compartilhado.

As principais funcionalidades da TOV são apresentadas no diagrama de casos de uso UML da Figura 16. Nele podemos observar que cada participante da TOV deve escolher um setor do estádio virtual para sentar, bem como um assento numerado. A localização do usuário é importante para limitar com quais outros usuários a interação por meio do áudio se dará. Na especificação da TOV foi definido que o áudio será ouvido pelos usuários sentados

próximos, virtualmente, implementando uma espécie de cone acústico. A Figura 17 apresenta a tela da TOV para execução dessas duas funcionalidades.

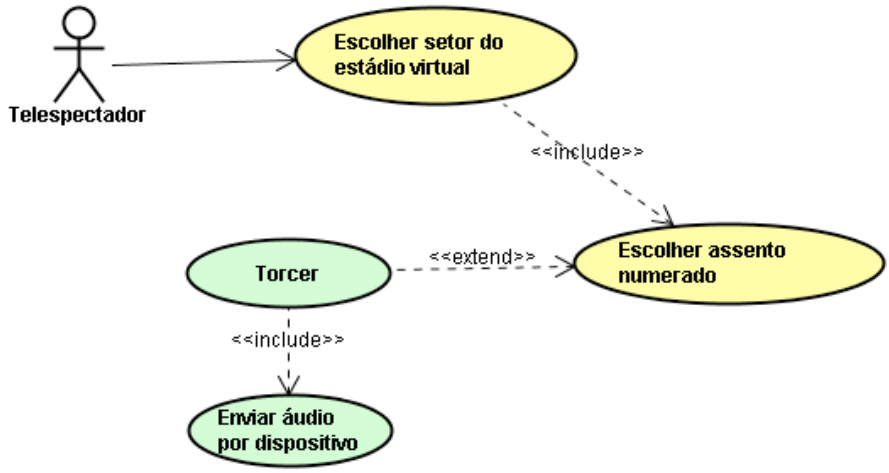

Figura 16: Diagrama de Casos de Uso UML da TOV.

Figura 17: Telas da TOV.

O caso de uso “Torcer” refere-se a característica mais inovadora e ponto central deste aplicativo: a possibilidade de torcer em um estádio virtual. Em outras palavras, o telespectador uma vez conectado pode gerar áudio que será mixado e retornado para ele e seus vizinhos do estádio. Para que essa funcionalidade se torne factível, é necessário ter algum

mecanismo de captura do áudio do ambiente onde o receptor de TVD está inserido (caso de uso “Enviar áudio por dispositivo”). Além disso, é necessário um conjunto de servidores de mixagem de áudio de retaguarda, responsáveis pela captura e mixagem do áudio e retorno aos usuários participantes. Esses servidores de retaguarda estão fora do escopo desse trabalho.

A API proposta torna possível capturar o áudio, por exemplo, de um celular conectado ao receptor. Na versão da TOV com suporte a dispositivos móveis, o áudio é capturado de todos dispositivos móveis com capacidade de captura de áudio e reproduzido localmente. O procedimento para iniciar e parar a captura do áudio, bem como de coletar o áudio e comandar sua reprodução é ilustrado na Figura 18.

```
HRemoteDevice dev = aDevice;
int action = aAction; //se action = 1, então iniciar audio.
if (action == 1){
      dev.startAudioRecording();
      dev.addActionListener(this); //adiciona ouvinte, que será notificado
da chegada dos pacotes de áudio
}else {
      dev.stopAudioRecording();
      dev.removeActionListener(this);
}
```

Figura 18: Trecho de código para iniciar e parar a captura de áudio em um dispositivo.

Como pode ser observado no trecho de código da Figura 18, além de iniciar a captura do áudio, a aplicação precisa adicionar um ouvinte para os eventos do dispositivo. Dessa forma, a cada pacote de áudio gerado na fase de captura do dispositivo, a aplicação será notificada e receberá tais pacotes para que possam ser utilizados. A notificação dos eventos ocorrerá por meio do método *void notifyDeviceEvent(GRemoteEvent <event>)*, da Interface GRemoteDeviceActionListener. Na Figura 19 pode ser observado o código necessário para identificar o tipo de informação de um evento e capturar os dados dessa informação.

```
if (event.getType() == GRemoteEvent.AUDIO_DATA){
      audioPlayer.addAudioData(event.getData()); //audioPlayer é um objeto
de uma classe que recebe pacotes de áudio e os reproduz.
}
```

Figura 19: Trecho de código para verificar o tipo de informação de um evento de dispositivo e a coleta dessa informação.

