UNIVERSIDADE FEDERAL DO CEARÁ - UFC
CENTRO DE TECNOLOGIA - CT
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA – DETI

ARACELI DO NASCIMENTO TEIXEIRA

TV DIGITAL NO BRASIL: HISTÓRICO, EVOLUÇÃO E DESAFIOS

FORTALEZA
2010

ARACELI DO NASCIMENTO TEIXEIRA

TV DIGITAL NO BRASIL: HISTÓRICO, EVOLUÇÃO E DESAFIOS

Monografia apresentada à disciplina de Projeto de Final de Curso do Curso de Engenharia de Teleinformática da Universidade Federal de Ceará.

Orientador: Prof. Dr. Carlos Pimentel de Sousa

FORTALEZA

2010

TERMO DE APROVAÇÃO

ARACELI DO NASCIMENTO TEIXEIRA

TV DIGITAL NO BRASIL: HISTÓRICO, EVOLUÇÃO E DESAFIOS

Monografia apresentada ao Programa de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará, como requisito para obtenção do título de Engenheiro de Teleinformática.

Aprovado em __/__/____, com nota ___

_____
Prof. Dr. Carlos Pimentel de Sousa

Banca Examinadora:
Primeiro Examinador: _____
Segundo Examinador: _____
Terceiro Examinador: _____

FORTALEZA

2010

Dedico este trabalho aos meus pais, que sempre me incentivaram na vida acadêmica, e ao meu esposo, que continuamente permaneceu ao meu lado me dando carinho e forças nos momentos mais difíceis.

RESUMO

A TV digital, apesar de já existir a vários anos, aparenta estar ainda nas suas fases iniciais no Brasil. O Sistema Brasileiro de Televisão Digital (SBTVD) foi concebido para proporcionar à população brasileira as melhores experiências que se pode obter com a TV digital, mas o que se percebe é que muitas das funcionalidades ainda não estão disponíveis para os usuários.

O desenvolvimento de aplicativos para o Ginga, o *middleware* brasileiro, parece se restringir primordialmente ao ambiente acadêmico, sendo muitas as aplicações criadas dentro das universidades e poucas as disponibilizadas pelas empresas emissoras para a população. O Ginga-J, subsistema do Ginga que faz uso da linguagem Java para implementar seus aplicativos, apenas recentemente foi definido pela ABNT, o que contribuiu para a demora sua utilização para a criação de forma fácil de aplicações interativas.

Este trabalho visa expor a história da TV digital e dos principais padrões existentes no mundo de forma resumida e apresentar as características do Ginga-J e do seu modelo de aplicação, de forma a demonstrar a facilidade da implementação de aplicações para o *middleware* brasileiro.

ABSTRACT

The digital TV, although it has been released several years ago, appears to be still in its early stages in Brazil. The Brazilian Digital Television System (SBTVD) was designed to give the Brazilians the best experiences you can get with digital TV, but what can be seen is that many of the features are not yet available to users.

Developing applications for Ginga, the Brazilian middleware, seems to be restricted primarily to the academic environment, with many applications being created within the universities and few being available by the television station companies for the population. Ginga-J, the Ginga subsystem which uses the Java language to implement their applications, only recently been defined by ABNT, which contributed to delay its use for creating interactive applications in an easy way.

This work aims to expose the history of digital TV and the main systems in the world and present a summary of the characteristics of the Ginga-J and its application model in order to demonstrate the ease of application development to the Brazilian middleware.

Sumário

1. Introdução

Desde o seu surgimento, a televisão tem representado um papel muito importante na sociedade. Presente em praticamente todos os lares, ela serve principalmente como veículo de comunicação e entretenimento, além de impactar também as áreas de cultura, comércio, educação, política e até mesmo como meio de formação de opinião. No Brasil ela assumiu um papel também de integração, devido à sua grande popularidade, chegando a estar presente em praticamente todos os lares.

Nos últimos anos, tem-se vivido a expectativa de uma novidade tecnológica que vem aquecendo o mercado brasileiro de televisores: o advento da TV digital. Embora essa tecnologia exista desde a última década do século passado, apenas nos últimos meses ela ganhou destaque e se espalhou pelo país, atraindo mais emissoras e telespectadores.

O Sistema Brasileiro de Televisão Digital (SBTVD) foi totalmente elaborado buscando facilitar a oferta de todos os recursos dessa tecnologia, unindo os principais benefícios dos sistemas já existentes e tentando corrigir seus pontos fracos, além de incorporar os avanços tecnológicos obtidos depois da criação dos outros sistemas. A ideia prioritária do governo foi tornar a TV um mecanismo inclusão digital, pois, por ser de mais fácil acesso, ela atinge uma parcela maior da população do que os computadores.

No entanto, algumas das principais características da TV digital ainda não estão bastante difundidas, como a interatividade, a convergência entre diferentes meios de comunicação, a *internet* e a transmissão de dados, sendo poucas as aplicações existentes e em uso implementadas para o público brasileiro. Com isso, o benefício que se tem observado ser mais utilizado, por enquanto, é a maior

qualidade das imagens, difundido principalmente após a explosão das vendas de televisores de alta definição causada pela baixa nos preços dos aparelhos de LCD e plasma, embora a tecnologia da HDTV (televisão de alta definição) seja independente da TV digital.

1.1. Motivação

A disseminação do conhecimento é fundamental para que uma tecnologia cresça e tenha mais usuários. Com a TV digital não é diferente. Grande parte da população ignora que um *middleware* brasileiro foi especialmente criado para prover a TV digital aqui no Brasil de todos os recursos que poderiam ser úteis à realidade do povo. Aliado a isso, o fato de que as normas ABNT NBR 15606-4:2010, que regulamenta o Ginga-J, e ABNT NBR 15606-6:2010, que especifica o JavaDTV, demoraram muito para ficarem prontas também contribui para que as empresas não tenham investido muito na implementação de todas as funcionalidades que a TV digital oferece.

Entretanto, é importante que a população possa desfrutar de todo o potencial da TV digital, pois, muito mais do que um simples meio de difusão de vídeos, ela possibilita o desenvolvimento de aplicações interativas, tais como realizar pesquisas de opinião, aplicações para pessoas portadoras de deficiência, tais como controlar a TV utilizando o reconhecimento de voz, aplicações de alto nível de segurança, como, por exemplo, realizar transações bancárias, dentre outras.

1.2. Objetivos

Tendo tudo isso em mente, foi produzido este trabalho para expor o processo que trouxe a TV digital ao Brasil e levou ao desenvolvimento de um *middleware* brasileiro, assim como promover o entendimento do Ginga e suas características, do Ginga-J e suas tecnologias e do modelo de aplicação por ele empregado. Com todo esse conhecimento teórico, tem-se por objetivo facilitar o desenvolvimento de novos aplicativos para o Ginga-J.

2. Histórico da TV Digital

A televisão convencional, analógica, teve seu ponto de partida no ano de 1924, quando Jonh Logie Baird desenvolveu um sistema semi-mecânico que permitiu a transmissão das primeiras imagens. Somente mais tarde, em 1927, foi que Philo Taylor Farnsworth desenvolveu o primeiro sistema eletrônico completo de televisão.

O primeiro serviço analógico de televisão foi disponibilizado em 11 de maio de 1928, na cidade de Nova York pela WGY, mas foi apenas em 1936 que foi fundada a BBC de Londres, considerada o primeiro canal de TV. Na década de 50, o surgimento da TV em cores possibilitou melhores imagens e um consequente aumento da popularização da televisão. Outras descobertas, como a TV a cabo e por satélite, contribuíram para o aumento do número de canais e da qualidade da imagem.

Com o tempo, os sistemas foram aprimorados e passou-se a desejar mais qualidade nas imagens recebidas. Na década de 70 começaram a surgir as primeiras pesquisas sobre TV de alta definição, realizadas pelos japoneses. O mundo se interessou por essas pesquisas e tanto europeus quanto americanos passaram também a desenvolver estudos nessa área. Com isso, surgiram os diversos sistemas de televisão digital.

Na década de 80, foi criado nos Estados Unidos *o Advisory Committee on Advanced Television Service*, que tinha por objetivo ajudar no desenvolvimento de novas tecnologias e fomentar políticas para que os resultados obtidos nas pesquisas pudessem ser postos em prática. Ao longo da década de 90, quatro sistemas digitais foram desenvolvidos e testados, e foi proposta a criação de um novo sistema que reunisse características dos sistemas digitais testados.

Surgiu então o *Advanced Television Systems Committee* (ATSC), formado por representantes das empresas de eletrônicos, informática e telecomunicações, das emissoras e produtoras e das associações e instituições de pesquisa. O ATSC documentou as especificações desejadas para o novo sistema, reuniu os padrões da HDTV e da SDTV e formulou, por fim, o padrão ATSC, que, em novembro de 1995, foi recomendado por um comitê da *Federal Communications Commission* (FCC) como sistema a ser adotado para as transmissões terrestres de televisão nos Estados Unidos e homologado no ano seguinte como o novo padrão oficial.

Na Europa, embora os sistemas HDTV tenham sido desenvolvidos antes do que nos Estados Unidos, não houve avanços significativos nas transmissões terrestres de televisão. Apenas no início da década de 90 é que foram iniciados debates sobre um possível sistema próprio de TV digital.

Em 1991, emissoras e empresas do segmento de eletroeletrônicos se reuniram e constituíram o *European Launching Group* (ELG), em conjunto com os órgãos reguladores de comunicações de diversos países europeus. O ELG permitiu uma maior colaboração entre seus membros no sentido de estabelecer metas e elaborar ações para alcançá-las. A partir de 1993, com a conversão da Comunidade Econômica Européia em União Européia (UE) e com a assinatura de um memorando selando o acordo de cooperação mútua, o ELG foi convertido *no Digital Vídeo Broadcasting* (DVB) e as pesquisas sobre o novo padrão europeu de televisão digital enfim ganharam fôlego.

As primeiras modalidades de transmissão a utilizar o DVB foram o sistema digital via satélite e a televisão a cabo, pois não apresentaram tantos problemas quanto à transmissão terrestre e ainda tiveram prioridade no mercado, devido à capacidade de atender mais rapidamente às suas demandas. Mesmo assim, a transmissão terrestre digital apresenta boa perspectiva de rentabilidade, pois

permite a transmissão de dados e de *internet* banda larga, além de televisão por assinatura.

O Japão, por sua vez, embora tenha sido o primeiro a desenvolver a televisão analógica de alta definição, acabou por se afastar das pesquisas sobre TV digital e ficou defasado tecnologicamente em relação aos Estados Unidos e à Europa. No entanto, esse atraso na definição de seu sistema de transmissão da TV digital permitiu que mais novidades e tecnologias fossem incorporadas, dando origem a um sistema com maior convergência tecnológica.

O sistema nipônico foi criado pelo consórcio *Digital Broadcasting Experts Group* (DiBEG), que buscou desenvolver uma plataforma com múltiplos serviços tecnológicos, abrangendo inclusive a transmissão digital de televisão. Surgia assim, o *Integrated Services Digital Broadcasting* (ISDB).

Além desses três padrões de TV digital, alguns outros países também desenvolveram pesquisas e criaram sistemas que melhor se adequavam à sua realidade. A China, por exemplo, a partir da segunda metade da década de 90 começou a analisar os sistemas já existentes e estudar o desenvolvimento de um sistema próprio, e, a partir de 2001, iniciou a elaboração das especificações e os testes para o *Digital Multimedia Broadcast* (DMB).

O Brasil, por sua vez, começou a dar seus primeiros no sentido de implantar a televisão digital no país em 1991, quando foi criada a Comissão Assessora de Assuntos de Televisão (COM-TV), que tinha por objetivo inicial fomentar a TV de alta definição e o suporte tecnológico necessário para a mesma. Em 1994, iniciaram-se os estudos para a implantação da TV digital a partir da criação de um grupo técnico formado pela Sociedade de Engenharia de Televisão (SET) e pela Associação Brasileira de Emissoras de Rádio e Televisão (Abert).

