UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

APLICATIVO PARA TV DIGITAL INTERATIVA DE ACESSO AO TWITTER

MARCOS ERNANI MARTINI

BLUMENAU
2010

2010/2-21

MARCOS ERNANI MARTINI

APLICATIVO PARA TV DIGITAL INTERATIVA DE ACESSO AO TWITTER

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Francisco Adell Péricas, Orientador

BLUMENAU
2010

2010/1-21

APLICATIVO PARA TV DIGITAL INTERATIVA DE ACESSO AO TWITTER

Por

MARCOS ERNANI MARTINI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, Ms. – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, M. Sc. – FURB
Membro:	Prof. Sérgio Stringari, Ms. – FURB

Blumenau, 13 de dezembro de 2010

Dedico este trabalho a todos os colegas que estiveram comigo nesta trajetória e com os quais pude desfrutar de momentos de descontração, companheirismo, aprendizado e suporte para a conclusão de nossa graduação.

AGRADECIMENTOS

A Deus, pela vida.

À minha família, por entender as ausências em encontros familiares e a minha companheira Simoni, que esteve comigo desde os primeiros semestres me dando força, motivação e alegrias.

À minha empresa Supero Tecnologia, que me auxiliou e incentivou durante todo o processo da graduação.

Aos meus amigos, pelos empurrões, cobranças e brincadeiras.

Ao meu orientador, Péricas, por ter sido pontual nas cobranças e ditado o ritmo para a conclusão deste.

Você nunca sabe que resultados virão de sua ação. Mas se você não fizer nada, não existirão resultados.

Mahatma Gandhi

RESUMO

A TV Digital é uma realidade e a partir da sua disseminação ocorre a possibilidade de se fornecer acesso a muitos recursos. Quando associada a um método de transmissão, a TV acrescenta um conceito novo, a Interatividade. Considerando esta possibilidade, o presente trabalho apresenta o desenvolvimento de uma aplicação para TV Digital Interativa, compatível com o ambiente Ginga-J do Sistema Brasileiro de TV Digital, que disponibiliza ao usuário o acesso a uma das mais utilizadas redes sociais, o Twitter. Com a aplicação, o usuário pode assistir a seu programa de televisão favorito e acompanhar as atualizações de seus contatos em tempo real.

Palavras-chave: Televisão digital interativa. Ginga-J. Twitter.

ABSTRACT

Digital TV is a reality and due to its dissemination it is possible to provide access to many resources. When connected to a transmission method, the TV adds a new concept, the Interactivity. Considering this possibility, this work prevents the development of an application for Interactive Digital TV, consistent with the Ginga-J of the Brazilian System of Digital TV, that provides to the user access to one of the most used social networking, the Twitter. With this application the user can watch its favorite television program and follow the updates of its contacts at real time.

Key-words: Interactive digital television. Ginga-J. Twitter.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ABNT – Associação Brasileira de Normas Técnicas

ASF - *Atom Syndication Format*

ATSC - *Advanced Television System Committee*

API – *Application Programming Interface*

ARIB - *Association of Radio Industries and Businesses*

BSD - *Berkeley Software Distribution*

CSS - *Cascading Style Sheets*

CPU - *Central Processing Unit*

DAVIC - *Digital Audio VIdeo Council*

DVB - *Digital Video Broadcasting*

GEM - *Globally Executable MHP*

Ginga-CC - Ginga *Commom-Core*

GTI - *Global Information Tracker*

GPL - *General Public License*

HAVi - *Home Audio Video interoperability*

HTTP - *HyperText Transfer Protocol*

IDE - *Integrated Development Environment*

ISDB - *Integrated Services Digital Broadcasting*

JAR - *Java ARchive*

JPEG - *Joint Photographic Experts Group*

JMF - *Java Media Framework*

JSON - *JavaScript Object Notation*

JVM - *Java Virtual Machine*

KB - *KiloBytes*

LAVID - Laboratório de Aplicações de VÍdeo Digital

LWUIT - *LightsWeight User Interface Toolkit*

MB - *MegaBytes*

MHP - *Multimedia Home Plataform*

MOSTvd - *MiddlewareOpenSource* para TV Digital

NCL - *Nested Context Language*

OAuth - *Open Autorization*

PDA - *Personal Digital Assistants*

PNG - *Portable Network Graphics*

PUCRJ - Pontifícia Universidade Católica do Rio de Janeiro

REST – REpresentional State Transfer

RF - Requisito Funcional

RNF - Requisito Não Funcional

RSSa - *Really Simple Syndication*

RSSb - *Rich Site Summary*

SBTVD - Sistema Brasileiro de TV Digital

SMS - *Short Message System*

TV - Televisão

TVDI - TV Digital Interativa

UFPB - Universidade Federal da ParaíBa

UML - *Unified Modeling Language*

XHTML - *eXtensible HyperText Markup Language*

W3C - *World Wide Web Consortium*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O advento da TeleVisão (TV) digital causará um grande impacto na forma como se conhece e se interage com a televisão. Em relação ao sistema analógico, este novo sistema não será apenas responsável pela entrega ao telespectador de um sinal com maior qualidade e resolução, mas também por um novo conceito, que é o da interatividade. Até então a experiência vivida pelo telespectador, assim como o próprio nome diz, era de ser um mero espectador do que era transmitido pelas emissoras, sendo que a única ação que ele poderia tomar era a troca de canal caso a programação não lhe convinha. Com a TV digital o fluxo de envio das informações prevê, além do áudio e vídeo, um canal de dados, onde aplicativos e outras informações serão transmitidos. Aplicativos estes que podem ter as mais variadas finalidades, como por exemplo, um guia de programação de canais, um formulário para compras de determinado produto, a tabela de um campeonato de futebol do jogo que está sendo assistido, entre outros.

Com a disseminação da TV digital interativa (TVDI) e a popularização das redes sociais, ocorre a possibilidade de se prover o acesso a estas redes não apenas pelos computadores, mas também através de aplicativos executados pelo televisor ou qualquer aparelho compatível com um sistema de TV digital. Dentre as redes destaca-se o Twitter, um serviço muito popular com intuito de servir como um *microblog*, possibilitando aos usuários conectados enviar e ler atualizações pessoais de outros contatos.

Segundo Soares e Barbosa (2009, p. 3), para tornar os aplicativos independentes da plataforma de hardware e software de um fabricante de receptor e para um melhor suporte aos aplicativos voltadas para a TV, uma nova camada é acrescentada aos padrões de referência de um sistema de TV digital, denominada de *middleware*.

Durante a definição do sistema de TV digital que seria adotado no Brasil, optou-se pelo aproveitamento das experiência vividas nos outros sistemas em uso, e utilizou-se como base o sistema japonês de TV digital. Porém, para o sistema brasileiro foi criada uma nova especificação de *middleware*, denominada Ginga, que segue dois paradigmas de programação, o declarativo conhecido como Ginga-NCL e o procedural chamado de Ginga-J.

Diante do exposto, foi desenvolvida uma aplicação para TV digital interativa, compatível com o ambiente procedural do Ginga, o Ginga-J, e que disponibilize ao usuário,

através da Twitter API[1], as principais funcionalidades do Twitter. Com esta aplicação o usuário poderá assistir a seu programa de televisão favorito e acompanhar as atualizações de seus contatos em tempo real, além de poder enviar, por exemplo, qual o canal que está assistindo para estar "sintonizado" com seus contatos.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi o desenvolvimento de um aplicativo *Xlet* para TVDI de acesso ao Twitter, em conformidade com as regras e normas da especificação Ginga-J, sendo desta forma portável a qualquer aparelho receptor compatível com o Ginga-J.

Os objetivos específicos do trabalho são:

a) efetuar *login* no Twitter;
b) disponibilizar a lista de atualizações dos contatos ligados à rede do usuário;
c) interagir e detalhar atualizações selecionadas;
d) enviar atualização por parte do usuário.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho estrutura-se em quatro capítulos. O capítulo 1 apresenta a introdução aos temas propostos, os objetivos e a estrutura do trabalho.

No capítulo 2 descreve-se a fundamentação teórica, base para o desenvolvimento e entendimento deste trabalho, através de conceitos sobre TV digital, TVDI e aplicativos. Apresenta-se também um aprofundamento sobre o sistema de TV digital adotado pelo Brasil, e a arquitetura de seu *middleware* Ginga. Ainda neste, vislumbra-se a rede social Twitter bem como sua API.

O capítulo 3 descreve todo o ciclo de especificação, desenvolvimento e utilização do aplicativo proposto tendo como base a análise orientada a objeto, utilizando a *Unified Modeling Language* (UML).

[1] Uma *Application Programming Interface* (API) é um conjunto de instruções e padrões de programação estabelecidos por determinado software para prover acesso as suas funcionalidades a partir de outro aplicativo.

E por fim, o capítulo 4 apresenta as conclusões obtidas e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes são detalhadas características de um sistema de TV digital, bem como explanados os conceitos em torno da TVDI e seus aplicativos. Também são levantados detalhes sobre o sistema brasileiro de TV digital, os conceitos gerais e a arquitetura do seu *middleware,* conhecido como Ginga, bem como um breve levantamento das implementações de referência Ginga-J disponíveis para o desenvolvimento do protótipo. Além disso, são explanados conceitos e informações sobre as redes sociais, o sistema Twitter e sua biblioteca, a Twitter API. Por fim, são apresentados os trabalhos correlatos ao tema proposto.