No momento então que o usuário escolhe um assento, a TOV inicia a captura do áudio e inicia a transmissão do áudio para o servidor de retaguarda.

5.2.2 HomeBanking

A aplicação do *Homebanking* tem o objetivo de oferecer as mesmas operações bancárias encontradas nos *sites* de bancos na Internet. Com a consolidação da TVDI, uma aplicação de banco na tela da TV pode ser uma opção mais próxima da realidade da grande maioria dos usuários de baixa renda. Na Figura 20 podemos observar o diagrama de casos de uso para esse aplicativo. Nesse diagrama observamos que o telespectador ao se logar na sua aplicação na TV tem a opção de abrir uma sessão para uso do mesmo aplicativo no celular.

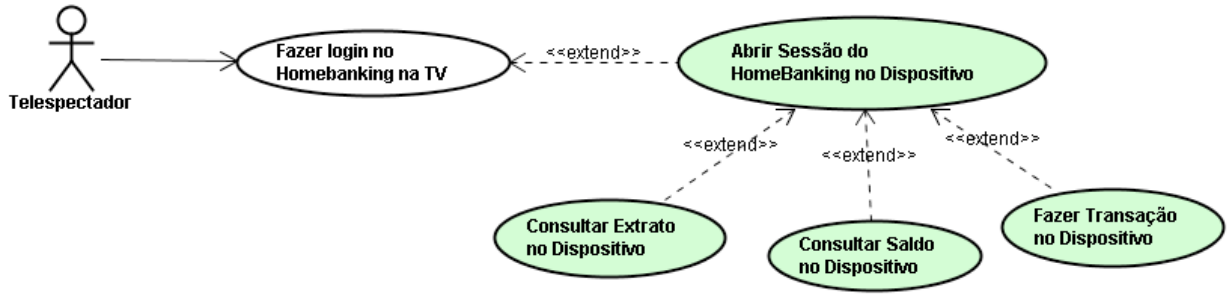

Figura 20: Diagrama de Casos de Uso UML do HomeBanking.

Entretanto, a transferência do banco para a tela da TV deve ser feita de forma cautelosa. Sabemos que a experiência de assistir TV é, na maioria das vezes, coletiva. Dessa forma, acessar sua conta bancária na tela da TV pode, muitas vezes, ser inconveniente, já que outras pessoas podem estar assistindo a TV naquele momento. Desse modo, é essencial manter a questão da privacidade, que só é alcançada se tivermos um único telespectador assistindo à TV num dado momento.

Na aplicação de *HomeBanking* ora apresentada, utilizamos a API proposta nesse trabalho para tornamos a aplicação muito mais factível ao ambiente real. Para essa aplicação, conseguimos garantir a questão de privacidade enviando a interface da aplicação para tela do dispositivo móvel. Ou seja, diferente do ambiente comum de TVDI onde o único recurso de exibição gráfica disponível é a tela da TV, no Ginga-J com a API proposta podemos ter recursos de exibição gráficas extras, relativos as telas dos dispositivos móveis conectados ao receptor de TV.

Além disso, o fato da aplicação identificar o dispositivo que está interagindo com ela é possível processar os comandos recebidos e enviar o resultado do processamento (no caso, a

saída de alguma operação bancária, como o saldo em conta corrente) diretamente para a tela do dispositivo em questão.

Buscando a facilidade para o desenvolvedor, mais uma vez, a forma de escrever numa tela de dispositivo é análoga a forma de escrever na tela TV. Para desenhar na tela usamos a mesma classe HScene já utilizada no Havi, entretanto estendemos essa classe para a GRemoteHScene, classe que contém funcionalidades específicas para o funcionamento remoto.

Cada dispositivo móvel conectado, desde que tenha a *Facility* de *Screen* disponível, indicando que possui um *display* presente, possuiu seu HScene relacionado, com suas propriedades gráficas correspondentes. O desenvolvedor pode então adicionar os componentes Havi a esse HScene, da mesma maneira como faz no HScene principal, que é mostrado na tela da TV.