Mas foi apenas em 1998, com a criação da Agência Nacional de

Telecomunicações (Anatel), que uma proposta para a realização de testes foi publicada, buscando-se definir qual dentre os diferentes padrões de TV digital seria o mais tecnicamente viável. No ano seguinte, a Anatel definiu que os testes de laboratório e de campo seriam realizados pelo grupo Abert/SET, em convênio com a Universidade Mackenzie, e que a Fundação do Centro de Pesquisa e Desenvolvimento (CPqD) definiria a metodologia a ser empregada nos testes e analisaria os resultados. Juntamente com entidades civis, tais como representantes dos consórcios detentores das patentes dos padrões ATSC e DVB, o convênio Abert/SET-Mackenzie esboçou os planos de implantação do sistema digital no país, as políticas e regulamentações para o novo padrão e os impactos do mesmo na sociedade.

No início do ano 2000, o grupo Abert/Set divulgou um relatório técnico sobre os padrões que estavam em teste. Através desse documento, o grupo sugeria que a modulação a ser adotada deveria ser a COFDM, excluindo assim o padrão americano dentre as opções para o sistema a ser adotado no Brasil, uma vez que o ATSC utiliza a modulação 8-VSB. Além disso, o resultado dos testes apontou que o padrão japonês era mais robusto e permitia maior imunidade ao ruído e melhor sinal para os receptores com antena interna e possibilitava a instalação dos novos canais digitais em paralelo com a manutenção dos canais analógicos.

Apesar da preferência pelo ISDB-T demonstrada no relatório, não houve decisão imediata sobre qual padrão a ser utilizado no Brasil. O governo decidiu adicionar mais requisitos e grupos da sociedade manifestaram o desejo de ter maior participação na escolha, o que levou ao decreto 4.901 de 26 de novembro de 2003, que instituía a criação de um grupo de trabalho para definir um modelo de referência para a TV digital no Brasil, dando origem ao programa Sistema Brasileiro de Televisao Digital (SBTVD).

Em 29 de junho de 2006, o governo por fim decidiu, com o decreto 5.820, definir como o padrão brasileiro o Sistema Brasileiro de Televisão Digital Terrestre (SBTVD-T, com nome comercial ISDB-Tb), baseado no ISDB-T e com interatividade, suporte à transmissão digital na definição padrão (SDTV) e em alta definição (HDTV) e suporte à transmissão digital simultânea para dispositivos fixos, móveis e portáteis. Já no final de 2006, o Fórum do Sistema Brasileiro de TV Digital Terrestre (Fórum SBTVD) foi criado com o objetivo de dar suporte à criação do padrão brasileiro e promover a cooperação entre os diversos setores da sociedade com interesses na TV digital.

As primeiras transmissões digitais no Brasil ocorreram em dezembro de 2007, inicialmente apenas em São Paulo, mas chegou a abranger cerca de 39% da população, ou aproximadamente 75 milhões de pessoas, até o início de novembro de 2010, de acordo com dados divulgados pelo Fórum SBTVD.

3. Características dos Padrões de TV Digital

Os padrões americano, europeu e japonês possuem características distintas, que, ao serem analisadas, contribuíram para a escolha do padrão que se tornaria a base para o sistema brasileiro.

3.1. ATSC

O principal foco do americano ATSC é a alta qualidade audiovisual. Sucessor do NTSC, o padrão analógico utilizado nos Estados Unidos, o ATSC trabalha com seis vezes mais *pixels*, fornecendo imagens com resolução 1920x1080 em proporção 16:9 e codificação de vídeo MPEG-2, e utiliza o sistema de som *Dolby Digital* com codificação *Dolby* AC-3, que utiliza seis canais de áudio (5.1), resultando em qualidade semelhante aos home theaters.

Este padrão usa os canais de 6 MHz da TV analógica e a modulação utilizada para a transmissão é a QPSK, para satélite, o 64QAM para cabo e o 8-VSB (*8 Level - Vestigial Side Band Modulation*) para a radiodifusão terrestre. Essa última sofreu várias críticas de especialistas, pois sofre interferência mais facilmente, degradando o sinal em regiões com obstáculos, tais como edifícios. Outra característica apontada como desvantagem é o fato de a capacidade de transmissão ser limitada a apenas um programa por canal. Além disso, o sistema não foi projetado para suportar receptores móveis ou portáteis, principalmente porque a prioridade foi dada para a alta qualidade de áudio e vídeo, tornando a decodificação mais pesada dispositivos como celulares.

O *middleware* primeiramente utilizado no ATSC foi o *DTV Application Software Enviornment* (DASE), que possibilita a utilização de aplicações interativas

em qualquer receptor. As aplicações desenvolvidas para este *middleware* podem ser declarativas, fazendo uso de tecnologias *web* tais como XHTML, CSS e DOM, procedurais, com códigos escritos em Java, ou híbridas, onde o conteúdo é gerado tanto através de aplicativos declarativos como procedurais.

No entanto, para as transmissões de DTV a cabo, um setor mais forte nos EUA do que a transmissão terrestre, a empresa CableLabs desenvolveu a *OpenCable Applications Platform* (OCAP), que rapidamente se tornou o sistema padrão para a transmissão via cabo. Um dos maiores trunfos da OCAP é que ela foi desenvolvida seguindo a estrutura do *Globally Executable MHP* (GEM), uma série de orientações para desenvolver *middlewares* compatíveis com o MHP, o *middleware* europeu. Na OCAP, muitas tecnologias e especificações DVB que não são usadas no ambiente de cabo norte-americano são retiradas e substituídas por seus equivalentes funcionais, conforme especificações no GEM.

Mais recentemente, o DASE está sendo substituído nos Estados Unidos pelo *Advanced Common Application* (ACAP), um *middleware* que é o resultado da harmonização das plataformas OCAP e DASE, que assegura compatibilidade entre as transmissões por cabo e terrestres. Tal como o OCAP, o ACAP é derivado do padrão MHP, por intermédio do GEM.

3.2. DVB

Já o padrão europeu, o DVB, foi desenvolvido para ser mais robusto e flexível que o ATSC. Ele utiliza modulação COFDM (*Coded Orthogonal Frequency Division Multiplexing*) para a radiodifusão terrestre, possibilitando melhor recepção em dispositivos com antena embutida, pois o sinal, antes de ser modulado em OFDM, é codificado para possibilitar a correção de erros. Neste esquema de

modulação, o sinal é dividido e transportado através de várias portadoras ortogonais, com a possibilidade de utilizar diferentes modulações para elas, tais como QPSK, 16-QAM ou 64-QAM.

A taxa de bits enviados pode ser alterada dependendo da necessidade, possibilitando que o sinal chegue com melhor qualidade em diversas áreas, e os canais podem utilizar as freqüências de 6 MHz, 7 MHz ou 8 MHz. Além disso, há o suporte à multiprogramação, com a possibilidade da transmissão de diferentes resoluções para um mesmo programa ou de vários programas em um mesmo canal, sendo que a codificação de áudio e vídeo é feita através do sistema MPEG-2.

No entanto, embora exista a possibilidade de transmitir conteúdo para dispositivos móveis, a recepção não é satisfatória, principalmente quando é utilizado o modo de transmissão hierárquico, no qual há a transmissão simultânea de dados para televisão de alta definição e receptores móveis. Esse é considerado o ponto fraco do sistema DVB.

O *Multimidia Home Plataform* (MHP) é o *middleware* do DVB e possibilita o fornecimento de serviços através da definição de uma interface entre os dispositivos e as aplicações. Ele possui suporte para aplicações procedurais, desenvolvidas para o DVB-J, e declarativas, com funções do DVB-HTML. O MHP trabalha com os seguintes perfis de suporte a plataformas de serviços:

- *Enhanced Broadcast*: Permite a transmissão de áudio e vídeo e serviços de *download* de aplicações e *broadcast*. A interação é apenas a nível local, pois o *Enhanced* não suporta canal de retorno.

- *Interactive Broadcast*: Além de englobar as funcionalidades do *Enhanced*, este perfil requer canal de retorno, permitindo interatividade remota, e suporta *Internet Protocol* (IP).

- *Internet Access*: Perfil presente no MHP a partir da versão 1.1, possui

todas as características das configurações anteriores e ainda permite acesso à *internet*. Suporta aplicações *web*, como navegador para e-mail, além de aplicações desenvolvidas em Java. Interações entre serviços *broadcast* e serviços *internet* também são possíveis.

A primeira versão do MHP, a 1.0, trouxe apenas o suporte a aplicações procedurais com o DVB-J, que se baseia em uma API específica para TVD criada pela Sun Microsystems, a JavaTV. A partir da versão 1.1, o MHP possibilitou o armazenamento local de *plugins* e aplicações recebidos por difusão, além do acréscimo da API DVB-HTML, uma interface de programação de aplicações baseadas em HTML. Além disso, já foi lançado também o MHP 1.2, com adição de suporte para redes *Internet* Protocol (IP) de banda larga como meio de transmissão do sinal digital de televisão (IPTV).

O MHP foi inicialmente projetado para rodar nas plataformas DVB, mas, como houve demanda para estender a interoperabilidade que ele oferece a outras plataformas de televisão digital, surgiu o GEM (*Globally Executable MHP*), uma estrutura que permite a outras organizações definirem especificações baseadas em MHP. As regras do projeto, que define APIs, protocolos e formatos para conteúdos multimídia, permitem aos desenvolvedores escrever aplicações que podem ser diretamente interoperáveis através dos receptores baseados na tecnologia GEM.

Atualmente, a tendência é que, cada vez mais, as APIs se harmonizem para que haja um núcleo comum de desenvolvimento de aplicações, pois desenvolver diferentes versões de aplicações é mais complexo do que criar conversores para a camada de difusão. Espera-se, também, que as mesmas APIs sejam utilizadas em transmissões terrestres, via satélite ou cabo, e que apenas poucos recursos específicos de cada padrão exijam adaptação do código.

3.3. ISDB

O padrão ISDB é considerado uma melhoria do DVB, pois é superior no quesito imunidade à interferência, o que o torna capaz de transmitir sinal de alta definição juntamente com o sinal para dispositivos móveis sem os problemas que aconteciam no DVB. Assim como o padrão europeu, o ISDB utiliza a modulação COFDM e a codificação MPEG-2 para áudio e vídeo. Ele oferece também a possibilidade da transmissão hierárquica, onde um mesmo programa pode apresentar diferentes resoluções.

A modulação das portadoras nesse padrão podem ser em 16-QAM, 64-QAM, QPSK ou DQPSK, e a quantidade das mesmas pode ser de 2K, 4K e 8K. O modo de operação em 4K representa um avanço do ISDB em relação ao DVB, pois atende às características de transmissão necessárias para a recepção em dispositivos fixos e em dispositivos móveis. Outra característica que foi melhorada no padrão japonês foi o entrelaçamento (*interleaving*), que passa a ser tanto na freqüência, como no tempo, conferindo maior robustez ao sistema.

Um aspecto muito importante do ISDB é a divisão de um mesmo canal em 13 segmentos diferentes. Dessa forma, é possível alcançar o objetivo principal para o qual o sistema foi criado, que é possibilitar a convergência de vários serviços de comunicação em uma única plataforma tecnológica. O padrão japonês une televisão digital, *Internet* e telefones celulares 3G com uma tecnologia robusta, permitindo, por exemplo, o desenvolvimento de dispositivos capazes de receber transmissões digitais terrestres em HDTV sem distorção na imagem, mesmo que em movimento.