2.1 TELEVISÃO DIGITAL

A TV digital é fruto de um século de aceleradas evoluções tecnológicas, onde métodos cada vez mais ágeis e eficazes de codificação, compactação, transmissão e recepção de dados são empregados para difundir conteúdos digitais numa velocidade inimaginável nos tempos das primeiras invenções televisivas. Porém, para o correto aproveitamento de toda essa inovação, faz-se necessário a definição e especificação estruturada de todos os agentes envolvidos, para que todos possam interagir ordenadamente e entregar ao agente final (telespectador) integridade e a melhor qualidade do sinal enviado.

Cruz, Moreno e Soares (2008, p. 1) afirmam que em um sistema de TV digital, de forma simplificada, existem dois conjuntos de atores: os provedores de conteúdo e os telespectadores. Os primeiros responsabilizam-se pelo o desenvolvimento dos programas televisivos, ou conteúdos, que deverão ser entregues aos telespectadores. Apesar deste conteúdo já ser produzido digitalmente, ele ainda é, em muitos lugares, transmitido analogicamente dos provedores para os telespectadores. A mudança da transmissão analógica para digital já foi realizada em alguns países, e está sendo implantada no Brasil. Isto muda a forma como a TV é feita, abrindo assim, novos horizontes, tanto para os desenvolvedores quanto para os telespectadores. Define-se então que o sistema de TV digital é um típico sistema cliente/servidor. O servidor é um ambiente de uma radiodifusora ou provedor de conteúdo e o ambiente do cliente, o usuário telespectador.

Ainda em Cruz, Moreno e Soares (2008, p. 2) relacionam-se as grandes vantagens da

TV digital em relação à analógica, sendo a primeira delas, e a melhor percebida pelo telespectador, a qualidade da imagem. Graças à natureza discreta do conteúdo digital, realiza-se uma recuperação muito precisa a partir do sinal recebido. Sendo assim, o conteúdo recebido é o mais próximo possível daquele que foi transmitido. De forma similar, a qualidade do áudio também sofre melhoras e mais canais de áudio também podem ser oferecidos. Isso possibilita, por exemplo, a escolha de línguas diferentes durante a exibição de um determinado programa. Outra vantagem da TV digital é permitir que o conteúdo transmitido seja facilmente armazenado e com a mesma qualidade do sinal transmitido. Assim, o telespectador poderia, por exemplo, pausar ou solicitar um *replay*, mesmo em transmissões feitas ao vivo.

De acordo com Soares e Barbosa (2009, p. 6), o impacto da TV digital é muito mais significativo do que a simples troca de um sistema de transmissão ou da melhora da qualidade de imagem e som transmitidos. Mais do que isso, este novo sistema permite uma flexibilidade inatingível com a difusão analógica, sendo um componente importante desta flexibilidade a possibilidade de expandir as funções do sistema com aplicativos interativos, desde que construídos sobre a base de um sistema padrão de referência.

A utilização destes aplicativos interativos na TV digital gera um desdobramento na área de TV digital, que será melhor contextualizado na seção seguinte.

2.1.1 TV DIGITAL INTERATIVA (TVDI)

Um dos grandes diferenciais, senão o maior, que a transmissão digital propicia é a interatividade. Surge uma nova frente de estudos que pode ser chamada de TVDI. Com ela o telespectador deixará de lado a figura de mero espectador, permitindo além de receber o sinal, interagir com o conteúdo, selecionando grades de programação, saldos bancários, respostas de enquetes, ou seja, explorando, transformando e construindo o conteúdo à vontade enquanto utiliza-o. De acordo com Waisman (2002), ao enviar um dado para outras pessoas, o telespectador atende a uma necessidade inerente ao ser humano que é a de participar, se pronunciar e se sentir mais inserido em seu contexto social.

Santos (2002 apud STANDKE, 2002) relata que desde muito tempo as emissoras têm buscado interagir com o telespectador. Alguns exemplos são os programas em que decide-se o final ou elimina-se um participante, onde perguntas são respondidas pela internet ou por frases que aparecem nos programas. O autor enfatiza, no entanto, que nada ainda se

apresentara no país tão interativo quanto o programa "Hugo", da TV Gazeta (Figura 1), que congestionava as linhas telefônicas na década de 90. Tratava-se de um jogo em que o telespectador participava utilizando os botões do telefone para mover o personagem Hugo para cima, esquerda ou direita, desviando assim os obstáculos. Pode-se afirmar, que este método foi uma das primeiras experiências com interatividade na televisão aberta brasileira.

Fonte: Opreh (2010).
Figura 1 – Programa “Hugo” da TV Gazeta

Brackmann (2010, p. 37) afirma que é fácil perceber a grande evolução que está ocorrendo com o principal meio de comunicação e informação do Brasil, a Televisão. Com a interatividade o telespectador poderá realizar uma imersão maior na programação de TV, ou seja, poderá interar-se mais sobre o assunto em discussão. Rompem-se barreiras da programação passiva, tornando o telespectador parte da mesma.

Para prover-se esta interatividade utilizando-se da nova estrutura, ou seja, da TV digital, surgem programas computacionais embarcados no dispositivo receptor ou enviados juntamente do sinal com o áudio e o vídeo principais. Através de uma capacidade computacional significativa no dispositivo receptor, é possível o processamento destes aplicativos (SOARES; BARBOSA, 2009).

2.1.2 Aplicativos para TVDI

Os aplicativos de TVDI podem ser divididos em três grandes áreas: os dependentes do conteúdo televisivo, os independentes do conteúdo televisivo, e os residentes ou embarcados no receptor digital, também conhecido como *Set-top Box*.

Pode-se chamar o aplicativo de dependente do conteúdo televisivo quando o mesmo está relacionado com o conteúdo sendo exibido no canal de escolha do telespectador, como

por exemplo, uma chuteira, meias ou o uniforme completo das equipes que estejam disputando uma partida futebol. Ainda no mesmo exemplo, o telespectador poderia em tempo real acompanhar estatísticas e classificação do campeonato (figura 2), ou ainda ver os próximos jogos de seu time. Neste ramo de aplicativos, vê-se claramente que o papel principal no desenvolvimento está relacionado com as difusoras dos conteúdos televisivos que enviam tanto o sinal principal do conteúdo, quanto os dados necessários para a execução da aplicação.

Fonte: Damasio (2007).

Figura 2 – Exemplo de um aplicativo dependente do conteúdo televisivo

Já os aplicativos independentes do conteúdo não possuem relação alguma com o conteúdo sendo transmitido, como por exemplo, uma pesquisa de satisfação quanto ao uso da TV digital, ou ainda a listagem de notícias de uma cidade escolhida pelo telespectador, e que independente do canal, é atualizada automaticamente. A geração deste conteúdo pode ser desvinculada com mais facilidade da difusora do conteúdo televisivo, sendo desenvolvido até mesmo por uma empresa terceirizada, ficando a cargo da difusora apenas a divulgação da aplicação juntamente do restante do conteúdo do canal (vídeo, canais de áudio, legendas, etc.).

O último grupo engloba os aplicativos embarcados nos *Set-top Boxes* ou receptores compatíveis, que são completamente independentes de conteúdo transmitido e podem ser executados inclusive com a ausência do sinal. Entre os exemplos mais comuns, encontram-se jogos, aplicações para configurações do próprio receptor, ou ainda um aplicativo para pesquisa de satisfação do cliente com relação ao aparelho, entre outros. Este grupo possibilita aos fabricantes diferenciar-se dos demais e abrir vantagens estratégicas perante seus concorrentes, ao disponibilizar gratuitamente funcionalidades extras para a conquista dos clientes.

2.2 SISTEMA BRASILEIRO DE TV DIGITAL

Pode-se definir como um sistema de TV digital um conjunto de padrões tecnológicos, que por sua vez são definidos como um conjunto de especificações previamente estudado, testado, e adaptado às necessidades da região que será empregado. O desenvolvimento de um novo padrão é um processo muito oneroso, e que geralmente emprega muitos anos de estudo.

Durante a definição do Sistema Brasileiro de TV Digital (SBTVD) concluiu-se inviável a criação de um novo padrão, principalmente pelo alto custo e pelo tempo que seria dedicado ao estudo do mesmo. Então a solução encontrada foi a análise dos padrões existentes na época: o *Advanced Television System Committee* (ATSC) empregado nos Estados Unidos, o *Digital Video Broadcasting* (DVB) que é utilizado na Europa e, por fim, o *Integrated Services Digital Broadcasting* (ISDB) utilizado no Japão. Cada qual possue particularidades específicas, indo desde os métodos de transmissão, até a definição de seu *middleware*[2], inclusive da linguagem e bibliotecas que compõem seu *middleware*.

Brackmann (2010, p. 39) afirma que no momento do estudo concluiu-se que: o ATSC possuía alta definição da transmissão, porém não demonstrava preocupação com a mobilidade e a interatividade; o DVB possibilitava múltipla programação, interatividade e novos serviços; e o ISDB possuía alta definição e a mobilidade desejada para a realidade brasileira. As pesquisas e necessidades brasileiras resultaram na definição de um novo padrão, o SBTVD, que segue como modelo o padrão japonês (ISDB), porém com algumas alterações em sua arquitetura.