O trecho de código ilustrado na Figura 21 mostra como referenciar o HScene de um dispositivo, e exibir o menu de opções da aplicação *homebanking*.

```
HScene cena = dev.getHScene(); //dev é um objeto do tipo GRemoteDevice
cena.removeAll();

cena.add(new HStaticText("Saldo em Conta", 1, 1, 100, 50, new
Font("Tiresias", Font.BOLD, 11), Color.yellow, Color.black, new
HDefaultTextLayoutManager()); //adiciona entrada de menu na tela
cena.add(new HStaticText("Extrato da Conta", 1, 1, 100, 50, new
Font("Tiresias", Font.BOLD, 11), Color.yellow, Color.black, new
HDefaultTextLayoutManager()); //adiciona entrada de menu na tela
cena.add(new HStaticText("Transferência", 1, 1, 100, 50, new
Font("Tiresias", Font.BOLD, 11), Color.yellow, Color.black, new
HDefaultTextLayoutManager()); //adiciona entrada de menu na tela

cena.setVisible(true);
cena.requestFocus();
cena.repaint();
```

Figura 21: Trecho de código para referenciar o HScene do dispositivo e adicionar elementos a este.

Com a API Ginga-J proposta por esse trabalho, a aplicação usufruiu de sua características multimídia para tornar a experiência de *homebanking* mais privativa. Esse ganho é certamente um diferencial de mercado para esse tipo de aplicação que se torna mais atrativa e principalmente mais segura e confidencial, como podemos observar na Figura 22.