Para compor a camada de *middleware* foi criado o padrão ARIB (*Association of Radio Industries and Business*), que define como se deve dar a multiplexação e transmissão via *broadcasting* de rádio do áudio, vídeo e serviços de dados do

sistema japonês. A transmissão se dá através do *Transport Stream* (TS) definido pelo MPEG-2 e os canais para interatividade são disponibilizados por meio dos canais interativos da rede.

Este *middleware* define três tipos de sistema de transmissão de dados, diferenciados pela forma com que os dados são armazenados. No *Data Stream*, o armazenamento de pacotes é realizado como um fluxo do PES (*Packetized Elementary Stream*), de forma que aplicações de tempo real ou que necessitem ser sincronizadas com outros fluxos se tornam possíveis.

O sistema de *Data Carrousel* é bastante utilizado para armazenamento de informação por meio de seções, pois dados que seriam transmitidos repetidamente são guardados no receptor logo após o primeiro *download*, permitindo que eles possam ser reutilizados sempre que for preciso. Já o terceiro sistema de transmissão define que os dados sejam armazenados diretamente no *payload* do pacote do TS, mas este sistema só será utilizado caso uma expansão da especificação se tornar necessária.

Os receptores do ARIB possuem como requisito a capacidade de receber e disponibilizar serviços multimídia. Além das funções comuns a todos os receptores de TV, eles devem possuir as funções de recepção e armazenamento através de receptores de baixo custo e apresentação do conteúdo da exata forma como ele foi transmitido.

3.4. SBTVD

Baseado no ISDB-T, surgiu no Brasil o ISDB-Tb ou SBTVD-T, como já foi explicado antes. O padrão brasileiro manteve a modulação COFDM, os canais com largura de banda de 6 MHz e a segmentação de canal do sistema japonês, mas a

codificação de vídeo passa a ser feita através do MPEG-4 AVC (H.264) e a de áudio pelo MPEG-4 AAC a 48 KHz. Para receptores portáteis, a taxa de apresentação de quadros foi dobrada, passando de 15 fps no ISDB para 30 fps no SBTVD.

O sistema brasileiro permite, por canal, a transmissão de um programa HDTV (1920x1080), um HD (1280x720) e um SDTV (720x480 ou 720x576), ou três programas SDTV simultaneamente. No entanto, a multiprogramação ainda foi bloqueada para uso comercial no Brasil, estando restrita somente a uso governamental. Há uma grande expectativa para que a mesma seja liberada para as emissoras comerciais, pois é uma das características mais atrativas da TV digital e que já está presente em diversos países, embora algumas emissoras brasileiras prefiram que ela continue apenas a nível governamental.

Outro grande diferencial do modelo de TV digital implantado no Brasil em relação ao modelo japonês é a utilização de uma camada de *middleware* com maior suporte à interatividade, o Ginga. Conforme será mais bem explicado no próximo capítulo, o Ginga consegue unir um ambiente declarativo a um procedural de modo a formar um dos melhores *middlewares* do mundo para TV digital, com a vantagem de possuir a especificação aberta.

4. O *Middleware* Ginga

Middleware é uma camada intermediária que realiza a mediação entre o software e o *hardware*, possibilitando a integração entre as aplicações de alto nível e as interfaces de baixo nível, além de permitir a comunicação entre programas de diferentes plataformas, sistemas operacionais ou protocolos de comunicação. Para os diversos padrões de TV Digital, o *middleware* é a camada responsável pelo suporte à interatividade, grade de programação, suporte à execução de aplicações, dentre outras facilidades.

O Ginga surgiu como fruto da parceria entre duas universidades brasileiras, a Universidade Católica do Rio de Janeiro (PUC-Rio) e a Universidade Federal da Paraíba (UFPB), que se basearam nos *middlewares* FlexTV e MAESTRO para criar um novo que unisse os paradigmas da programação procedural e da declarativa. Assim, foram criados os subsistemas que hoje compõem o Ginga, cujo nome foi escolhido como uma referência e homenagem à cultura brasileira.

A escolha pelo desenvolvimento de um *middleware* próprio para o SBTVD, ao invés da utilização dos *middlewares* dos outros padrões já existentes, deveu-se tanto pelo fato de que os outros *middlewares* não atendiam às expectativas do que seria melhor para o padrão brasileiro, como pelo fato de que, com essa decisão, foi evitado o pagamento de *royalties* pelo simples uso da especificação do *middleware*, barateando o custo dos receptores. Além disso, desenvolver o *middleware* no Brasil contribui para o fortalecimento da indústria nacional de *software* e o fortalecimento do conhecimento sobre *middleware*.

Os principais subsistemas que integram o Ginga são o Ginga-CC (Ginga Common-Core), o Ginga-NCL e o Ginga-J. Além deles, há a máquina virtual Java, base para a execução das aplicações dos Ginga-J, e uma ponte para comunicação

entre o Ginga-NCL e o Ginga-J, que permite que os ambientes de apresentação e execução se complementem, com uma implementação sem redundâncias.

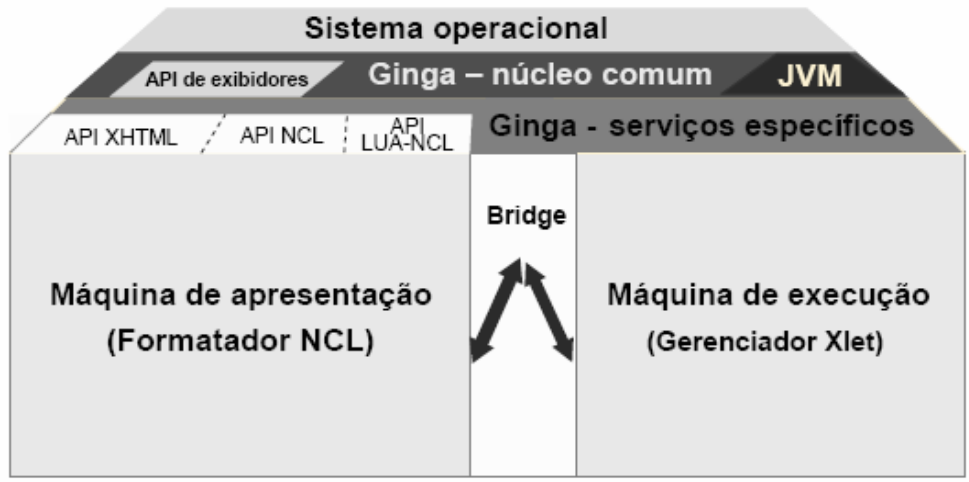

Figura 1 - Arquitetura GINGA

A ponte é utilizada para a utilização de aplicações híbridas. Há as aplicações declarativas híbridas, que são aquelas constituídas por um documento NCL que faz uso de um Xlet procedural, e há as aplicações procedurais híbridas, que são constituídas por um Xlet que faz uso de um objeto NCL.

4.1. Ginga-CC

O Ginga-CC oferece o suporte básico tanto para a máquina de apresentação (declarativo) quanto para a máquina de execução (procedural), de forma que funcionalidades comuns a ambas são tratadas por ele, tais como decodificação de áudio e vídeo e controle do plano gráfico e do canal de retorno.

Este subsistema foi projetado de forma que ele abstraia a camada de *hardware*, sendo, portanto, o único a sofrer alterações para a adaptação a diferentes plataformas. Além disso, o núcleo comum faz interface com o sistema operacional, lidando de forma mais estreita com o *hardware* e provendo acesso aos recursos

através de APIs (*Application Program Interfaces*) bem definidas.

O Ginga-CC tem, entre seus componentes, um conjunto de exibidores (*players*) monomídia comuns, cujas características estão definidas na Norma ABNT NBR 15606-1:2010. Eles são exibidores de áudio, vídeo, texto e imagem, incluindo o exibidor MPEG-4/H.264, implementado por *hardware*. O acesso a eles se dá através de adaptadores, responsáveis por notificar eventos de apresentação e seleção (interação do usuário). Entre eles também se encontra o exibidor (agente do usuário) HTML, especificado nas normas ABNT NBR 15606-2:2007 e 15606-5:2008.

Figura 2 – Componentes do Ginga-CC

O gerenciador gráfico é o responsável pelo gerenciamento do modelo conceitual do plano gráfico de apresentação. É ele que define o plano de exibição do vídeo principal, os planos de exibição dos outros objetos de mídia que compõem uma aplicação TVD, e como esses planos se superpõem. A norma ABNT NBR 15606-1:2010 é responsável também por tal definição.

Todo o acesso a dados obtidos através do canal de retorno (ou canal de interatividade) é também de responsabilidade do Ginga-CC. As diversas possibilidades de canal de interatividade são especificadas na Norma ABNT NBR 15607.

4.2. Ginga-NCL

De responsabilidade da PUC-Rio, este módulo é responsável pelo ambiente declarativo do *middleware* e provê a infra-estrutura para as aplicações escritas em NCL (*Nested Context Language*). Ele é constituído de três componentes: o formatador NCL, que é a máquina de interpretação do conteúdo declarativo, o exibidor XHTML, que inclui interpretadores CSS e ECMAScript, e a máquina de apresentação LUA, responsável pela interpretação dos scripts Lua.

A linguagem NCL, sozinha, oferece suporte a todos os requisitos desejáveis para um *middleware*. Ela foi desenvolvida dentro da própria PUC-Rio, com base no NCM (*Nested Context Model*) e seguindo os princípios do W3C (*World Wide Web Consortium*). Ela é uma das linguagens mais utilizadas para a definição do sincronismo espaço-tempo entre os objetos de mídia e tem por objetivo facilitar as especificações de interatividade, além de suportar múltiplos dispositivos e programas ao vivo interativos não-lineares.

Enquanto as máquinas de apresentação dos principais padrões de TV digital do restante do mundo se baseiam em XHTML, uma linguagem de marcação, cuja estrutura é definida pelo relacionamento entre objetos XHTML que estão embutidos no conteúdo das mídias do documento, o NCL, linguagem da máquina de apresentação do ambiente declarativo do Ginga, define uma separação bem demarcada entre o conteúdo e a estrutura de um documento (ou aplicativo), provendo um controle não invasivo da ligação entre o conteúdo e sua apresentação e layout.

Em um documento NCL, há apenas a definição de como os objetos de mídia são estruturados e relacionados no tempo e espaço. Assim, pode-se ter objetos de imagem (GIF, JPEG etc.), de vídeo (MPEG, MOV etc.), de áudio (MP3,

WMA etc.), de texto (TXT, PDF etc.), de execução (Xlet, Lua etc.), entre outros. O suporte a um determinado objeto de mídia depende dos exibidores de mídia que estão atrelados ao formatador NCL (exibidor NCL). Por exemplo, o decodificador/exibidor MPEG-4 é um desses exibidores e, dessa forma, o vídeo e o áudio MPEG-4 principal são tratados como todos os demais objetos de mídia que podem estar relacionados utilizando NCL.

Outro objeto de mídia NCL que deve obrigatoriamente ser suportado é o objeto de mídia baseado em XHTML. A NCL não substitui, mas embute documentos (ou objetos) baseados em XHTML. Como acontece com outros objetos de mídia, qual linguagem baseada em XHTML tem suporte em um formatador NCL é uma escolha de implementação e, portanto, depende de qual navegador XHTML, incorporado no formatador NCL, atua como exibidor dessa mídia.

Já a linguagem LUA, que também é utilizada no Ginga-NCL como linguagem de script, acabou por se tornar padrão para personalização de aplicações voltadas para a área de entretenimento, tais como jogos, pois é poderosa, leve, extensível e tem alto desempenho. Assim como o NCL, ela foi criada dentro da PUC-Rio.

Como ocorre com todos os exibidores de objetos de mídia, um exibidor Lua, uma vez instanciado, deve executar os procedimentos de iniciação do objeto NCL que controlará. Porém, diferente dos outros exibidores de mídia, o código de iniciação deve também ser especificado pelo autor do objeto NCLua. Os procedimentos de iniciação são executados apenas uma vez, para cada instância, e cria funções e objetos que podem ser usados durante a execução do objeto NCLua e, em particular, registra um ou mais tratadores de eventos para a comunicação com o formatador NCL.