Segundo Costa e Melo Jr. (2009, p. 2), o SBTVD possui características diferentes em relação aos outros padrões não só na camada de *software*, e apesar de ser baseado no padrão japonês, o padrão brasileiro incorporou novas técnicas a esse padrão que já são características de uma evolução do padrão europeu. Assim, pode-se considerar que SBTVD é uma evolução do padrão japonês aliado ao padrão europeu, além de contar com melhorias estudas pelos próprios pesquisadores brasileiros.

Dentre as grandes inovações no SBTVD de acordo com Costa e Melo Jr. (2009, p.2), destaca-se o novo *middleware* onde existem as maiores inovações que possibilitam uma especificação livre, interação com dispositivos móveis, multiusuário, multidispositivo e *royalty-free*.

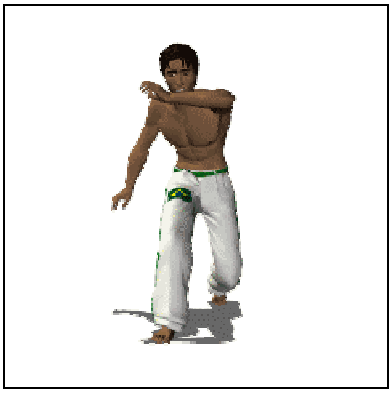
---

[2] Segundo Sampaio (2008, p.17) os *middlewares* são APIs genéricas que criam uma camada de abstração que permitem que a mesma implementação de um programa seja executada em qualquer aparelho receptor.

2.2.1 Ginga – o middleware do SBTVD

A criação de um *middleware* próprio para o padrão brasileiro possibilitou ao Brasil um alto desenvolvimento da indústria de software, tendo em vista que em outros países os custos são mais elevados devido aos valores pagos em *royalties* para bibliotecas proprietárias utilizadas nos *middlewares* destes.

O *middleware* brasileiro foi batizado de Ginga (Figura 3), e de acordo com o site Ginga (2008), homenageia a qualidade, quase que indefinível, presente em todos os brasileiros, de movimento e de atitude e que é evidente em tudo o que é feito pelos mesmos. A forma como caminham, falam, dançam e como se relacionam com tudo em suas vidas. E em reconhecimento à cultura, arte e contínua luta por liberdade e igualdade do povo brasileiro.



Fonte: Ginga (2008).

Figura 3 – A ginga é um movimento da capoeira, e dá origem ao nome do *middleware*

Sua especificação e desenvolvimento foi efetuado pela Pontifícia Universidade Católica do Rio de Janeiro (PUCRJ) em conjunto com a Universidade Federal da ParaíBa (UFPB). Como o mesmo se baseia numa implementação mais acessível, devido a sua natureza livre e tecnologia 100% brasileira, fundamenta a importância de - para um país como o Brasil, onde muitos estão desprovidos de recursos e acessibilidade - adotar um padrão próprio que se adéqua as suas reais necessidades.

Ainda em Ginga (2008), o Ginga leva em consideração a importância da TV, presente na totalidade dos lares brasileiros, como um meio complementar a inclusão social/digital. Suporta o que é chamado de “aplicações de inclusão”, tais como o *T-Government*, *T-Health* e *T-Learning*. Tais serviços podem ser exemplificados com a disponibilização de acesso aos sistemas do governo (imposto de renda, visualização de tributos), serviços de saúde (agendamento de consultas, visualização de exames) e materiais educativos de suporte ao

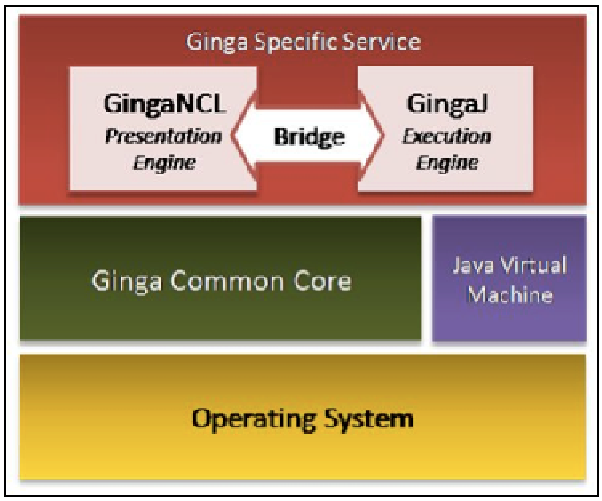
ensino a distância. E com a especificação aberta, de fácil aprendizagem e livre de *royalties*, permitem que todos os brasileiros produzam conteúdo interativo, dando impulso às TVs comunitárias e à produção de conteúdo pelas grandes emissoras.

Segundo Cruz (2008, p. 30), o domínio das aplicações para o Ginga pode ser dividido em dois conjuntos: o das aplicações declarativas e o das aplicações procedurais. Conceitualmente falando, uma linguagem declarativa descreve “o que” e não “como” seus procedimentos funcionam, assim descrevendo as propriedades da solução desejada e não especificamente o que o algoritmo deve fazer. Já na linguagem procedural, o programador deve especificar como as coisas devem acontecer, o que exige uma melhor experiência e conhecimento do mesmo. A linguagem mais comum encontrada nos ambientes procedurais em um sistema de TV digital é Java.

O Portal do Software Público Brasileiro (2009) afirma que uma aplicação não precisa ser puramente declarativa ou puramente procedural. Nos sistemas de TV digital, os dois tipos de aplicação coexistirão.

2.2.1.1 Arquitetura do Ginga

A definição do *middleware* Ginga é dividido em dois motores: o Ginga-NCL ou Máquina de Apresentação (domínio declarativo) e o Ginga-J ou Máquina de Execução (domínio procedural). Para prover a integração entre os motores foi definida uma ponte de comunicação, e por fim, rodando sobre os dois motores, existe um núcleo comum para a utilização de funções genéricas aos mesmos e uma Java Virtual Machine (JVM). Uma visualização da arquitetura em alto nível pode ser vislumbrada na Figura 4.



Fonte: Paulinelli e Kulesza (2009).

Figura 4 – Arquitetura em alto-nível do *middleware* Ginga

Durante a definição de um projeto de TV digital deve-se analisar qual domínio atende as reais necessidades deste, e mesmo com a adoção de um domínio específico não restringirá o domínio da aplicação, pois através da ponte de execução presente, os domínios podem comunicar-se, provendo funcionalidades do Ginga-J em aplicações Ginga-NCL e vice-versa.

Detalhes pertinentes a arquitetura serão apresentados nas seções seguintes.

2.2.1.1.1 Ginga-NCL

O Ginga-NCL ou Máquina de Apresentação é um motor lógico do Ginga, desenvolvido pela PUCRJ, vinculado ao domínio declarativo das aplicações e que provém uma estrutura de apresentação das aplicações com base em documentos hipermídia.

Sampaio (2008, p. 46) afirma que o Ginga-NCL traz uma inovação muito importante, a linguagem *Nested Context Language* (NCL)[3]. Ao contrário dos outros *middlewares* que utilizam o eXtensible *HyperText Markup Language* (XHTML), a linguagem NCL separa de maneira bem demarcada o conteúdo e a estrutura da aplicação. Desta forma o controle do relacionamento entre o conteúdo e a maneira como este é apresentado torna-se menos invasivo.

O processamento destes documentos é realizado pelo motor de decodificação de conteúdo declarativo chamado NCL *formatter*. Este recebe um documento NCL e controla sua apresentação, sincronizando os objetos de mídia, fazendo com que eles sejam

apresentados no momento programado.

Ainda em Sampaio (2008, p. 47), a linguagem NCL, processa e decodifica objetos comuns como imagens, arquivos compactados e até mesmo documentos XHTML. Em outras palavras, um documento NCL pode referenciar um documento XHTML. Para exibir este tipo de mídia o Ginga possui um navegador XHTML que segue as recomendações do World Wide Web Consortium (W3C). A norma não especifica um navegador padrão, pelo contrário, a especificação do Ginga permite que existam vários navegadores e que de acordo com as necessidades do documento XHTML seja utilizado um navegador capaz de exibir a mídia da maneira correta.

Além disso o navegador deve ser capaz de decodificar corretamente *Cascading Style Sheets* (CSS) e ECMAScript. Enquanto a linguagem XHTML e os scripts ECMAScript eram padrão nos outros *middlewares* declarativos, o Ginga-NCL utiliza scripts Lua[4] como linguagem procedural.

2.2.1.1.2 Ginga-J

O Ginga-J ou Máquina de Execução é um motor lógico do Ginga, desenvolvido pela UFPB, vinculado ao domínio procedural das aplicações e que provém uma estrutura de execução das aplicações procedurais baseadas na linguagem Java através da JVM comum à especificação.

2.2.1.1.2.1 Especificação Ginga-J

Conforme Associação Brasileira de Normais Técnicas (2007, p. 7), a definição Ginga-J foi composta por APIs projetadas para suprir todas as funcionalidades necessárias para a implementação de aplicativos para TV digital, desde a manipulação de dados multimídia até protocolos de acesso. Inicialmente, na primeira especificação do Ginga-J (figura 5), dava-se suporte às aplicações Ginga-J através das APIs da *Globally Executable MHP* (GEM), modelo criado mundialmente para interoperabilidade de aplicações no *middleware Multimedia Home Plataform* (MHP) do sistema europeu de TV digital.