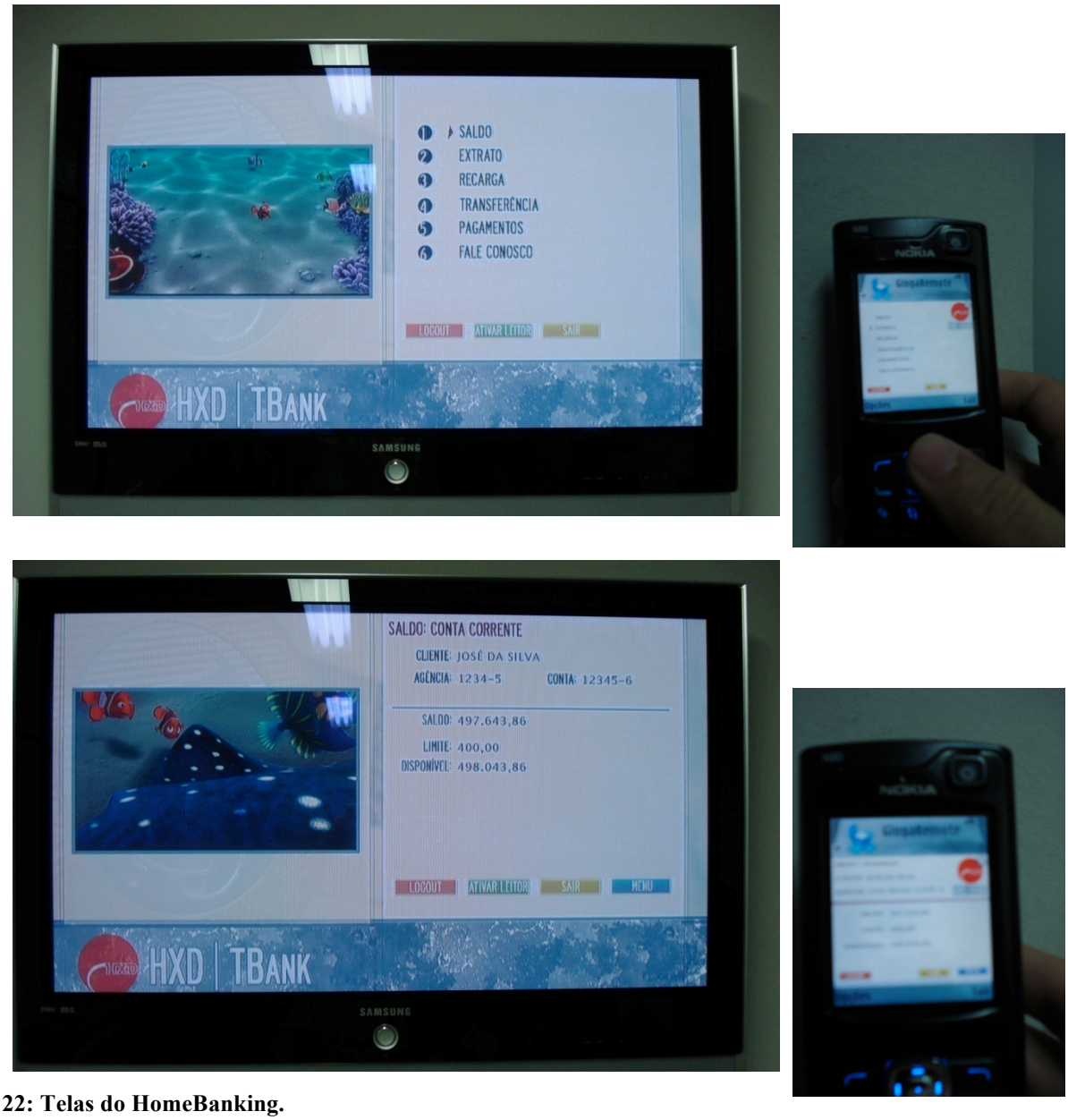

Figura 22: Telas do HomeBanking.

5.2.3 TV Tunes

A aplicação TV Tunes, desenvolvida no LAVID, tem como funcionalidade principal a troca de conteúdo entre a TV e o dispositivo móvel.

Como vimos anteriormente, em um *transport stream*[9] de TV Digital, além do áudio e vídeo dos canais de TV, fluxos de dados podem ser multiplexados paralelamente. Esses dados muitas vezes representam aplicações, e dessa forma podem conter arquivos.

[9] *Transport stream:* O fluxo MPEG-2 TS multiplexado enviado para difusão.

Tomando como base a capacidade de se transmitir arquivos multiplexados num *transport stream*, a aplicação TV Tunes tem o objetivo de transferir alguns desses arquivos para os dispositivos móveis. Dentre esses arquivos, o TV Tunes se empenha na transferência de conteúdo multimídia, como *ringtones*, músicas e vídeos. Adicionando a aplicação funcionalidades de cobrança (que não serão abordados na nossa versão), a aplicação TV Tunes se torna uma maneira de vender conteúdo digital multimídia pela TV Digital. Na Figura 23 podemos observar o diagrama de casos de uso, reduzido, para a aplicação TV Tunes, e na Figura 24 é ilustrada a tela de aplicacao mostrada na TV.

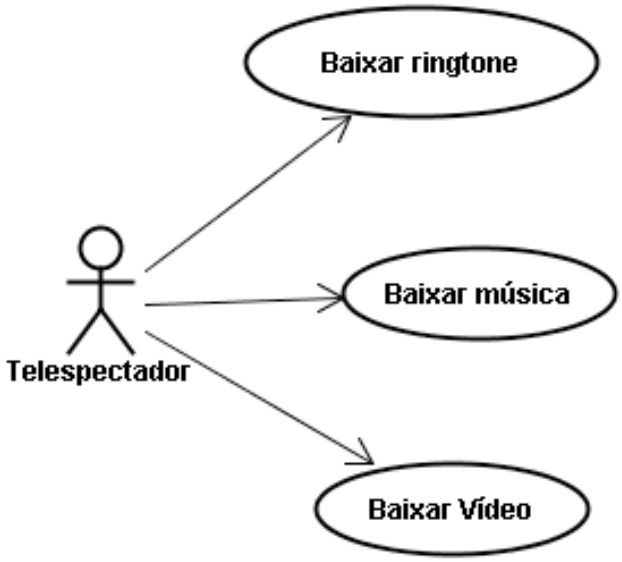

Figura 23: Diagrama de Casos de Uso UML da TV Tunes.

Figura 24: Tela do TV Tunes.

A transferência do conteúdo se dá pela mesma conexão do dispositivo pelo receptor, e não utilizando algum canal de comunicação como acessório (como a rede do celular).

Tal aplicação se tornou possível com a API proposta no presente trabalho. Mais especificamente, a classe GRemoteDevice possui o método *submitFile(java.io.File <file>)* que transfere um arquivo do sistema de arquivos da aplicação para o dispositivo, que coleta o arquivo e faz o processamento de acordo com seu tipo. A Figura 25 ilustra a utilização desse método.

```
dev.submitFile(mp3File); //dev é um objeto do tipo GRemoteDevice
```

Figura 25: Trecho de código para transferir um arquivo para o dispositivo.