Depois da iniciação, a execução do objeto NCLua torna-se orientada a

evento, em ambas as direções. Isto é, qualquer ação comandada pelo formatador NCL é dirigida aos tratadores de evento registrados, e qualquer notificação de mudança de estado de eventos NCL é enviada como um evento ao formatador NCL (como por exemplo, o fim da execução do código procedural). O exibidor Lua estará então pronto para executar qualquer instrução de start ou set.

Além das duas linguagens, a PUC-Rio também criou ferramentas para o desenvolvimento do Ginga-NCL, tais como o Composer, um ambiente de desenvolvimento gráfico para o *middleware* brasileiro.

O subsistema Ginga-NCL está presente tanto na versão 1.0 como na versão 2.0 do Ginga. Ao contrário do Ginga-J, sua presença é obrigatória tanto nos receptores fixos e móveis, quanto nos portáteis, pois se trata de um ambiente leve, exigindo menos do *hardware* geralmente não muito potente dos dispositivos portáteis.

4.3. Ginga-J

Desenvolvido pela UFPB, o subsistema Ginga-J provê o ambiente procedural para o SBTVD, mediante o suporte à infra-estrutura para execução de aplicações em Java. Constituído por APIs (*Application Programming Interface*), ele se divide em três módulos, o vermelho, o amarelo e o verde.

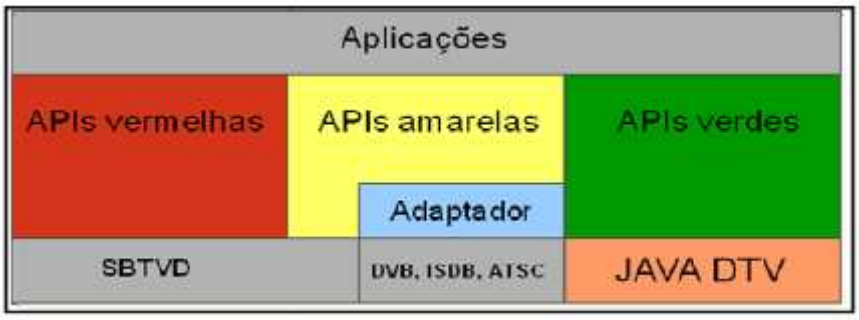

Figura 3 – APIS do Ginga-J

As APIs do módulo vermelho dão suporte aos requisitos do SBTVD específicos do Brasil, especialmente suporte à interatividade, promovendo a inclusão social. Permitem também a integração do conteúdo declarativo e procedural na mesma aplicação.

As APIs amarelas oferecem suporte a múltiplos usuários, dispositivos e redes, além de possibilitar o recebimento, armazenamento e futura execução de aplicações. O JMF (*Java Media Framework*), usado para desenvolver aplicações avançadas como a captura de áudio e vídeo, entre outras, está incluso nesse conjunto de APIs. Este módulo é uma inovação brasileira que também pode ser exportado para outros sistemas.

Já as APIs do módulo verde constituem o núcleo e são responsáveis por manter a compatibilidade entre o sistema nipo-brasileiro com o europeu e o americano. Para tanto, faz uso da API JavaDTV, que será detalhada no próximo capítulo.

5. Tecnologias para o Ginga-J

De acordo com a norma ABNT 15606-4:2010, o Ginga-J deve conter um mínimo de pacotes para que os desenvolvedores de aplicativos saibam com o que podem contar e, assim, seja possível manter a compatibilidade de uma aplicação com todos os receptores Ginga-J. Nessa lista estão inclusos pacotes da plataforma Java e das especificações JavaTV 1.1, JMF 1.0, JavaDTV 1.3, JSSE 1.0.1, JCE 1.0.1 e SATSA, além dos pacotes específicos Ginga-J.

Este capítulo dará uma breve visão sobre algumas destas tecnologias, explicando o seu nicho de atuação e sua contribuição para o Ginga-J. Será explorado também o GEM, para mostrar as diferenças entre esta especificação e o JavaDTV.

5.1. Java

A história do surgimento do Java remota de 1991, quando foi iniciado o projeto Green pela Sun Microsystems. Este projeto tinha por objetivo o estudo e a análise de interações entre dispositivos eletrônicos e computadores, não exatamente para o desenvolvimento de uma linguagem para aplicativos embarcados. Este projeto quase foi descontinuado pela Sun, mas o advento da *internet* revelou uma característica da linguagem que até então era chamada de Oak: dar mais dinamismo e interatividade às páginas da *Internet*.

Em 1995, a linguagem recebeu o nome de Java e o seu uso começou a ser rapidamente difundido, conquistando usuários no mundo todo. Desde seu lançamento, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Em 2004 Java atingiu a

marca de 3 milhões de desenvolvedores em todo mundo. Atualmente, o Brasil figura como o país a possuir uma das maiores comunidades Java do mundo.

Uma das principais características do Java é que se trata de uma linguagem que utiliza orientação a objetos. É também portável, ou seja, aplicações escritas em Java podem ser executadas em qualquer plataforma. Para tanto, é necessária a instalação de uma máquina virtual Java (JVM), pois ela converte os *bytecodes* do aplicativo em código executável. Além disso, a máquina virtual protege o ambiente no qual ela está instalada contra códigos maliciosos, pois ela permite o controle sobre o que será executado. Outra característica notável da linguagem é o fato da mesma possuir um coletor de lixo (*garbage colector*), permitindo a desalocação automática dos espaços de memória que não estão sendo mais utilizados.

De acordo com a norma ABNTNBR 15606-4:2010, para que uma implementação do Ginga-J seja considerada como de acordo com a especificação da ABNT, é necessário que certos pacotes estejam presentes, os quais são chamados de pacotes mínimos do Ginga-J. Da plataforma Java, é necessário, por exemplo, que os pacotes java.awt, java.beans, java.io, java.lang, java.math, java.net, java.rmi, java.security.spec, java.text, java.util, javax.microedition.io, javax.microedition.pki, javax.microedition.xlet e javax.security.auth.x500 estejam implementados, assim como alguns de seus sub-pacotes.

5.2. JMF

O *framework* de mídia do Java (*Java Media Framework*) é uma biblioteca Java que habilita as mídias de áudio, vídeo e outras baseadas em tempo a serem adicionadas nas aplicações e nos applets Java. Esse pacote opcional, que pode capturar, executar, fazer stream e transcodificar múltiplos formatos de mídia,

estende o Java SE (*Java Platform Standard Edition*) e permite o desenvolvimento de aplicações multimídia multi-plataforma.

O JMF abstrai a mídia com que ele está trabalhando em *DataSources* (fonte de dados) para dados que estão sendo exportados. Ele não dá ao desenvolvedor acesso significante às particularidades de qualquer formato, proporcionando antes a representação da mídia como fontes (elas mesmas obtidas através de URLs) que podem ser lidas, executadas, processadas e exportadas (embora nem todos os codecs suportem processamento e transcodificação).

O Ginga-J requer a implementação dos pacotes javax.media e javax.media.protocol desta especificação.

5.3. JSSE

A *Java Secure Sockets Extension* (JSSE) é a biblioteca de *sockets* que abstraem a utilização de criptografia na comunicação para garantir integridade, cifra e autenticação. É utilizada sobretudo para comunicação na *Internet*, constituindo uma camada adicional na pilha de protocolos de rede, entre o nível de transporte e o nível da aplicação.

Fazem parte do Ginga-J os pacotes com.sun.net.ssl, javax.net, javax.net.ssl e javax.security.cert desta especificação.

5.4. JCE

O Java *Cryptography Extension* (JCE) fornece um *framework* e implementações de criptografia, geração de chaves e conformidade de chave, e algoritmos de *Message Authentication Code* (MAC). O JCE foi anteriormente um

pacote opcional (extensão) para o Java 2 SDK, *Standard Edition*, nas versões 1.2.x e 1.3.x. Atualmente, esta especificação se encontra integrada ao Java 2 SDK a partir da versão 1.4.

O JCE é baseado nos mesmos princípios de design encontradas em outras partes do JCA: independência de implementação e, sempre que possível, independência de algoritmo. Ele também faz uso da mesma arquitetura de provedor. Provedores assinados por uma entidade confiável podem ser conectados ao *framework* JCE, e novos algoritmos podem ser adicionados sem problemas.

A API JCE abrange criptografia em massa simétrica, como o DES, RC2 e IDEA, criptografia simétrica de fluxos, como RC4, criptografia assimétrica, como o RSA, criptografia baseada em senha (PBE), concordância de chaves e código de autenticação de mensagens (MAC).

Os pacotes da JCE que devem estar presentes em uma implementação do Ginga-J são javax.crypto, javax.crypto.interfaces e javax.crypto.spec.

5.5. SATSA

A *Security and Trust Services API* (SATSA) é uma coleção de APIs que provêem serviços de segurança criptográfica para dispositivos habilitados pelo J2ME. Essas APIs são um passo necessário para um dispositivo se tornar confiável, de modo a fornecer mecanismos de segurança para dar suporte a uma ampla variedade de serviços baseados em aplicações, tais como acesso a redes corporativas, comércio móvel e gerenciamento de direitos digitais.

Muitos desses serviços dependem da interação com um *Security Element* (Elemento de Segurança) no dispositivo para armazenamento e execução seguros. O armazenamento seguro serve para proteger dados sensíveis, tais como chaves

privadas de usuários, certificados de chaves públicas, credenciais de serviços, informações pessoais, etc. A execução segura refere-se a operações de criptografia para dar suporte a protocolos de pagamento, integridade e confidencialidade de dados, dentre outros.

A SATSA permite a comunicação de uma aplicação Java ME com um *smart card* através dos protocolos APDU (*Application Protocol Data Unit*) e Java Card RMI (*Remote Method Invocation*). Ela possui algumas características, as quais não possuem suporte nativo na plataforma J2ME original, como armazenamento seguro e troca de dados com terceiros (tais como troca de dados durante transações de um pagamento), além de identificação e autenticação de usuário durante essas trocas.

O Ginga-J inclui o pacote javax.microedition.apdu da SATSA em sua lista de pacotes mínimos. Este pacote define o manipulador de protocolo APDU para comunicação ISO7816-4 com um dispositivo de *smart card*.

5.6. GEM

Quando a comunidade do SBTVD pensou em um ambiente procedural para o Ginga, foi natural que a linguagem escolhida fosse o Java, já em uso pelos *middlewares* de outros países.

Da mesma forma, cogitou-se utilizar o GEM como núcleo do Ginga-J, pois se trata de um padrão seguido a nível mundial, que facilita o desenvolvimento de aplicações compatíveis com outros sistemas de TV digital.

O GEM (*Globally Executable MHP*), derivado do MHP, é composto por, entre outros, três blocos fundamentais, DAVIC, HAVi e JavaTV, que devem estar presentes para que a compatibilidade com o MHP, ACAP e ARIB seja mantida.

A API JavaTV é uma extensão da plataforma Java que oferece um

framework para os *middlewares* e permite a produção de conteúdo para TV Interativa. Sua função é permitir o acesso aos dados transmitidos com o sinal de TV e a criação de aplicações interativas portáteis para TV, independentes da tecnologia de transmissão, além de oferecer controle dos *media players* específicos da TV via JMF.

JavaTV consiste basicamente de uma máquina virtual Java e bibliotecas de códigos reusáveis e específico para TV Interativa. A máquina virtual é residente no receptor e pode ser usada pelos outros blocos funcionais (como o DAVIC, HAVi e bibliotecas do MHP). Esta API possibilita níveis avançados de interatividade às aplicações, com ou sem canal de retorno, e associado ou não com fluxo de *broadcast* (vídeo e áudio).