---

[3] A linguagem NCL - Nested Context Language - é uma linguagem declarativa para autoria de documentos hipermídia baseados no modelo conceitual NCM - Nested Context Model. Um aplicativo NCL define apenas como os objetos de mídia são estruturados e relacionados no tempo e espaço.

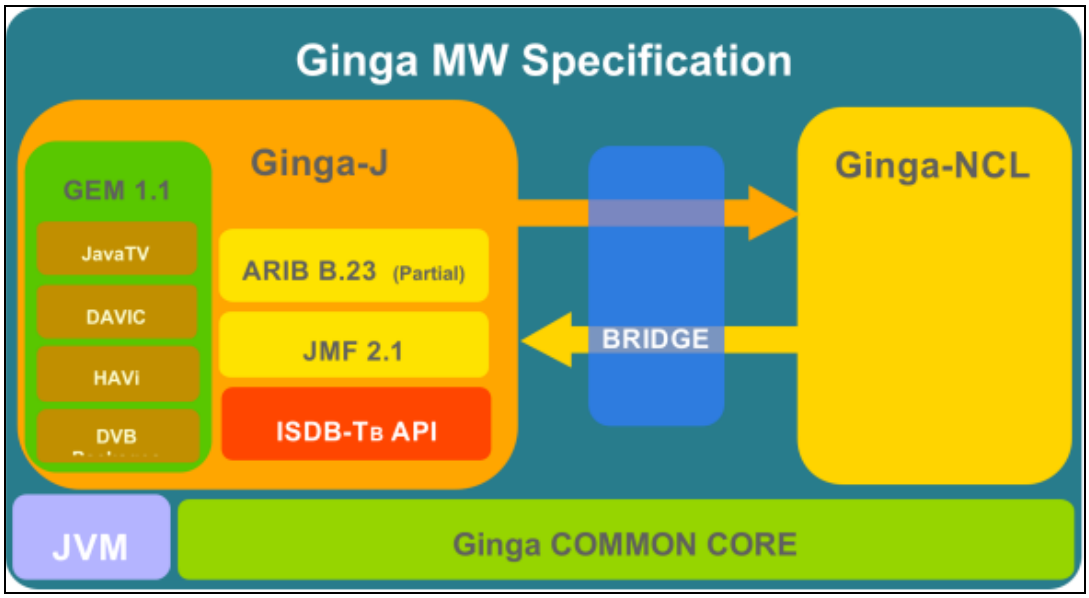


Figura 5 – Cenário inicial da especificação Ginga-J

Por questões jurídicas e principalmente para seguir a premissa de que o Ginga deveria ser acessível e livre de *royalties*, o fórum SBTVD decidiu que seria necessário mudar as especificações do sistema. E em acordo com a Sun Microsystems, recebeu a especificação de uma API recém criada, chamada de Java DTV para utilização no Ginga-J sem cobrança alguma de *royalties* (FÓRUM SBTVD, 2009). Com a utilização da Java DTV a especificação do Ginga-J apresenta-se conforme a figura 6.

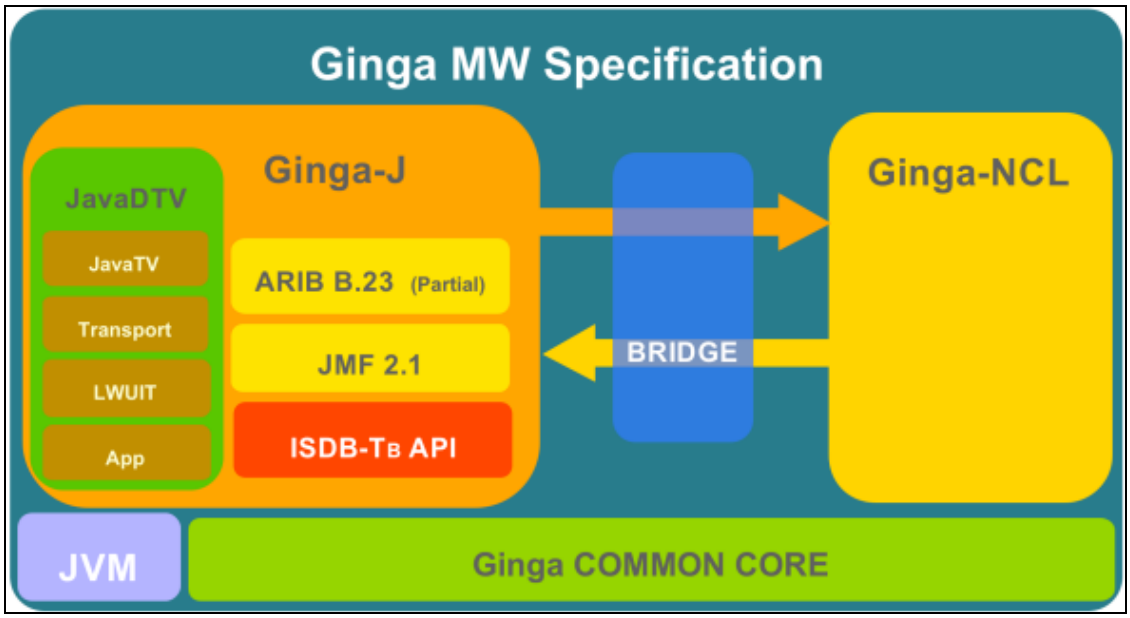


Figura 6 – Cenário final da especificação Ginga-J

Segundo Paulinelli e Kulesza (2009), a especificação Ginga-J fica formada então por uma adaptação da API de acesso à informação de serviço do *middleware* japonês, o *Association of Radio Industries and Businesses* (ARIB), pela especificação Java DTV (que

[4] Lua é uma linguagem brasileira desenvolvida em 1993 pela PUCRJ, leve, robusta, e voltada para interfaces.

inclui a API Java TV), além de um conjunto de APIs adicionais de extensão ou inovação.

Ainda em Paulinelli e Kulesza (2009), detalha-se a especificação Java DTV como uma plataforma aberta e interoperável que permite a implementação de serviços interativos com a linguagem Java, tendo sido inserida recentemente ao conjunto de APIs do Ginga-J. Funcionalmente, a Java DTV substitui a coleção de APIs utilizada anteriormente e definida pelo padrão GEM, tais como Digital *Audio VIdeo Council* (DAVIC) e *Home Audio Video Interoperability* (HAVi). Entre as principais diferenças da Java DTV quanto ao desenvolvimento de aplicações, destava-se a API *LightWeight User Interface Toolkit* (LWUIT), responsável por definir elementos e extensões gráficas para TV digital, gerenciadores de layout e eventos do usuário. O objetivo é substituir a API HAVi do GEM.

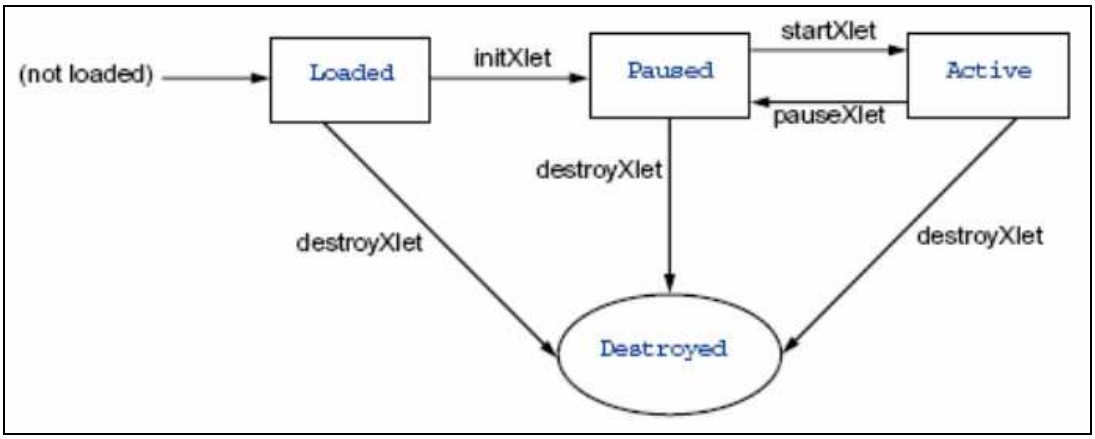
De acordo com o Fórum SBTVD (2009), o Ginga-J é a única especificação de *middleware* onde não há cobrança de *royalties* disponível, com isso é esperado a adoção global dessa especificação mais ainda pelo fato do SBTVD não estar somente no Brasil e ser considerado como o padrão mais avançado do mundo. Outra vantagem destacável ao Ginga-J é a grande comunidade brasileira de desenvolvedores em Java, que conta com mais de 110.000 desenvolvedores (SUN MICROSYSTEMS, 2010).

Em relação ao desenvolvimento em Ginga-NCL, a Ginga-J possui anos de atraso devido às questões supracitadas, porém finalmente Uchoa (2010) noticia que a nova especificação Ginga-J está disponível para consulta nacional pela Associação Brasileira de Normas e Técnicas, e desta forma, destrava o mercado e os desenvolvedores para implementação conforme estas normas.