Além da transferência de arquivos multimídia, a aplicação pode se utilizar da API para fazer o *preview* do conteúdo diretamente no dispositivo, sem a necessidade de transferir o arquivo em si. Essa funcionalidade pode ser implementada ao utilizar outros recursos multimídia do dispositivo, como alto-falantes e tela.

5.3 CENÁRIO 3: USO DE RECURSOS MULTIUSUÁRIO

Por fim, abordamos um último cenário onde testamos outra importante contribuição do trabalho proposto, que é a capacidade de desenvolver aplicações interativas multiusuário.

A API proposta permite a conexão simultânea de vários dispositivos móveis ao receptor de TVD. Adicionando a essa capacidade a possibilidade das aplicações identificarem individualmente a interação de cada dispositivo, é possível fazer com que a aplicação reaja diferentemente para cada um destes, criando as condições necessárias para implementação de uma aplicação multiusuário.

Essa funcionalidade está intimamente relacionada com a que foi apresentada no Cenário 1. Ao prover meios de identificar as teclas pressionadas em dispositivos como teclas de controle remoto e, adicionalmente, identificar qual dispositivo originou cada interação, além de permitir a conexão de múltiplos dispositivos simultaneamente; essa API Ginga-J ofereces as ferramentas necessárias para a criação de aplicações interativas multiusuário.

O que vai determinar, então, se uma aplicação é ou não multiusuário é a forma como o desenvolvedor a concebeu.

As aplicações de teste que ilustram esse cenário serão apresentadas nas subseções seguintes.

5.3.1 TOV

A TOV pode ser vista também como uma aplicação multiusuário. No entanto, atentemos para o fato de que a questão multiusuário que estamos tratando não é o fato de que muitos usuários podem torcer ao mesmo tempo, essa característica é conseguida apenas pelo uso dos servidores de *back-end*.

A questão multiusuário conseguida com o uso da API é local e se relacionada com a capacidade de conectar mais de um dispositivo móvel. E cada um desses dispositivos ser capaz de enviar o seu áudio para o receptor de TV simultaneamente.

O código do TOV foi concebido de forma que, quando ativar a função torcer, a aplicação procura todos os dispositivos conectados ao receptor de TV que possuem capacidade de capturar áudio. A aplicação requisita então a captura do áudio em cada um desses dispositivos, e os transmite para o servidor de mixagem.

Sendo assim, a TOV exemplifica, também, uma aplicação multiusuário desenvolvida com o uso da API Ginga-J proposta pelo trabalho.

5.3.2 Homebanking

O *Homebanking* é outra aplicação que, além dos recursos multimídia, também possue características de aplicação multiusuário.

No caso do *Homebanking*, a aplicação usa a identificação de qual dispositivo originou a interação para controlar uma sessão de *homebanking* individual para cada dispositivo conectado ao receptor de TV.

Identificando então a interação de cada dispositivo a aplicação pode gerenciar sessões paralelas e independentes de forma privativa. Essa funcionalidade é muito interessante justamente pelo fato de assistir TV ser uma experiência coletiva; tendo então cada usuário controlando o seu dispositivo para acessar a aplicação, ganhamos sessões privadas e a navegação pela aplicação não interferirá na experiência de assistir TV dos demais presentes. A Figura 26 ilustra como deve ser o código para identificar a interação de um dispositivo e reagir de forma específica para cada um destes.

```
if ((tecla instanceof HRemoteKeyEvent)){
      HRemoteKeyEvent teclaDisp = (HRemoteKeyEvent) tecla;
      int idDispositivo = teclaDisp.getOriginDevice().getID();
      switch (idDispositivo){
            case disp1:
                  sessaoDisp[0].newAction(tecla);
                  break;
            case disp2:
                  sessaoDisp[1].newAction(tecla);
                  break;
            case disp3:
                  sessaoDisp[2].newAction(tecla);
                  break;
      }
}
```

Figura 26: Trecho de código exemplificando a identificação do dispositivo onde uma tecla foi pressionada e tratamento de sessões independentes em dispositivos distintos.

6 CONCLUSÕES

Um dos grandes fatores limitantes para o desenvolvimento de aplicações para televisão interativa é o uso do controle remoto como o principal meio de interação. O uso exclusivo deste recurso como dispositivo de entrada limita a interatividade e compromete a usabilidade na TVDI. Para que as aplicações se tornem mais atraentes, alternativas ao controle remoto devem ser analisadas. O uso de dispositivos móveis, como os telefones celulares, é levantado como uma alternativa, uma vez que estes dispositivos são bastante utilizados (7) (5).