DAVIC (*Digital Audio-Visual Council*) é uma associação sem fins lucrativos que representa diversas indústrias do segmento Audiovisual. Seu objetivo é especificar um padrão comum para difusão, além de sustentar a interoperabilidade real de ponta a ponta na execução de serviços de áudio e vídeo, transmitidos via radiodifusão, e de permitir a interatividade com o usuário.

Neste padrão, são definidas interfaces em diversos pontos de referência, onde cada interface é definida por um conjunto de fluxo de informações e protocolos. A especificação também define formatos para fonte, texto, hipertexto, áudio e vídeo. Controle de acesso, filtragem de informações, notificação de modificação de recursos, informações de serviço, estão dentro do escopo de atividades do DAVIC.

HAVi é fruto de uma organização criada em 1998, por um grupo de fabricantes que inclui Grudig AG, Hitachi, Philips, Thomson, Panasonic, Sharp, Sony e Toshiba. Essa API especifica um padrão para interconexão e interoperação de dispositivos de áudio e vídeo, de modo que possam interagir entre si. O objetivo

principal é permitir que o usuário possa usar e controlar esses dispositivos remotamente e de forma familiar, usando o controle remoto ou outro meio de interação.

A especificação HAVi define um conjunto de APIs e *middleware* capaz de automaticamente detectar dispositivos na rede, coordenando as funções de vários dispositivos, a instalação de aplicações e de interface do usuário do software em cada aparelho, e garantindo a interoperabilidade entre múltiplas marcas de dispositivos. Além disso, oferece uma boa base para a construção de aplicações gráficas avançadas utilizando funções de controle remoto, *displays* e gráficos de TV. Parte desta API herda do pacote Java AWT (*Abstract Window Toolkit*).

Embora o GEM não tenha sido escolhido para compor o Ginga-J, alguns pacotes da API JavaTV foram incluídos na lista de pacotes mínimos do *middleware* brasileiro, dentre os quais se pode citar javax.tv.graphics, javax.tv.media, javax.tv.net, javax.tv.service e javax.tv.xlet, entre outros.

5.7. JavaDTV

De olho no mercado brasileiro e sabendo do interesse do governo em priorizar especificações livres de *royalties*, a Sun Microsystem criou o JavaDTV, uma alternativa ao GEM que atendia perfeitamente aos requisitos brasileiros. Assim, com a colaboração da comunidade SBTVD, a API foi definida, o que possibilitou uma especificação aberta para que ela pudesse se tornar o principal pacote do Ginga-J, em substituição ao GEM.

O JavaDTV é uma API de código aberto que provê suporte ao *middleware*, o que permite que as emissoras e os fabricantes de dispositivos estejam livres para desenvolver aplicações que permitam aos telespectadores acessar uma série de

serviços digitais interativos sem *royalties*. Outro benefício da plataforma aberta JavaDTV para Ginga-J é o fato de que ela permite produzir aparelhos de TV e/ou conversores por um custo muito mais acessível, pois o preço final do software será menor.

Além da redução dos custos, há outros fatores que impactaram positivamente na adoção dessa plataforma ao invés do GEM, como a comunidade brasileira de desenvolvedores em Java, que é uma das maiores do mundo. Por se tratar de uma linguagem madura e bem difundida, há mão-de-obra qualificada pronta para prover aplicações e novas funcionalidades para o Ginga-J.

A especificação JavaDTV consiste na API JavaDTV e na API JavaTV acrescentadas à base comum dos componentes do *Java Runtime*, incluindo o *Connected Device Configuration* (CDC), o *Foundation Profile* (FP) e o *Personal Basis Profile* (PBP). Estes componentes devem obrigatoriamente estar presentes no *Java Runtime*, seguindo a lista de pacotes mínimos já citada.

Com isso, o JavaDTV é capaz de oferecer suporte a informações de serviço, sintonização (*tuner*), transporte (MPEG *streams*), *media players*, controle das mídias de áudio, vídeo e legendas (*close caption*), API de acesso aos recursos de mídia da plataforma, acesso aos arquivos de transmissão (*broadcast*), persistência (limitada), canal de retorno, acesso aos dispositivos de rede, comunicação inter-Xlet e interface gráfica através do uso do LWUIT.

Na figura seguinte, temos toda a estrutura do JavaDTV, mostrando seus componentes e funcionalidades.

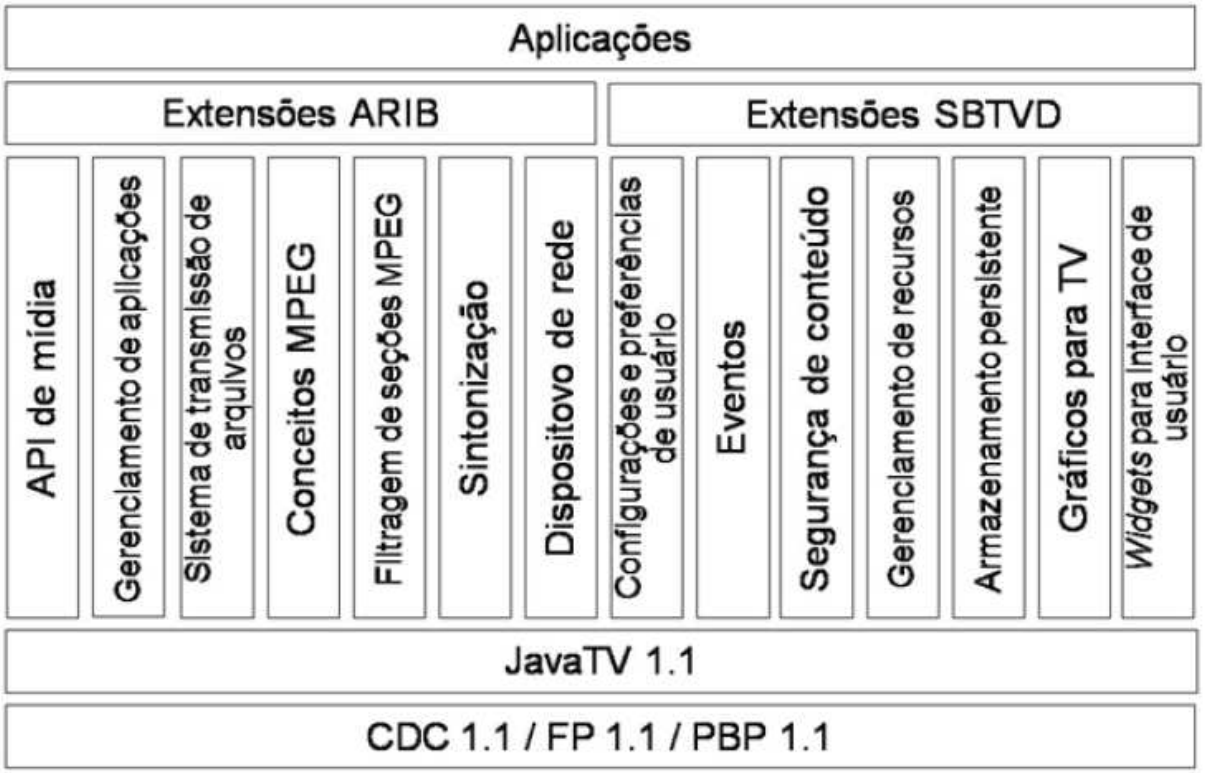

Figura 4 - Estrutura da API JavaDTV

Para suportar todas essas funcionalidades, faz-se necessário que a implementação do JavaDTV contenha uma lista mínima de pacotes. Nas tabelas que seguem abaixo, temos uma breve descrição do que cada pacote da API JavaDTV se propõe, de acordo com o descrito na norma ABNTNBR 15606-6:2010.

Tabela 1: Descrição dos pacotes da API JavaDTV

Nome do pacote	Descrição resumida do pacote
com.sun.dtv.application	Gerenciamento de aplicativos e controle do ciclo de vida
com.sun.dtv.broadcast	Acesso a arquivos de transmissão e *streams*
com.sun.dtv.broadcast.event	Acesso a eventos de transmissão
com.sun.dtv.filtering	Provê suporte para filtro de seções MPEG
com.sun.dtv.io	Estende o pacote java.io provendo direitos de acesso e propriedades aos arquivos
com.sun.dtv.locator	Esse pacote define localizadores para uso através de todo o sistema
com.sun.dtv.lwuit	Pacote widget principal contendo o “composto” componente/container similares tanto em terminologia quanto em projeto para Swing/AWT

com.sun.dtv.media	Pacote para funcionalidade básica relevante de mídia e controles congela / restaura
com.sun.dtv.net	Estende o pacote java.net com controle de dispositivo de comunicação extensiva
com.sun.dtv.platform	Provê classes que são específicas da plataforma JavaDTV, particularmente classes que são específicas para o consumidor
com.sun.dtv.resources	Provê um quadro básico de recursos escassos
com.sun.dtv.security	Pacote para a funcionalidade adicional de segurança
com.sun.dtv.service	Esse pacote provê uma interface para acessar o banco de dados SI de baixo nível
com.sun.dtv.smartcard	Pacote que provê funcionalidade *smartcard* adicional
com.sun.dtv.test	Provê um suporte extensivo de controle de teste
com.sun.dtv.transport	Provê acesso a entidades contidas em um *stream* de transporte
com.sun.dtv.tuner	Provê API para acesso a e controle de interface de redes de transmissão (ou “sintonizador”) usado para recepção de *streams* de transporte
com.sun.dtv.ui	Funcionalidade UI específica da televisão
com.sun.dtv.ui.event	Sub-pacote de evento de funcionalidade UI específica da TV

Além desses pacotes, com.sun.dtv.media e com.sun.dtv.lwuit possuem também uma lista de sub-pacotes que devem obrigatoriamente estar implementados, os quais estão descriminados em tabelas separadas.

Tabela 2: Descrição dos sub-pacotes do pacote com.sun.dtv.media

Nome do pacote	Descrição resumida do pacote
com.sun.dtv.media.audio	Pacotes para controle de língua do áudio
com.sun.dtv.media.control	Controles adicionais para obter informações sobre a mídia apresentada
com.sun.dtv.media.dripfeed	Controle para prover dado de imagem fixa para um tocador JMF, os dados ficam sob controle do aplicativo
com.sun.dtv.media.format	Controle para o formato do vídeo
com.sun.dtv.media.language	Provê a funcionalidade básica de informar-se sobre e definir a língua do áudio, das legendas e do *closed caption*
com.sun.dtv.media.text	Controle das legendas e do *closed caption*
com.sun.dtv.media.timeline	Controle da linha de tempo da mídia

Tabela 3: Descrição dos sub-pacotes do pacote com.sun.dtv.lwuit

Nome do pacote	Descrição resumida do pacote
com.sun.dtv.lwuit.animations	Todos os componentes são animáveis por animações potenciais e adicionais (não-relacionadas a um componente específico) podem ser instalados instantaneamente; transições entre formas também são tratadas como parte desse pacote
com.sun.dtv.lwuit.events	Padrão observável de receptores de evento no espírito da arquitetura despachadora de eventos do AWT1.1, todos os eventos são despachados na EDT (*Event Dispatch Thread*)
com.sun.dtv.lwuit.geom	Contém classes relacionadas aos locais geométricos e cálculos tais como retângulo e tamanho
com.sun.dtv.lwuit.layouts	Gerenciadores de layout permitem que um container organize seus componentes por um conjunto de regras que seriam adaptadas para tamanhos específicos de fonte/tela
com.sun.dtv.lwuit.list	As listas são altamente personalizáveis e servem como base para o *ComboBox* e outros componentes (tais como carrossel, etc.). Elas empregam uma abordagem MVC similar ao SWING incluindo o padrão renderizador
com.sun.dtv.lwuit.painter	Permite desenhar elementos arbitrários de gráficos provenientes de imagens simples/escalonadas/lado a lado a gradientes e praticamente todas as formas de desenho gráfico que pudermos imaginar
com.sun.dtv.lwuit.plaf	A aparência do aplicativo pode ser inteiramente personalizada por meio desse pacote, ele representa uma camada processadora que pode ser plugada separadamente em *runtime* e tematizado para prover qualquer aparência personalizada
com.sun.dtv.lwuit.util	Funções de utilitários que são ou muito específicas de um domínio ou não se “encaixam” em nenhum outro pacote

O LWUIT (*LightWeight User Interface Toolkit*) é uma biblioteca de *widgets* leves inspirados no *Swing*, mas projetados para dispositivos com restrições, tais como telefones celulares e *set-top boxes*. O LWUIT fornece componentes gráficos de alto nível e oferece a possibilidade de usar temas plugáveis, hierarquia de componente e container, tratador de eventos hierárquico, abstração dos componentes nativos e personalização através de *look and feels*. Os componentes podem ser animados e os *resource bundles* permitem o empacotamento dos recursos de maneira portável.