2.2.1.1.2.2 Aplicações em Ginga-J

Segundo Costa e Melo Junior (2009, p.3), pode-se denominar de *Xlet* toda e qualquer aplicação desenvolvidas em Ginga-J ou em qualquer outro *middleware* de TV digital que tenha a máquina de execução procedural como plataforma.

O conceito dos *Xlets* é semelhante aos dos *Applets*, *Servlets* ou *Midlets*, ou seja, possui um ciclo de vida, gerenciado pelo *container*. Assim como o browser gerencia o ciclo de vida de um *applet* e um *container web* gerencia o ciclo de vida de um *servlet*, o *middleware* embarcado no receptor digital gerencia o ciclo de vida (figura 7) do *Xlet* (DEMONIO DE MAXWELL, 2009).



Fonte: Sampaio (2008).

Figura 7 – Ciclo de vida de um *Xlet*

Segundo Pawlan (2001) não há nenhum método principal e todo e qualquer *Xlet* sempre deverá implementar a interface `javax.tv.xlet.Xlet`. Cada método desta interface tem um papel importante no gerenciamento do um ciclo de vida do *Xlet*, e estão listados na figura 8.

```
public interface Xlet {

   public void initXlet(XletContext ctx) throws XletStateChangeException;
   public void startXlet() throws XletStateChangeException;
   public void pauseXlet();
   public void destroyXlet(boolean unconditional) throws XletStateChangeException;
}
```

Figura 8 – Interface `javax.tv.xlet.Xlet`

Sampaio (2008, p.32) afirma que um *Xlet* pode estar em um dos quatro estados possíveis: carregado, ativo, pausado ou destruído. Para fazer a transição entre os estados deve ocorrer uma chamada a um método da interface do *Xlet*, que pode ser feito pelo gerenciador de aplicação ou pela própria *Xlet*.

Ainda em Sampaio (2008, p. 33), as transições entre os possíveis estados e métodos da *interface* podem ser vislumbrados conforme:

a) estado carregado: uma aplicação chega ao estado carregado quando o gerenciador de aplicação carrega na memória o *Xlet* invocando o método `new` do Java. Uma vez carregada e inicializada uma aplicação não volta ao estado carregado;
b) estado pausado: uma *Xlet* chega ao estado pausado após a invocação do método `InitXlet`. Neste processo o gerente passa à aplicação o contexto de execução (*XletContext*) e a partir desse momento a aplicação pode obter propriedades do ambiente e enviar mudanças de estado para o gerenciador de aplicação. Enquanto estiver no estado pausado a *Xlet* não pode consumir nenhum recurso

compartilhado;

c) estado ativo: para passar ao estado ativo um método `StartXlet` deve ser invocado. No estado ativo a aplicação pode usufruir dos recursos para ativar suas funcionalidades e prover serviços. A aplicação pode retornar ao estado pausado através do método `pauseXlet`;
d) estado destruído: a qualquer momento uma aplicação pode passar ao estado destruído com a chamada do método `destroyXlet`. Quando neste estado a aplicação libera todos os recursos e para a sua execução.

2.2.1.1.3 Ponte de motores

Segundo o Portal do Software Público Brasileiro (2009), diferente dos outros sistemas, os ambientes de apresentação e execução do *middleware* Ginga se complementam, unidos por uma ponte em uma implementação sem nenhuma redundância, o que confere ao sistema uma ótima eficiência, tanto em termos de uso de *Central Processing Unit* (CPU) quanto de ocupação de memória. Ao contrário dos outros sistemas, Ginga, desde seu projeto inicial, foi desenvolvido tendo em mente os dois ambientes de programação.

A ponte provê um mecanismo de mapeamento que permite a interação de qualquer aplicação nos dois motores, seja através da manipulação de documentos NCL pelo Ginga-J, ou pela execução de código Java através de *Xlets* NCL ou efetuando chamadas para as funcionalidades do Ginga-J com *scripts* Lua.

2.2.1.1.4 Ginga-CC

O núcleo comum, Ginga *Commom-Core* (Ginga-CC), oferece o suporte básico necessário para os motores Ginga-NCL e Ginga-J, e localiza-se entre estes e o sistema operacional do dispositivo. É responsável pelo processamento e decodificação de conteúdo comum aos motores citados. Exemplos disto são os objetos de mídia como *Portable Network Graphics* (PNG) e *Joint Photographic Experts Group* (JPEG).

O Ginga-CC encarrega-se de efetuar este processamento e ainda contém

procedimentos para controlar o fluxo de transporte através do carrossel[5] de dados e do canal de interatividade, além de prover gerenciamento das aplicações, decodificação dos conteúdos, controle do plano gráfico, entre outros.

2.2.1.2 Implementações de referência Ginga-J

Em paralelo às indefinições na especificação do Ginga-J, projetos com intuito de servir como implementações de referência foram desenvolvidos. Entre elas, destacam-se a OpenGinga (GINGADF, 2009) e a MOStvd (LANG, 2009).

Segundo GingaDF (2009), a OpenGinga é uma plataforma que possibilita a execução de aplicações Ginga num computador pessoal. Essa inclui um sistema operacional, uma implementação de referência do *middleware*, o emulador Ginga-J em Java e aplicações de exemplo. Disponibiliza atualmente apenas o suporte à execução de aplicações Java (Ginga-J). É desenvolvido pelo Laboratório de Aplicações de VÍdeo Digital (LAVID) e a UFPB em parceria com outras empresas.

Costa e Melo Junior (2009, p.5) cita o OpenGinga como uma implementação de referência do Ginga-J, que está na segunda versão beta, porém esta ainda é uma versão para desenvolvedores avançados e com conhecimento de sistemas de *middleware*, não existindo assim ferramentas auxiliares para esse desenvolvimento.

De acordo com Lang (2009), o MiddlewareOpenSource para TV Digital (MOStvd) é uma implementação de um *middleware* para TV digital que esteja de acordo com a norma Ginga. Este projeto tem como objetivo facilitar a criação de aplicativos para TV digital, bem como a criação de módulos de um *middleware*, disseminação da tecnologia e auxílio a novos projetos. O MOSTvd, consiste em um ambiente pronto para executar *Xlets*, padronizado na especificação JavaDTV e conseqüentemente no padrão Ginga-J.

Segundo Costa e Melo Junior (2009, p. 4), por estar no início da implementação, o MOStvd também traz algumas falhas e limitações. A interface gráfica do gerenciador do *middleware* apresenta problemas com a implementação feita no LWUIT, algumas limitações também foram notadas até mesmo no uso de imagens na interface gráfica, onde na aplicação construída não foi possível colocar nenhum tipo de imagem. Outra limitação do MOStvd é o uso do *Java Media Framework* (JMF), uma biblioteca externa, para que o *Xlet* possa carregar

[5] Carrosel de dados define-se como um mecanismo de transmissão cíclica de dados.

um vídeo.

2.3 TWITTER

Segundo Martins e Leal (2009), vive-se na era da exposição e do compartilhamento. A idéia da privacidade vai mudar ou desaparecer. No mundo todo são disparados 2,4 trilhões de *Short Message System*[6] (SMS) por mês. Também é comum enviar *e-mails*, falar com pessoas através de redes sociais, usar comunicadores instantâneos e celulares, receber chamados em qualquer parte a qualquer hora. Todos estão conectados, de várias formas, e não querem parar de falar e não querem parar de receber, sendo o mais recente exemplo desta demanda total por conexão e de uma nova sintaxe social, o Twitter.

Criado em 2006, o Twitter foi fundado por Jack Dorsey, Evan Willians e Biz Stone. A idéia, que partiu de Dorsey durante uma reunião de discussão de idéias, era de um serviço de troca de *status*, como um SMS. Chamado inicialmente de *Status*, optou-se pela troca do nome pelo fato de que ele não mostrava exatamente o que era o serviço. Após uma busca em dicionários por sinônimos encontrou-se a palavra *twitter*, que possui dois significados em inglês: “uma pequena explosão de informações inconseqüentes” e “pios de pássaros”. Ambos encaixavam-se no conceito do serviço e dão significado ao logotipo posteriormente criada (figura 9).

Figura 9 – Logotipo do Twitter

[6] O *Short Message System (SMS)* é um serviço disponível em telefones celulares (telemóveis) digitais que permite o envio de mensagens curtas com até 255 caracteres entre estes equipamentos e outros dispositivos com suporte ao mesmo.

A limitação de 140 caracteres por mensagem parte do conceito inicial do projeto que estava relacionado às mensagens SMS que possuem também limitação.

Ainda de acordo com Martins e Leal (2009), a grande novidade do Twitter é o ritmo. Por algum motivo inexplicável, as pessoas não param de trocar mensagens. O *site* do Twitter faz uma pergunta básica - "O que você está fazendo?" – e todo mundo responde, várias vezes ao dia. Como é possível a atualização em celulares, os usuários não descansam na narração trivial. A facilidade de informar e estar informado com rapidez são um dos atrativos do serviço. As atualizações são conhecidas como *tweets*, e podem partir de pessoas simples narrando fatos presenciados pelas mesmas, ou até mesmo celebridades que informam detalhes sobre sua vida pessoal a milhares de fãs seguidores.