Considerando esse contexto, o trabalho aqui discutido apresenta uma solução que possibilita a interação com a TV por meio de dispositivos móveis. Neste trabalho descrevemos a estratégia usada para incorporar essa característica ao Ginga - middleware adotado no Sistema Brasileiro de Televisão Digital. Para tanto, foi definida uma API de Integração de disposistivos móveis. Esta API foi testada através do desenvolvimento de uma implementação de referência e de aplicações explorando os diferentes cenários de uso previstos para a API.

Uma das contribuições da API de Integração de Dispositivos é a possibilidade criação de aplicações inovadoras para o atual cenário de aplicações para TVDI. Esse fato se deve as novas características disponibilizadas pela API:

- suporte a execução de aplicações interagindo com mais de um usuário simultanemente em um único receptor de TV;
- suporte a utilização dos recursos multimídia disponíveis nos dispositivos móveis;
- suporte a criação de estratégias de personalização, que viabilizem a definição e uso de perfis de usuários.

Pretende-se dar prosseguimento a investigação dessas pontencialidades, com a criação de novas aplicações explorando características da API. Dentre as quais sugerimos o desenvolvimento das seguintes aplicações como propostas de trabalhos futuros :

- uma aplicação de controle de grade programação através do reconhecimento do perfil do usuário;

- jogos educativos multiusuário utilizando os dispositivos móveis em apenas um receptor, o que poderia ser útil em escolas, por exemplo;
- aplicações de produção e distribuição de conteúdo multimídia em redes domésticas, tendo o receptor como ponto central.

Além do cenário de aplicações, este trabalho abre novas possibilidades em termos de expansão da API, tanto em funcionalidades, quanto em abrangência de dispositivos móveis compatíveis, onde destacamos:

- a implementação do componente móvel do Ginga compatível com novos dispositivos móveis, como os PDAs e os controles remotos avançados já existentes no mercado americano (9) (11) (10);
- a adição de extensões na API que permita o controle de diferentes recursos numa rede doméstica, tais como luz, temperatura e os eletrodomésticos (*Home Automation)*;
- a adição de extensões que permitam a expansão dos modos de interface atualmente disponíveis, em particular a Realidade Virtual.

REFERÊNCIAS

1. **Maior, M. S.** TV interativa e seus caminhos. *Dissertação (Mestrado profissional em Engenharia de Computação)*. Campinas : Instituto de Computação, Universidade Estadual de Campinas, 2002.

2. **MORRIS, S. e SMITH-CHAIGNEAU, A.** *Interactive TV Standards – A Guide to MHP, OCAP and JavaTV*. s.l. : Elsevier, Focal Press, 2005.

3. **Fernandes, J., Lemos, G. e Silveira, G.** Introdução à Televisão Digital Interativa. *Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação.* Salvador : SBC, 2004.

4. **SOARES, L. F. G., RODRIGUES, R. F. e MORENO, M. F.** Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System. *Journal of the Brazilian Computer Society (ISSN: 0104-6500).* 2007. pp. n. 4, v. 13. p.37-46.

5. **SOUZA FILHO, G. L. de, LEITE, L. E. C. e BATISTA, C. E. C. F.** Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. *Journal of the Brazilian Computer Society (ISSN: 0104-6500).* 2007. pp. n. 4, v. 13. p.47-56.

6. **Miranda, Leonardo Cunha de.** Uma Proposta de Taxonomia e Recomendação de Utilização de Artefatos Físicos de Interação com a TVDI. *INTERACT - CLIHC - 2007.* 2007.

7. **Roibás, A.C., Sala, R., Ahmad, S.N.A.S. e Radman, M.** Beyond the remote control: Going the extra mile to enhance iTV access via mobile devices & humanizing navigation experience for those with special needs. *Proceedings of the 3rd European Conference on Interactive Television.* 2005.

8. **SILVA, Lincoln David Nery e, et al.** Suporte para desenvolvimento de aplicações multiusuário e multidispositivo para TV Digital com Ginga. *T&C Amazônia Magazine. N. 12, ISSN 1678-3824, pp 75-84.* Manaus, AM : s.n., 2007.

9. **Globo.** Big Brother Brazil. *Globo.com.* [Online] http://bbb.globo.com/.