Uma das principais vantagens dessa biblioteca é a possibilidade de criar interfaces amigáveis sem a necessidade de desenhar tudo em telas de *canvas*.

Uma vez que este processo é bem trabalhoso e os mesmos resultados podem ser obtidos utilizando os componentes que o LWUIT disponibiliza com muito menos esforço e com o adicional de que toda a aparência pode ser alterada com a aplicação de um tema, que pode ser via programação, ou fazendo um *link* da aplicação com um arquivo de tema externo que muito se assemelha a um arquivo de estilos CSS (*Cascading Style Sheets*). A possibilidade de criação de arquivos de tema externos é uma poderosa ferramenta de customização, uma vez que o usuário da API não precisa alterar o código da aplicação para fazer alterações em relação à aparência do aplicativo.

Outra vantagem do LWUIT é que não é necessário especificar o tamanho da tela de exibição, basta especificar os componentes que irão compor a interface que a estrutura de *Layout Manager* fará a diagramação de acordo com o *layout* escolhido.

5.8. Pacotes específicos do Ginga-J

Além de todos os pacotes mencionados, o Ginga-J ainda requer a implementação de alguns pacotes específicos para o *middleware* brasileiro. Estes pacotes são br.org.sbtvd.net, br.org.sbtvd.net.si, br.org.sbtvd.net.tuning, br.org.sbtvd.bridge e br.org.sbtvd.ui, que complementam as funcionalidades existentes, provendo suporte aos requisitos do SBTVD.

6. Modelo de aplicação Ginga-J

De acordo com a norma ABNT NBR 15606-4:2010, as aplicações Ginga-J devem possuir pelo menos uma classe que implemente a interface definida em javax.microedition.xlet.Xlet, pois o modelo de aplicação Ginga-J é baseado no modelo de aplicação definido na especificação JavaDTV 1.3. A interface *Xlet* deve ser referenciada de acordo com as definições de sinalização de aplicação, pois, do contrário, a classe e a instância da aplicação podem ser ignoradas.

As aplicações Ginga-J são executadas em um ambiente orientado a serviços e mantidas por um gerenciador de aplicações global do sistema. Todo serviço é apresentado em um contexto de serviço, que poderia ser definido como seu ambiente de execução. Para aplicações Ginga-J, o contexto de serviço é representado por uma instância da classe javax.microedition.xlet.ServiceContext. Além disso, aplicações Ginga-J podem ser controladas tanto por um *zapper*, pela emissora, por outra aplicação Ginga-J usando a API "*Application Management and LifeCycle Control*" ou ainda por documentos Ginga-NCL.

6.1. Ciclo de vida

As aplicações Ginga-J são também chamadas de *Xlets* e, como já explicado, devem conter uma classe que implementa a interface javax.microedition.xlet.Xlet, o que faz com que esta classe contenha ao menos quatro métodos que são chamados pela plataforma para informar à aplicação de mudanças de ciclo de vida iminentes. Um diagrama exibindo o ciclo de vida de aplicações Ginga-J é mostrado na figura 5.

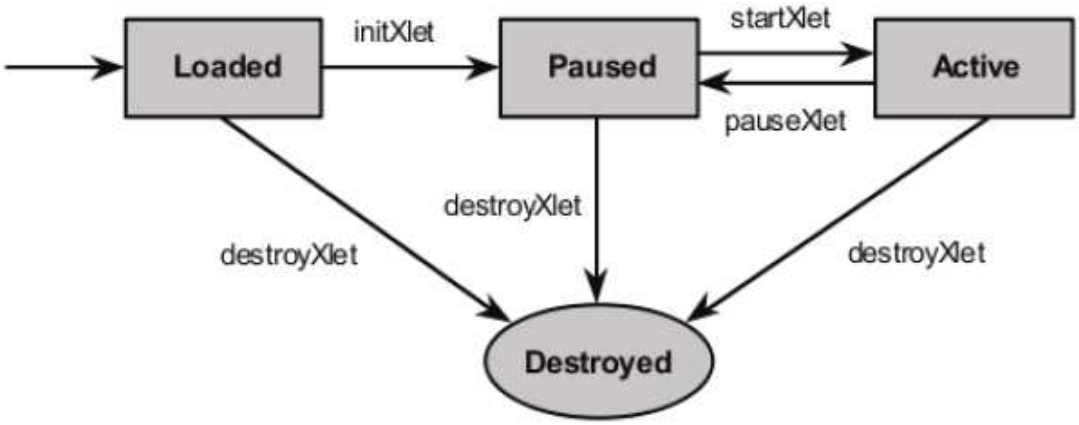

Figura 5 - Diagrama com estados do ciclo de vida de um Xlet

Assim que o *xlet* é criado, é feita a sua inicialização através do método *initXlet*(). Para inicializar a aplicação do método *initXlet*, é chamado um objeto de uma classe que instancia de javax.microedition.xlet.XletContext, a qual possui informação do contexto de execução para a aplicação, incluindo propriedades e mecanismos para notificação de mudanças de estados iniciadas pela aplicação. Após ser totalmente carregada, o seu estado de execução é passado para *Loaded*, indicando que ela está pronta para ser executada. Se acontecer alguma exceção, o estado é passado para *Destroyed* através da chamada do método *destroyXlet*().

A partir do momento em que o *xlet* está pronto para entrar em execução, ele se torna apto a receber o método *startXlet*() e, assim, passa para o estado *Active*. Chamadas ao método *pauseXlet*() irá levar novamente ao estado *Paused*, assim como o método *destroyXlet*() pode ser chamado para destruir o *xlet*. Uma instância de aplicação no estado *Paused* deve reduzir seu consumo de recursos se ela tiver a intenção de maximizar sua probabilidade de sobrevivência. Isto não significa que ela não pode manter quaisquer recursos, mas que, caso mantenha, ela deve ter uma prioridade menor ao acesso de recursos do que se estivesse no estado *Active*.

Em qualquer estado, o método *destroyXlet*() pode ser chamado para que um aviso seja enviado à aplicação sinalizando que ela está prestes a ter sua execução

terminada. A aplicação deve salvar informação de seu estado (se possível e necessário) e liberar recursos previamente utilizados o mais rápido possível. Este método possui um parâmetro booleano que indica se é o encerramento da aplicação é incondicional.

O processo de sinalização de aplicações envolve o envio periódico de uma tabela denominada Tabela de Informação de Aplicações (AIT – *Application Information Table*). O controle dinâmico do ciclo de vida é sinalizado através do campo "*application_control_code*" para a aplicação na AIT. Os códigos de controle das aplicações Ginga-J são mostrados na tabela seguinte.

Tabela 4: Códigos de controle de aplicações Ginga-J

Código	Identificador	Descrição
0x00	Não definido	Reservado para uso futuro
0x01	AUTOSTART	Aplicativos com o código de controle AUTOSTART são iniciados automaticamente, logo após a seleção do serviço que os contém
0x02	PRESENT	Aplicativos com o código de controle PRESENT não podem ser iniciados automaticamente e devem ser adicionados à lista de aplicativos disponíveis do receptor. Podem ser inicializados usando a API "*Application Management and LifeCycle Control*" (JAVADTV 1.3:2009)
0x03	DESTROY	Aplicativos com o código de controle DESTROY devem ser incondicionalmente finalizados pelo gerenciador de aplicações. Aplicativos sinalizados previamente com código de controle STORE devem ser removidos do *cache*
0x04	KILL	Aplicativos com o código de controle KILL devem ser finalizados pelo gerenciador de aplicações logo que possível. Caso seja lançada uma exceção do tipo javax.microedition.xlet.XletStateChangeException durante uma tentativa de finalização, o aplicativo deve continuar em execução. Aplicativos sinalizados previamente com código de controle STORE podem opcionalmente ser mantidos no *cache*
0x05	Não definido	Reservado para uso futuro
0x06	REMOTE	Aplicativos com código de controle REMOTE não tem seus arquivos transmitidos no *transport stream* corrente. A fonte de dados para estes aplicativos deve estar de acordo com o descritor de aplicação "*Transport Protocol Descriptor*" (ABNT NBR 15606-3:2007) Aplicativos com o código de controle REMOTE não podem ser iniciados automaticamente e devem ser adicionados à lista de aplicativos disponíveis do receptor. Podem ser inicializados usando a API "*Application Management and LifeCycle Control*" (JAVADTV 1.3:2009)
0x07	UNBOUND	Aplicativos com código de controle UNBOUND são similares a aplicativos sinalizados com PRESENT, exceto que o

		usuário decide se a aplicação pode ser armazenada para execução posterior. Caso o receptor não possua capacidade de armazenagem disponível para armazenar a aplicação ou o usuário escolha não instalar a aplicação, esta deve ser tratada como não disponível (não pode ser listada entre as aplicações disponíves). Receptores sem suporte a armazenamento de aplicações devem ignorar as aplicações sinalizadas com este código de controle
0x08	STORE	Aplicativos com código de controle STORE não podem ser iniciados automaticamente, mas indicam que técnicas de *caching* podem ser utilizadas de modo a acelerar o carregamento de seus recursos durante a inicialização. Caso a plataforma não tenha capacidade para realizar técnicas de *caching* pró-ativo ou armazenamento de dados, aplicações sinalizadas como STORE devem ser tratadas de modo idêntico a aplicações sinalizadas com PRESENT
0x09…0xFF	Não definido	Reservado para uso futuro

6.2. Inicialização e finalização de aplicações

No momento em que o usuário ou o próprio sistema seleciona um novo serviço para exibição, o gerenciador de aplicações global do sistema deve verificar as aplicações disponíveis e identificar as que devem ser iniciadas imediatamente após a seleção do serviço em questão. Caso já exista uma aplicação em execução no momento em que ocorrer a seleção do serviço que a contém, esta pode continuar em execução, caso seja sinalizada como uma aplicação válida para o novo serviço selecionado.

Após a seleção de um serviço é possível que aplicações alternativas sinalizadas sejam iniciadas pelo usuário, pelo *zapper* ou por quaisquer outras aplicações usando a API “*Application Management and LifeCycle Control*”. Isto é, o usuário pode iniciar uma aplicação após receber uma oferta de aplicações através de alguma interface de usuário. Já que tal interface é dependente de implementação, serviços sinalizados devem indicar explicitamente, caso necessitem que a aplicação seja iniciada automaticamente.

Aplicações Ginga-J podem voluntariamente terminar sua execução usando a API *Xlet* ou podem ser finalizadas pelo gerenciador de aplicações global do sistema. Além disso, uma aplicação deve ter sua execução interrompida incondicionalmente sempre que uma das seguintes condições acontecerem:

- A tabela AIT foi atualizada ou foi trocada e nesta nova versão não consta referência para a aplicação em questão;

- A tabela AIT não é mais referenciada na tabela PMT (*Program Maple Table*) do serviço que está sendo exibido.