Toda essa necessidade de acesso pode estar relacionada à outra atividade muito trivial em muitos dos domicílios brasileiros: assistir a um programa televisivo. Muito mais utilizada do que computadores e celulares, a televisão está presente em 95,1% dos domicílios brasileiros (TELECO, 2009), somente não se faz presente em domicílios sem recurso de energia elétrica. Ao ampliar a esfera de possibilidades de acesso, utilizando a TVDI junto ao Twitter, os usuários não precisam deslocar-se até seu computador ou estar de posse de seus celulares para acompanhar as atualizações e estar sincronizado com a sua rede social.

2.3.1 Twitter API

Segundo Strickland (2010), o Twitter baseia-se numa API com a arquitetura *REpresentional State Transfer* (REST[7]). Para o Twitter, usar a arquitetura REST significa em parte que o serviço funciona com a maioria dos formatos de sindicação da Internet. A sindicação de conteúdo é um conceito simples: um aplicativo reúne informações de uma fonte e as envia para vários destinos . Existem alguns formatos de sindicação usados na Internet. O Twitter é compatível com dois deles, o *Really Simple Syndication (RSSa)* e o *Atom Syndication Format* (ASF). Ambos os formatos utilizam o sistema de representação da informação baseado na XML.

[7] Segundo Strickland (2010) a Transferência de Estado Representacional (REST) refere-se a um conjunto de princípios de *design* para a rede que definem os recursos e as maneiras de localizar e acessar dados. A arquitetura é uma filosofia de design e não um conjunto de planos, não existindo um ajuste único e fixo de computadores, servidores e cabos.

Ainda em Strickland (2010), os dois formatos de sindicação consistem na utilização de algumas linhas de código, que podem ser adicionadas gratuitamente, por exemplo, em qualquer *site* por seu administrador. Os visitantes do *site* podem inscrever-se no serviço de sindicação, chamado de *feed*, e receber uma atualização toda vez que o administrador atualizá-lo. De fato, os membros do Twitter se inscrevem para receber *feeds* de outros membros.

Segundo o Twitter Developers (2010a), a Twitter API atualmente é composta por duas APIs REST discretas e uma API de *Streamming* para buscas, ou seja existem realmente três APIs, sendo comum a utilização e combinação de ambas pela maioria dos desenvolvedores na produção de suas aplicações. A comunicação entre as APIs é simples, porém a integração plena dos serviço, por se tratar de uma conversão complexa, não deve ocorrer tão cedo.

Ainda em Twitter Developers (2010a), os métodos da API REST permitem que os desenvolvedores acessem dados essenciais do Twitter como as atualizações, dados das atualizações e informações do usuário. Já a API de busca provém aos desenvolvedores métodos para interagir com as buscas em mensagens e dados de tendências.

Uma preocupação para os desenvolvedores dada esta separação entre as APIs é a limitação da taxa de uso dos serviços, já que a API de busca retorna quase que em tempo real um alto volume de dados de *tweets* que podem ser filtrados e formulados conforme a necessidade. Esta limitação se fez necessária para evitar a utilização do serviço para fins impróprios como à divulgação de *Spams*[8]. Questões relacionadas às limitações do serviço podem ser encontradas em Twitter Developers (2010b).

2.3.2 A utilização da Twitter API

A Twitter API é totalmente baseada em *HyperText Transfer Protocol* (HTTP), e utiliza conforme a necessidade e funcionalidade uma das quatro requisições deste: GET, POST, PUT e DELETE.

O fluxo da utilização da API pode ser dividido em três passos: autenticação, requisição e retorno. A autenticação não é obrigatória, pois existem métodos que não necessitam da mesma, já a requisição e retorno, em forma geral seguem o formato

[8] O termo Spam, abreviação em inglês de "*spiced ham*" (presunto condimentado), é uma mensagem eletrônica não-solicitada enviada em massa.

[URL]/[Função].[Formato]?[Parâmetros] onde:

a) URL: segue o endereço padrão HTTP://api.twitter.com;
b) função: nome único da função e que deve ser observado na documentação;
c) formato: define a saída da informação, ou seja, o formato dos dados que podem ser retornados em XML, *JavaScript Object Notation* (JSON), *Rich Site Summary* (RSSb) e ATOM;
d) parâmetros: listagem de parâmetros não opcional, relacionado à função e que deve ser observado na documentação.

Como exemplo, pode-se verificar a função `Friends Timeline`, documentada em Twitter Developers (2010c), e que retorna as vinte mais recentes atualizações da rede de contatos do usuário. Para sua utilização, deve-se efetuar uma requisição em qualquer um dos quatro formatos supracitados para a função `statuses/friends_timeline`. Pode-se ainda passar parâmetros como o `include_rts`, que retorna ou não os encaminhamentos (*retweets)* enviados pelo mesmo, o `count` que especifica o número de registros que podem ser retornados, entre outros.

O retorno das informações, no formato XML por exemplo, pode ser requisitado na URL http://api.twitter.com/version/statuses/friends_timeline.xml e vislumbrado no Quadro 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<statuses type="array">
<status>
  <created_at>Mon Oct 11 17:12:35 +0000 2010</created_at>
  <id>27052654477</id>
  <text>Venha ao Palmeiras fazer a festa no Dia das Crian&#231;as:
http://bit.ly/cd92vn</text>
  <source>web</source>
  <truncated>false</truncated>
  <in_reply_to_status_id></in_reply_to_status_id>
  <in_reply_to_user_id></in_reply_to_user_id>
  <favorited>false</favorited>
  <in_reply_to_screen_name></in_reply_to_screen_name>
  <retweet_count>0</retweet_count>
  <retweeted>false</retweeted>
  <user>
    <id>46104914</id>
    <name>Palmeiras Oficial</name>
    <screen_name>SitePalmeiras</screen_name>
    <location>S&#227;o Paulo/Brazil</location>
    <description>Tudo oficial do Verd&#227;o no Twitter. </description>

<profile_image_url>http://a0.twimg.com/profile_images/269730492/teste2_normal.gi
f</profile_image_url>
    <url>http://www.palmeiras.com.br</url>
    <protected>false</protected>
    <followers_count>129481</followers_count>
    <profile_background_color>9AE4E8</profile_background_color>
    <profile_text_color>333333</profile_text_color>
    <profile_link_color>0084B4</profile_link_color>
    <profile_sidebar_fill_color>DDFFCC</profile_sidebar_fill_color>
    <profile_sidebar_border_color>BDDCAD</profile_sidebar_border_color>
    <friends_count>7</friends_count>
    <created_at>Wed Jun 10 12:57:48 +0000 2009</created_at>
    <favourites_count>0</favourites_count>
    <utc_offset>-10800</utc_offset>
    <time_zone>Brasilia</time_zone>