10. **DAVIC.** DAVIC 1.4.1 Specification Part 9: Information Representation. 1999.

11. **HAVi.** HAVi v1.1 - Home Audio Video Interoperability Version 1.1. s.l. : www.havi.org, 2001.

12. **Sun.** Java TV 1.0 - Java TV API Technical Overview: The Java TV API White Paper, Version 1.0, Sun Microsystems. 2000.

13. —. JMF 2.0 - Java Media Framework 2.0 API Guide, Sun Microsystems. 1999.

14. **IEEE.** IEEE1394 - Standard for a High Performance Serial Bus. 1995.

15. **Lindholm, T. e Yellin, F.** The Java Virtual Machine Specification. *Second Edition.* s.l. : Addison-Wesley, 1999.

16. **Sun.** JSR62 - J2ME Personal Profile Specification 1.0. *Final Release.* 2002.

17. **ETSI.** ETSI TS 102 812 - Digital Video Broadcasting (DVB) – Multimedia Home Platform (MHP) Specification 1.1.1. 2003.

18. **ATSC.** Standard A/100 - DTV Application Software Environment – Level 1 (DASE-1). 2003.

19. **CableLabs.** OpenCable Application Platform Specification – OCAP 1.0 Profile. s.l. : Cable Television Laboratories, 2005.

20. **ATSC.** ATSC Standard, Advanced Common Application Platform (ACAP). s.l. : Advanced Television Systems Committee, 2005.

21. **ARIB.** ARIB STD B24 - ARIB Standard: Data Coding and Transmission. *Revision 3.2. 2002.* 2002.

22. **ITU.** ITU Recommendation J.200 – Worldwide common core – Application environment for digital interactive television services. 2001.

23. —. ITU Recommendation J.201 – Harmonization of declarative content format for interactive television applications. 2004.

24. —. ITU Recommendation J.202 – Harmonization of procedural content formats for interactive TV applications. 2003.

25. **ETSI.** Globally Executable MHP (GEM) Specification 1.1. DVB Document A103, Rev. 1. 2007.

26. **LEITE, L. E. C. et al.** FlexTV – Uma Proposta de Arquitetura de Middleware para o Sistema Brasileiro de TV Digital. *Revista de Engenharia de Computação e Sistemas Digitais, v. 2, p 29-50.* 2005.

27. **SOARES, L. F. G.** MAESTRO: The Declarative Middleware Proposal for the SBTVD. *4th European Interactive TV Conference.* Atenas : Proceedings of the 4th European Interactive TV Conference, 2006.

28. **IBGE.** Pesquisa Nacional por Amostra de Domicílios. [Online] 2006. http://www.ibge.gov.br/home/estatistica/populacao/trabalhoerendimento/pnad2005/tabsintese.shtm.

29. **Nokia.** Nokia Brasil. *Nokia N80.* [Online] http://www.nokia.com.br/A4524762.

30. —. Nokia Brasil. *Nokia E61.* [Online] http://www.nokia.com.br/A4524678.

31. **Booch, G., Rumbaugh, J. e Jacobson, I.** *UML - Guia do Usuário.* Rio de Janeiro : Elsevier, 2005.

32. **TAVARES, Tatiana A., et al.** Sharing Virtual Acoustic Spaces over Interactive TV Programs – Presenting "Virtual Cheering" Application. *ICME 2004.* Taipei, Taiwan : s.n., 2004.

33. **ESPN.** ESPN The Ultimate Remote. [Online] http://www.espnremote.com/.

34. **Ricavision.** VAVE100 Universal Remote Control. [Online] http://www.ricavision.com/vave100.html.

35. **Logitech.** Harmony® One Advanced Universal Remote. [Online] http://www.logitech.com/index.cfm/remotes/universal_remotes/devices/3898&cl=us,en.

36. **Barros, G.G.** A consistência da interface com o usuário para a TV interativa. *Dissertação (Mestrado em Engenharia).* s.l. : Universidade de São Paulo, 2006.

37. **Jääskeläinen, Kari.** Strategic Questions in the Development of Interactive Television Programs. *Dissertation.* s.l. : University of Art and Design Helsinki UIAH A 31, 2001.

38. **Piccolo, L. e Baranauskas, M.C.C.** Desafios de Design para a TV Digital Interativa. *Anais do VII Simpósio de Fatores Humanos em Sistemas Computacionais.* 2006.