Quando uma instância de aplicação deve ter sua execução terminada, o gerenciador de aplicações chama seu método *destroyXlet*. Caso esta instância esteja sendo interrompida devido a uma operação de seleção de serviço, sua finalização deve ser incondicional.

6.3. Armazenamento e *caching* de aplicações

O modelo de aplicação Ginga-J permite que aplicações sejam armazenadas e iniciadas a partir de memória persistente. Aplicações podem sofrer *caching* pró-ativamente ou em resposta a uma requisição de outra aplicação, sujeito a consentimento do usuário e limites de recursos da plataforma. Em ambos os casos o objetivo é melhorar a velocidade de carregamento da aplicação.

Aplicações sinalizadas com o código de controle STORE ou UNBOUND na AIT devem adicionar informação extra no arquivo MANIFEST.MF, de modo a indicar quais arquivos devem ser armazenados. Para cada arquivo que deve ser armazenado, a entrada no MANIFEST.MF para este arquivo deve conter o valor *true* para o atributo “*Persistent-Flag*” e um valor para o atributo “*File-Version*” indicando a versão do arquivo. Receptores que suportem o armazenamento de aplicações

devem verificar a versão do arquivo armazenado e atualizá-lo quando a aplicação recebida for maior. Além disso, devem ser adicionadas ao MANIFEST.MF entradas de arquivo para o *application_id* e o *organization_id*, ambos sinalizados na AIT para a aplicação em questão.

O espaço disponível para o armazenamento das aplicações pode não ser suficiente para comportar todas as aplicações sinalizadas como persistentes (STORE ou UNBOUND), sendo assim necessário escolher quais delas devem ser efetivamente persistidas. Eventualmente, também pode ser necessário remover aplicações previamente armazenadas. Nestes casos, fica a critério da implementação do fabricante do receptor a política persistência e remoção de aplicações sinalizadas como STORE. As aplicações sinalizadas como UNBOUND devem ser instaladas e removidas com autorização explícita do usuário e devem ter prioridade em relação às aplicações sinalizadas como STORE.

6.4. Transmissão de aplicações

Aplicações Ginga-J devem ser empacotadas, autenticadas e autorizadas de acordo com as definições especificadas no JAVADTV. Cada aplicação Ginga-J pode conter um ou mais arquivos JAR. O arquivo JAR principal contém os arquivos de classe da aplicação, arquivos de recurso e um manifesto que descreve a aplicação e seus requisitos. O mecanismo de assinatura de JAR permite que o JAR seja autenticado.

Se a transmissão for feita com uso do carrossel de objetos, não pode ser utilizado o empacotamento em um arquivo JAR, mas obrigatoriamente o sistema de arquivos transmitido deve estar organizado da mesma maneira. Se a aplicação for transmitida pelo canal de interatividade, é obrigatório que ela seja transmitida em

arquivos JAR.

A autenticação de aplicações Ginga-J deve estar de acordo com a Norma Brasileira vigente no momento da transmissão da aplicação. Esta norma ainda está em elaboração.

Para que uma aplicação seja considerada sinalizada em diversos serviços, todas as seguintes condições devem ser atendidas:

- a sinalização deve estar presente em uma tabela AIT em todos os serviços;

- o identificador de aplicação deve ser igual em todos os serviços;

- o descritor de protocolo de transporte deve ser igual em todos os serviços ou a aplicação está sinalizada com o código de controle UNBOUND e já se encontra persistida no receptor.

Se, além das condições descritas acima a aplicação, também estiver sinalizada com o campo *service_bound_flag* com o valor 0, esta aplicação deve ser mantida em execução entre todas as trocas de serviço, a menos que haja alguma restrição de recursos no receptor.

Com relação ao *download* de aplicações através do canal interativo, as regras de transporte de aplicação devem obrigatoriamente estar de acordo com a ABNT NBR 15606-3:2007. Como já foi dito antes, as aplicações obtidas através do canal de interatividade devem estar contidas em um arquivo JAR. O suporte à compressão no arquivo JAR é opcional.

7. Exemplo de Aplicativo *Xlet*

7.1. Metodologia

A título de demonstração da facilidade de implementação de um aplicativo do tipo *Xlet*, que é o utilizado pelo Ginga-J, foi realizada uma pequena demonstração utilizando o *XleTView*, uma aplicação Java que simula um *set-top box* no computador. Ele foi criado para executar *Xlets* criadas para o MHP, ou seja, ele utiliza a API JavaTV ao invés da JavaDTV. Outra distinção é que a API gráfica empregada não é o LWUIT, e sim a HAVI, que faz parte do JavaTV e possibilita a criação de aplicações gráficas através de classes baseadas no Java AWT.

Embora seja uma *Xlet* desenvolvida para o ambiente do MHP, ela ainda faz uso do modelo de aplicação apresentado anteriormente, apesar de ser perceptível apenas a utilização do mesmo ciclo de vida, pois não foram simuladas as partes de transmissão e recepção de *streams* ou de seleção de aplicações, somente a execução das mesmas.

O aplicativo desenvolvido apresenta uma pergunta para o usuário sobre a preferência do mesmo quanto às quatro cores apresentadas. Ao pressionar a tecla correspondente a cada cor, o nome da cor aparece sobre a área colorida com essa respectiva cor, e permanece visível até que uma outra cor seja escolhida ou alguma tecla não-mapeada seja pressionada.

No momento em que o *XleTView* é inicializado, é apresentado ao usuário um controle remoto que pode ser ativado através do teclado e do *mouse*. No menu *Applications*, é possível configurar um ou mais *Xlets* para serem executados, sendo necessário apenas criar uma referência ao *Xlet* desejado no gerenciador de aplicações do *XleTView*. Ao selecionar o aplicativo desenvolvido para execução, a

tela inicial do mesmo é mostrada, conforme a figura 6.

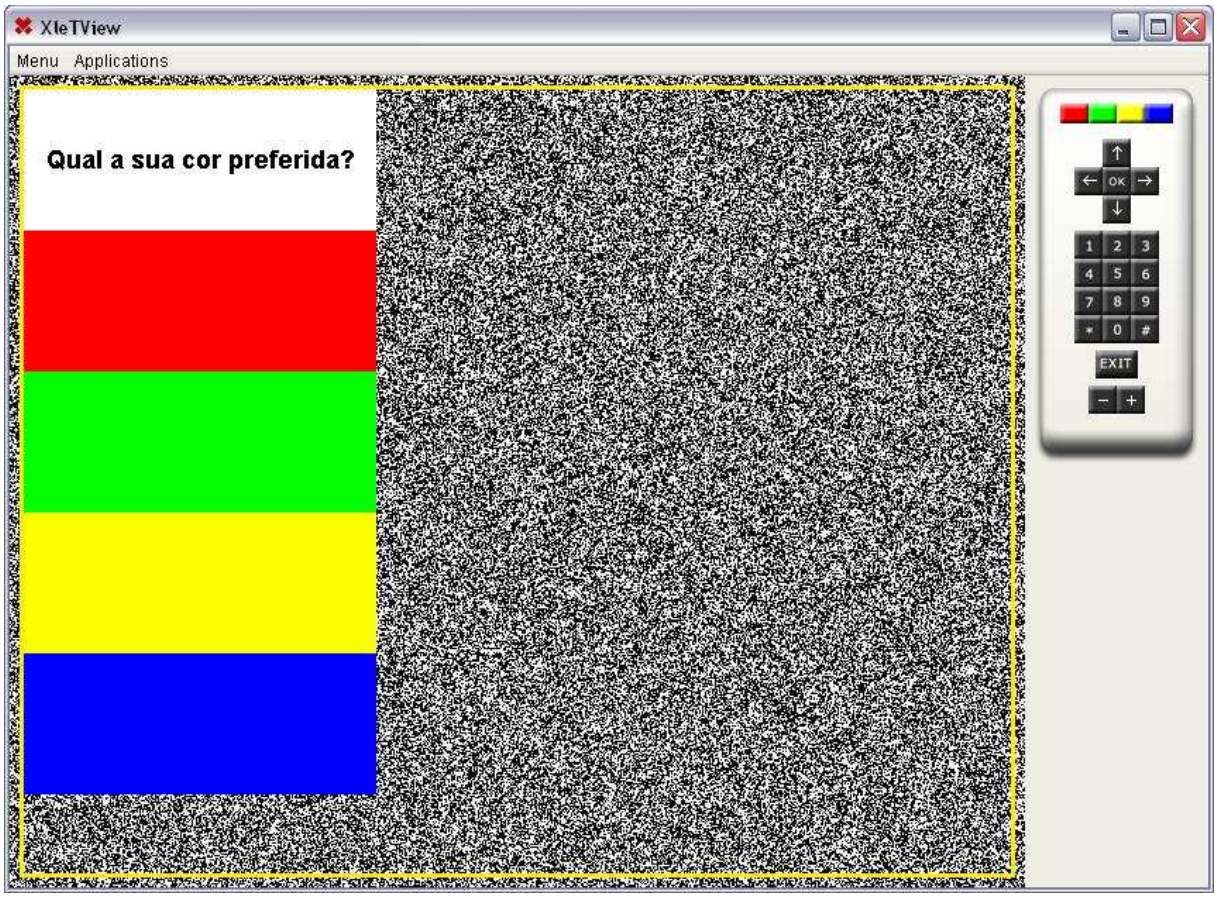

Figura 6 – Tela inicial do aplicativo desenvolvido

As teclas mapeadas para cada cor foram as seguintes:

- Número 1: Vermelho

- Número 2: Verde

- Número 3: Amarelo

- Número 4: Azul

Além delas, a tecla *Esc* também foi mapeada, representando a tecla *Exit* do controle remoto. Sua função é simplesmente encerrar a execução do aplicativo através da chamada da função *destroyXlet*,

A aplicação foi criada de forma que ela ficasse limitada à lateral da tela, permitindo ao usuário continuar assistindo ao seu programa ao mesmo tempo em que interage com a enquete. A área foi dividida em cinco, com cada parte ocupada

por uma caixa de texto. A caixa da pergunta foi configurada para ter o fundo branco, enquanto as outras quatro partes receberam, cada uma, uma cor distinta, de modo a sinalizar ao usuário as opções disponíveis. No momento em que uma das teclas mapeadas para as cores é pressionada, é inserido o texto correspondente ao nome da cor na caixa de texto correspondente a ela, enquanto as outras caixas de texto têm seus conteúdos removidos, com exceção da caixa com a pergunta.