<profile_background_image_url>http://a1.twimg.com/profile_background_images/1886
0732/twiter.jpg</profile_background_image_url>
    <profile_background_tile>false</profile_background_tile>
    <profile_use_background_image>true</profile_use_background_image>
    <notifications>false</notifications>
    <geo_enabled>false</geo_enabled>
    <verified>false</verified>
    <following>true</following>
    <statuses_count>1935</statuses_count>
    <lang>en</lang>
    <contributors_enabled>false</contributors_enabled>
    <follow_request_sent>false</follow_request_sent>
    <listed_count>2125</listed_count>
    <show_all_inline_media>false</show_all_inline_media>
  </user>
  <geo/>
  <coordinates/>
  <place/>
  <contributors/>
</status>
...
```

Quadro 1 – Retorno da função `Friends Timeline`

Nota-se que além das informações de cada atualização, retornam-se também informações do usuário responsável pela mesma, coordenadas de localização da atualização, entre outras que podem ser vislumbradas na documentação da função.

A API completa é composta por diversas funções além da citada cima, e divididas em

grandes áreas como os métodos da linha de tempo, das atualizações, de usuários, de listagens, de associações, entre outras. Cada funcionalidade possui suas especificidades, como o método HTTP que deve ser empregado, se a mesma necessidade de autenticação, se ela possui acesso limitado, quais os formatos de retorno previstos e outras informações relevantes. O detalhamento completo de cada função pode ser encontrado na documentação completa da API presente em Twitter Developers (2010d).

2.3.3 Autenticação a Twitter API com a *Open Autorization*

A *Open Autorization* (OAuth) consiste num protocolo de autenticação que permite aos usuários do Twitter aprovarem ou não o acesso de aplicativos terceiros ao serviço através de suas credenciais de acesso. Ao aceitar a utilização, o usuário concorda que o aplicativo possa agir em seu nome e interagir com os serviços da Twitter API.

Toda aplicação que deseja utilizar a Twitter API deve estar devidamente registrada. Durante o processo de registro, o preponente deve indicar informações da aplicação em si, da organização mantenedora, da forma de autenticação ao serviço, se é uma aplicação *on-line*, como um *web-site*, ou *off-line*, como um aplicativo para celular e se utilizará apenas métodos de leitura ou de escrita.

Caso a aplicação seja *on-line*, o fluxo de autenticação consiste no envio das informações de conexão com métodos HTTP como o POST, onde o usuário é direcionado a uma página contendo as informações da aplicação e perguntando-o sobre a autorização ou não do acesso da mesma. Já para aplicações *off-line*, o padrão consiste no desvio do fluxo da aplicação para uma URL onde o usuário faz a mesma aprovação, porém o resultado é um número PIN que deve ser informado à aplicação para que o acesso seja consolidado.

Existe ainda um terceiro método, conhecido como xAuth, onde aplicações *off-line* são liberadas do processo de autorização por parte do usuário. Porém para que a equipe da Twitter API libere este método é necessária a abertura de um chamado em seu setor de suporte e ocorre um processo de análise da aplicação, onde detalhes e capturas de telas da mesma são solicitados.

Findado o processo de registração, recebe-se duas chaves únicas, a *Consumer Key* e a *Secret Key*, que juntas dos dados de acesso serão enviadas ao serviço de autenticação, e apenas com a validação das mesmas o aplicativo terceiro consegue o acesso aos recursos da

API.

2.4 TRABALHOS CORRELATOS

Existe uma vasta gama de trabalhos envolvendo o assunto da TV digital interativa, porém quando a busca é relacionada ao *middleware* Ginga os resultados são mais escassos, e na maioria dos casos tendo o Ginga-NCL como o enfoque principal. Neste contexto, o trabalho “Ginga-NCL para dispositivos portáteis” de Cruz (2008) é referenciado. No contexto geral , “InteraTV: um portal para aplicações colaborativas em TV digital interativa utilizando a plataforma MHP” de Andreata (2006) também é referenciado. O trabalho “Desenvolvimento de servidor de RSS para TV digital Interativa” de Schroeder (2007) também é citado.

Cruz (2008) apresenta um estudo com intuito de servir como prova de conceito da especificação de Ginga-NCL e desta forma demonstra a viabilidade e uso na prática da especificação para os dispositivos portáteis. É um trabalho que apresenta comparativos entre diversos sistemas operacionais e linguagens, chegando à conclusão que para o desenvolvimento o sistema operacional Symbian apresentava as maiores vantagens, e a linguagem escolhida foi o Symbian C++. Trata de diversos temas gerais, como limitações e soluções encontradas, bem como diversos testes de aplicações sendo executas na implementação proposta.

Andreata (2006) propõe um portal para disponibilização de aplicações colaborativas em ambiente de TV Digital Interativa (TVDI), denominado InteraTV. Sua especificação é compatível com a camada de software MHP e desta forma padronizada com o sistema europeu de TV digital. O objetivo geral é utilizar o portal para apoio à área de educação a distância, mais conhecida como *T-learning*, sendo apresentadas diversas aplicações educacionais e acadêmicas desta natureza, e discutidas as viabilidades de implementação. Apresenta também as dificuldades que foram encontradas durante o desenvolvimento do protótipo, além de abordagem da diferença no desenvolvimento de aplicações para o ambiente de TV digital em comparação ao tradicional ambiente computacional.

Por fim, Schroder (2007) introduz conceitos dos sistemas de TV digital com enfoque no tráfego de dados juntamente do sinal, e desenvolve assim um protótipo para TVDI. Devido ao serviço de TV digital ainda estar em implantação na época do desenvolvimento deste, o

protótipo é executado no ambiente de emulação OpenMHP. A aplicação tem como função principal interagir com um provedor de serviço de difusão, através do canal de retorno e receber informações e notícias no formato RSSb. Este trabalho também utiliza o padrão MHP, ficando desta forma compatível com o sistema europeu de TV digital.

3 DESENVOLVIMENTO DO APLICATIVO

Este capítulo apresenta todo o ciclo de desenvolvimento do aplicativo proposto, contendo os requisitos a serem trabalhados, a especificação, a implementação e por fim os resultados obtidos com o desenvolvimento do aplicativo.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O aplicativo desenvolvido neste trabalho contempla os seguintes requisitos:

a) conectar ao Twitter através de tela com *login* e senha (Requisito Funcional – RF);
b) permitir a listagem para o usuário das últimas vinte (20) atualizações de sua rede de contatos, contendo imagem do contato e a linha do tempo (RF);
c) enviar ou excluir um *tweet* de sua autoria (RF);
d) responder ou encaminhar (*re-tweet*) um *tweet* selecionado (RF);
e) desconectar do Twitter (RF);
f) ser implementado utilizando a linguagem de programação Java no ambiente de desenvolvimento Eclipse (Requisito Não Funcional - RNF);
g) ser desenvolvido em conformidade com as normas do *middleware* Ginga-J (RNF);
h) implementar as funcionalidades de acordo com a especificação Twitter API (RNF);
i) utilizar o formato XML para retorno das informações provenientes da Twitter API (RNF);
j) utilizar teclado virtual semelhante ao teclado de celular para efetuar o login e enviar um *tweet* (RNF);
k) rodar em qualquer plataforma com suporte ao *middleware* Ginga-J (RNF).

3.2 ESPECIFICAÇÃO

A próxima seção apresenta a especificação do aplicativo e suas funcionalidades, modelada com base em análise orientada a objeto, utilizando-se a UML. Para a análise optou-

se pela modelagem dos diagramas de casos de uso, de classe, e de estado através da ferramenta *Enterprise Architecht*. A seguir, são detalhados os diagramas e por conseqüência as funcionalidades previstas para o aplicativo.

3.2.1 Diagrama de casos de uso

Ao todo se verificou a necessidade de oito casos de uso (ilustrados na figura 10) para a representação das funcionalidades previstas ao aplicativo. São elas: efetuar *login* à conta do Twitter, efetuar logoff à conta do Twitter, listagem dos *tweets* (mensagens) da rede, excluir *tweet*, redigir um *tweet*, enviar *tweet*, encaminhar um *tweet* e responder à um contato.

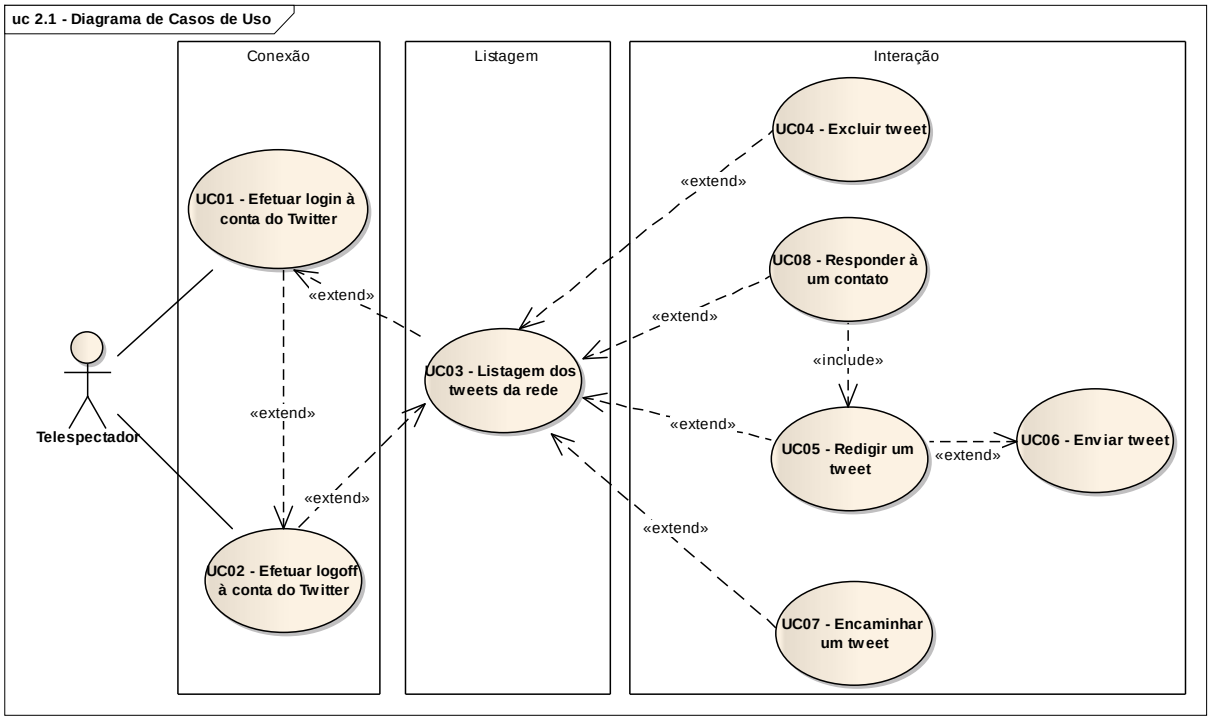


Figura 10 – Diagrama de casos de uso do aplicativo