7.2. Código fonte

O código fonte do aplicativo criado será mostrado e comentado por partes. Primeiro, tem-se os *imports*, a declaração da casse e a criação de atributos de classe, que serão manipulados nos métodos subsequentes, e do construtor, deixado vazio.

```
import java.awt.Color;
import java.awt.Font;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;

import org.havi.ui.HDefaultTextLayoutManager;
import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;
import org.havi.ui.HScreen;
import org.havi.ui.HState;
import org.havi.ui.HStaticText;

public class HelloTV implements Xlet, KeyListener {

    private XletContext context;
    private HScene scene;
    private HStaticText title, red, green, yellow, blue;

    public HelloTV() {
    }
```

Por se tratar de uma classe que implementa a interface *Xlet*, é obrigatória a criação dos métodos *initXlet*, *startXlet*, *pauseXlet* e *destroyXlet*. A seguir, temos o método *initXlet*, que é executado para inicializar a aplicação. Note que o contexto

Xlet é passado a ele nesse momento

```
    public void initXlet(XletContext xletContext) throws
XletStateChangeException {
        System.out.println("initXlet");
        context = xletContext;
    }
```

Em sequência, é criado o método *startXlet*, que define e prepara a área a ser utilizada pela aplicação e preenche as caixas de texto com seus valores iniciais.

```
    public void startXlet() throws XletStateChangeException {
        System.out.println("startXlet");
        HSceneFactory hsceneFactory = HSceneFactory.getInstance();
        scene =
hsceneFactory.getFullScreenScene(HScreen.getDefaultHScreen().getDefaultHGra
phicsDevice());
        scene.setSize(260, 510);
        scene.setLayout(null);
        scene.addKeyListener(this);

        title = new HStaticText("Qual a sua cor preferida?", 10, 10, 250,
100, new Font("Tiresias", Font.BOLD, 20), Color.black, Color.white, new
HDefaultTextLayoutManager());
        scene.add(title);

        red = new HStaticText("", 10, 110, 250, 100, new Font("Tiresias",
Font.PLAIN, 20), Color.lightGray, Color.red, new
HDefaultTextLayoutManager());
        scene.add(red);

        green = new HStaticText("", 10, 210, 250, 100, new Font("Tiresias",
Font.PLAIN, 20), Color.darkGray, Color.green, new
HDefaultTextLayoutManager());
        scene.add(green);

        yellow = new HStaticText("", 10, 310, 250, 100, new
Font("Tiresias", Font.PLAIN, 20), Color.darkGray, Color.yellow, new
HDefaultTextLayoutManager());
        scene.add(yellow);

        blue = new HStaticText("", 10, 410, 250, 100, new Font("Tiresias",
Font.PLAIN, 20), Color.lightGray, Color.blue, new
HDefaultTextLayoutManager());
        scene.add(blue);

        scene.setVisible(true);
            scene.requestFocus();
    }
```

O método *pauseXlet*, nesta aplicação, foi deixado em branco, pois não será trabalhado com a transição para o estado *PAUSED*. Já o método *destroyXlet* foi implementado e ele se encarrega de remover da tela todos os componentes do *Xlet* e encerrar o aplicativo.

```
    public void pauseXlet() {
  }

    public void destroyXlet(boolean flag) throws XletStateChangeException {
        System.out.println("destroyXlet");
        if (scene != null) {
            scene.setVisible(false);
            scene.removeAll();
            scene = null;
        }
        context.notifyDestroyed();
    }
```

Uma vez que a classe implementa também a interface *KeyListener*, é necessário implementar os métodos *keyTyped*, *keyPressed* e *keyReleased*. Como o evento que se quer tratar é apenas o relacionado ao pressionamento das teclas, os métodos *keyTyped* e *keyRelease* foram deixados em branco.

```
    public void keyTyped(KeyEvent e) {
    }

    public void keyReleased(KeyEvent e) {
    }
```

Por fim, foi implementado o método *keyPressed*, responsável por capturar os eventos gerados pelo pressionamento das teclas e, de acordo com a tecla pressionada, inserir ou remover os nomes das cores de suas respectivas caixas de texto.

```
    public void keyPressed(KeyEvent e) {

      int code = e.getKeyCode();

      if(code == KeyEvent.VK_1)
            red.setTextContent("Vermelho", HState.ALL_STATES);
      else
            red.setTextContent("", HState.ALL_STATES);
      red.repaint();
      if(code == KeyEvent.VK_2)
            green.setTextContent("Verde", HState.ALL_STATES);
      else
            green.setTextContent("", HState.ALL_STATES);
      green.repaint();
      if(code == KeyEvent.VK_3)
            yellow.setTextContent("Amarelo", HState.ALL_STATES);
      else
            yellow.setTextContent("", HState.ALL_STATES);
      yellow.repaint();
      if(code == KeyEvent.VK_4)
            blue.setTextContent("Azul", HState.ALL_STATES);
```

```
        else
              blue.setTextContent("", HState.ALL_STATES);
        blue.repaint();

        if(code == KeyEvent.VK_ESCAPE) {
              try {
                    destroyXlet(true);
              } catch (XletStateChangeException exc) {
                    exc.printStackTrace();
              }
        }
    }
}
```

De acordo com a lógica empregada, se uma tecla não mapeada for pressionada, os nomes de todas as cores se apagarão. Se a tecla *Esc* for pressionada, o método *destroyXlet* é chamado e a aplicação se encerra.

8. Conclusão

O trabalho realizado teve por objetivo expor o histórico e as principais características dos mais difundidos sistemas de TV digital em utilização no mundo e analisar o sistema brasileiro, o SBTVD, e seu *middleware*, destacando o ambiente procedural, o Ginga-J, assim como explicar o seu modelo de aplicação e exemplificá-lo com a implementação de um aplicativo básico.

Foi apresentado um resumo da história da TV, desde a criação da TV analógica, passando pela TV em cores e em alta definição, até chegar na TV digital, foco das atenções de todo o mundo principalmente na última década. Foi relatado também o processo que levou a vários países criarem seus próprios sistemas de TV digital, segundo a necessidade e a realidade de cada um.

As características mais significantes dos principais padrões de TV digital foram expostas, mostrando as vantagens e desvantagens de cada sistema e o que levou à escolha do padrão japonês ao invés dos padrões americano e europeu. Diferenciais como maior imunidade à interferência, possibilidade de integração de diferentes serviços digitais e custo mais baixo dos receptores, dentre outros, fizeram com que o sistema escolhido como base para o sistema brasileiro fosse o ISDB-T, dando origem ao sistema nipo-brasileiro ISDB-Tb ou SBTVD.

Realizando um estudo mais aprofundado do *middleware* criado para o sistema brasileiro, foram mostrados os componentes do Ginga, explicando a funcionalidade de cada um. O Ginga-CC, por exemplo, provê o suporte básico para os dois componentes principais, o Ginga-NCL e o Ginga-J, além de abstrair a camada de *hardware*. Enquanto o Ginga-NCL define o ambiente declarativo do *middleware*, o Ginga-J, foco deste trabalho, define o ambiente procedural.

No capítulo 5, foi realizada uma breve apresentação dos principais

frameworks e APIs que compõem o Ginga-J, fornecendo uma visão melhor das tecnologias envolvidas e das características deste subsistema. Assim, formou-se a base para o capítulo 6, que explica mais detalhadamente as particularidades do modelo de aplicação do Ginga-J.

Um *Xlet*, outra denominação para o aplicativo do Ginga-J, foi implementado para exemplificar o modelo de aplicação apresentado. Dessa forma, demonstrou-se a facilidade do desenvolvimento de aplicativos para o *middleware* brasileiro, tornando claro que a TV digital possui o potencial de ser mais atrativa e diversificada com a implementação de uma grande variedade de aplicações, especialmente se for levada em consideração a grande comunidade Java que o Brasil possui.

Portanto, é necessário mais empenho das empresas para que a TV digital do Brasil disponibilize para a população todos os recursos que o SBTVD suporta, principalmente no investimento em desenvolvimento de aplicações, seja para entretenimento, como jogos, ou para informação, a exemplo de notícias, entre tantas outras possibilidades existentes. Somente através da exploração de todo potencial da TV digital é que o sistema irá crescer e se popularizar, fazendo com que a diferença dela para a TV analógica não se resuma apenas à exibição de áudio e vídeo com melhor qualidade.

8.1. Desafios

Apesar dos esforços da comunidade empenhada no desenvolvimento do Ginga-J, ainda há muitas dificuldades no desenvolvimento de aplicações para este subsistema. A especificação do JavaDTV, por exemplo, foi disponibilizada a alguns meses, mas ainda não é possível encontrar uma implementação desta API cujo código fonte esteja disponível.

Algumas máquinas virtuais já se encontram disponíveis para download, cada qual procurando apresentar um ambiente com o Ginga-NCL e/ou o Ginga-J. No entanto, são ambientes pesados e que nem sempre funcionam adequadamente. Infelizmente ainda não foram criados simples visualizadores de Xlets para o Ginga-J, tais como uma versão do *XleTView* para o middleware brasileiro. Este aplicativo, voltado para o MHP, é bem leve e fácil de usar, facilitando o desenvolvimento de aplicações para o middleware europeu.

Pode-se notar também a ausência de ferramentas para o Ginga-J equivalentes ao *Composer* do Ginga-NCL, que provê um ambiente de desenvolvimento para aplicativos escritos em NCL-LUA. Isso dificulta a disseminação do conhecimento a respeito da implementação de *Xlets*, pois uma ferramenta como essa ampliaria o interesse das pessoas no ambiente Ginga-J e apresentaria de forma mais explícita as possibilidades que os desenvolvedores têm a mão ao utilizar esse subsistema.

REFERÊNCIAS

ALVES, Kellyanne C.; FEITOSA, Deisy F. **TV digital: surgimento e perspectivas**. In: Intercom Junior - Jornada de Iniciação Científica em Comunicação, do XXIX Congresso Brasileiro de Ciência da Comunicação. Disponível em: < http://www.intercom.org.br/papers/nacionais/2006/resumos/R2111-1.pdf> Acesso em: 01 jun. 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS**. NBR 15606-4**: Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais. Rio de Janeiro, 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 15606-6**: Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 6: JavaDTV 1.3. Rio de Janeiro, 2010.

BARBOSA, Jorge F. R. **Padrões de TV Digital – Overview**. Disponível em: <http://jorgefelliphe.spaces.live.com/blog/cns!43B56C1CFDA6DC17!307.entry> Acesso em: 07 ago. 2010.

BRASIL. Decreto nº 4.901, de 26 de novembro de 2003. Institui o Sistema Brasileiro de Televisão Digital - SBTVD, e dá outras providências. **Diário Oficial da União**. Brasília, DF, 27 nov. 2003. Disponível em: <http://www.planalto.gov.br/CCIVIL/decreto/2003/D4901.htm>. Acesso em: 09 jul. 2010

BRASIL. Decreto nº 5.820, de 29 de junho de 2006. Dispõe sobre a implantação do SBTVD-T, estabelece diretrizes para a transição do sistema de transmissão analógica para o sistema de transmissão digital do serviço de radiodifusão de sons e imagens e do serviço de retransmissão de televisão, e dá outras providências. **Diário Oficial da União**. Brasília, DF, 30 jun. 2006a. Disponível em: < http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2006/Decreto/D5820.htm>. Acesso em: 09 jul. 2010.

BOLAÑO, César Ricardo Siqueira; VIEIRA, Vinícius Rodrigues. TV digital no Brasil e no Mundo: estado da arte. **Revista de Economia Política de lãs Tecnologías de La Información y Comunicación**, v. VI, n. 2, may. – ago. 2004. Disponível em: <http://www2.eptic.com.br/sgw/data/bib/periodicos/41dce6abbc81528e172f2df21e9bbffc.pdf>. Acesso em: 20 jun. 2010.

COELHO Jr., Hélio. **Sistema de Transmissão no Padrão Brasileiro de TV Digital**. Niteroi, 2008. Disponível em: <http://www.midiacom.uff.br/~debora/fsmm/trab-2008-2/transmissao.pdf> Acesso em: 15 de abr. 2010.

DAMASCENO, Jean R. **Middleware Ginga**. Niteroi, 2008. Disponível em: <http://www.midiacom.uff.br/~debora/fsmm/trab-2008-2/middleware.pdf> Acesso em: 15 de abr. 2010.

FÓRUM SBTVD. Sala de Imprensa. **Avanço a passos largos**. Disponível em: <http://www.forumsbtvd.org.br/materias.asp?id=539> Acesso em: 05 dez. 2010.

ORACLE. **Java ME Technology – JavaTV API**. Disponível em: <http://www.oracle.com/technetwork/java/javame/tech/javatv-136131.html> Acesso em: 10 dez. 2010.

PAES, Alexsandro; SAADE, Débora C. M.; ANTONIAZZI, Renato H. **Padrões de Middleware para TV Digital.** Disponível em: <http://www.teleco.com.br/tutoriais/tutorialtvdpadrao/> Acesso em: 20 jun. 2010.

SVEDEN, Martin, **XleTView**. Informações e *download* do emulador para visualização de *Xlets* do MHP. Disponível em <http://www.xletview.org/> Acesso em 12 dez. 2010.