O primeiro caso de uso (Quadro 02), denominado UC01 - Efetuar *login* à conta do Twitter, permite a conexão ao Twitter através do nome de usuário e senha que serão informados pelo telespectador. Para informar as credencias de acesso o usuário utilizará as teclas numéricas de seu controle. Em caso de falha no *login* por mais de três vezes, o aplicativo é finalizado.

UC01 – Efetuar *login* à conta do Twitter: Conectar a conta do Twitter no aplicativo.	
Pré-condições	Não há.
Cenário principal	1 - Cenário Principal 1.1 - A tela de *login* é mostrada para o telespectador; 1.2 - O telespectador informa o nome de usuário; 1.3 - O telespectador informa a senha de acesso; 1.4 - O telespectador clica no botão verde; 1.5 - O aplicativo faz a tentativa de conexão ao Twitter (API); 1.6 - O aplicativo conecta-se ao Twitter com as credencias informadas. 1.7 - Executa UC03.
Cenário alternativo 01	No passo 1.4, valida-se os campos obrigatórios: 1.4.1 - Caso o nome de usuário esteja em branco, apresentar mensagem "Nome de usuário deve ser informado!", e retornar ao passo 1.2;
Cenário alternativo 02	No passo 1.4, valida-se os campos obrigatórios: 1.4.1 - Caso a senha esteja em branco, apresentar mensagem "Senha deve ser informada!", e retornar ao passo 1.3;
Cenário alternativo 03	No passo 1.2, caso o telespectador não deseje continuar: 1.2.1 - O telespectador clica no botão vermelho; 1.2.2 - O aplicativo é finalizado.
Exceção 01	No passo 1.5, caso o acesso ao Twitter não seja consolidado: 1.5.1 - Exibir mensagem 'Login/Senha inválidos! Favor tentar novamente!"; 1.5.2 - Caso o número de tentativas de acesso for igual a três (3), mostrar a mensagem "Número de tentativas excedido, finalizando o aplicativo!", e finalizar o aplicativo. 1.5.3 - Caso o número de tentativas de acesso não tenha sido excedido, retornar o fluxo ao passo 1.2;
Pós-condições	O aplicativo conecta-se ao Twitter do telespectador. O aplicativo é encerrado.

Quadro 2 – Detalhamento do caso de uso `Efetuar *login* à conta do Twitter`

O segundo caso de uso (Quadro 03), denominado UC02 - Efetuar *logoff* à conta do Twitter, permite a desconexão ao Twitter para evitar o uso inadequado do aplicativo por terceiros.

UC02 – Efetuar *logoff* à conta do Twitter: Desconectar a conta do Twitter no aplicativo.	
Pré-condições	Não há.
Cenário principal	1 - Cenário principal 1.1 - O aplicativo pergunta "Você realmente deseja desconectar-se do Twitter?" 1.2 - O aplicativo efetua a desconexão do Twitter (API); 1.3 - Executa UC01.
Cenário alternativo 01	No passo 1.1, caso o usuário não confirme a desconexão: 1.2.1 - Finalizar o caso de uso.
Exceção 01	No passo 1.2, caso ocorra erro durante a chamada de desconexão da API: 1.1.1 - Apresentar mensagem: "Não é possível desconectar do Twitter, impossível efetuar conexão a API!"; 1.1.2 - Finalizar o caso de uso.
Pós-condições	O aplicativo desconecta-se ao Twitter do telespectador.

Quadro 3 – Detalhamento do caso de uso `Efetuar *logoff* a conta do Twitter`

O terceiro caso de uso (Quadro 04), denominado UC03 - Listagem dos *tweets* da rede, é responsável pela listagem dos últimos *tweets* ou “atualizações” da rede do telespectador, provendo funcionalidades para exclusão e criação de *tweets* do mesmo, encaminhamento e

resposta de *tweets* de contatos, e desconexão do aplicativo.

UC03 – Listagem dos *tweets* da rede: Listagem dos últimos *tweets* da rede.	
Pré-condições	O aplicativo deve estar conectado ao Twitter.
Cenário principal	1.1 - O aplicativo lista os últimos *tweets* da rede (API); 1.2 - O aplicativo lista para cada *tweet* a imagem e o nome de usuário do contato, seguido da própria mensagem; 1.3 - O telespectador navega pelos *tweets* através das teclas de direção (para cima navega ao *tweet* anterior e para baixo o posterior); 1.4 - O telespectador clica no botão verde; 1.5 - Executa UC05 para a redação de um novo *tweet*. 1.6 - Retorna ao passo 1.1;
Cenário alternativo 01	Durante a execução do passo 1.3, caso o telespectador resolva excluir o *tweet* selecionado: 1.3.1 - O telespectador clica no botão vermelho; 1.3.2 - Executa UC04; 1.3.3 - Caso tenha sido efetuada a exclusão, a mensagem selecionada será eliminada da listagem. 1.3.4 - Retorna ao passo 1.1;
Cenário alternativo 02	Durante a execução do passo 1.3, caso o telespectador resolva encaminhar o *tweet* selecionado: 1.3.1 - O telespectador clica no botão azul; 1.3.2 - Executa UC07 passando como parâmetro o *tweet* selecionado; 1.3.3 - Retorna ao passo 1.1;
Cenário alternativo 03	Durante a execução do passo 1.3, caso o telespectador resolva responder ao contato do *tweet* selecionado: 1.3.1 - O telespectador clica no botão amarelo; 1.3.2 - Executa UC08 passando como parâmetro o nome do contato do *tweet* selecionado; 1.3.3 - Retorna ao passo 1.1;
Cenário alternativo 04	Durante a execução do passo 1.3, caso o telespectador resolva desconectar do Twitter: 1.3.1 - O telespectador clica no botão *exit*; 1.3.2 - Executa UC02.
Exceção 01	No passo 1.1, caso o aplicativo ainda não esteja conectado ao Twitter: 1.1.1 - Apresentar mensagem: "Não é possível desconectar do Twitter, pois não existe nenhuma conexão ativa!"; 1.1.2 - Finalizar o caso de uso.
Pós-condições	O aplicativo desvia o foco da aplicação para a tela de edição de *tweets*.

Quadro 4 – Detalhamento do caso de uso `Listagem dos tweets da rede`

O quarto caso de uso (Quadro 05), denominado UC04 - Excluir *tweet*, permite que o telespectador exclua *tweets* de sua autoria.

UC04 – Excluir *tweet*: Exclusão de *Tweet* do telespectador.	
Pré-condições	O aplicativo deve estar conectado ao Twitter. O aplicativo deve estar posicionado sobre um *tweet* do telespectador.
Cenário principal	1.1 - O aplicativo pergunta ao usuário se ele realmente deseja excluir o *tweet* através da mensagem: "Você realmente deseja excluir este *tweet*?"; 1.2 - O telespectador confirma com o clique no botão verde; 1.3 - O aplicativo efetua a exclusão do *tweet* (API); 1.4 - Finalizar o caso de uso.
Cenário alternativo 01	Durante a execução do passo 1.1, caso não haja a confirmação da exclusão: 1.1.1 – O telespectador clica no botão vermelho; 1.1.2 - Finalizar o caso de uso.
Exceção 01	Durante o passo 1.1: 1.1.1 - Caso o *tweet* selecionado não seja de autoria do telespectador, mostrar mensagem: "Você não pode excluir um *tweet* de outro contato!"; 1.1.2 - Finalizar o caso de uso.
Exceção 02	No passo 1.3, caso ocorra erro durante a chamada da exclusão de *tweets* da API: 1.3.1 - Apresentar mensagem: "Não é possível conectar-se a API, impossível efetuar exclusão do Tweet"; 1.3.2 - Finalizar o caso de uso.
Pós-condições	O aplicativo exclui um *tweet* da autoria do telespectador.

Quadro 5 – Detalhamento do caso de uso `Excluir *tweet*`

O quinto caso de uso (Quadro 06), denominado UC05 - Redigir um *tweet*, permite ao telespectador criar uma nova mensagem para enviar a sua rede utilizando o teclado numérico do controle remoto.

UC05 – Redigir um *tweet*: Permite a criação de uma nova mensagem para envio.	
Pré-condições	O aplicativo deve estar conectado ao Twitter.
Cenário principal	1.1 - A tela de edição de mensagem é mostrada ao telespectador; 1.2 - O telespectador utiliza o teclado numérico do celular para criar uma nova mensagem; 1.3 - O telespectador clica no botão verde; 1.4 - Executa o UC06; 1.5 - O aplicativo esconde a tela de edição; 1.5 - O aplicativo atualiza a listagem de *tweets* da rede.
Cenário alternativo 01	Durante a execução do passo 02, caso o telespectador desista da redação do *tweet*. 1.2.1 - O telespectador clica no botão vermelho; 1.2.2 - O aplicativo esconde a tela de edição; 1.2.3 - Finaliza o caso de uso.
Pós-condições	A tela de edição de mensagem é escondida.

Quadro 6 – Detalhamento do caso de uso `Redigir um *Tweet*`

O sexto caso de uso (Quadro 07), denominado UC06 - Enviar *tweet*, é responsável pelo envio da mensagem preenchida no campo de edição.

UC06 – Enviar *tweet*: Envio da mensagem preenchida no campo de edição.	
Pré-condições	O aplicativo deve estar conectado ao Twitter. O campo de edição deve estar preenchido com pelo menos um caractere.
Cenário principal	1.1 - O aplicativo valida a quantidade máxima de caracteres do texto contido no campo de edição; 1.2 - O aplicativo envia o texto do campo de edição para o Twitter (API); 1.3 - O aplicativo limpa o campo de edição; 1.4 - O aplicativo desvia o foco para o campo de edição.
Cenário alternativo 01	Durante a execução do passo 1.2: 1.1.1 - Caso o campo de edição possua quantidade de caracteres superior a quantidade máxima permitida, apresentar mensagem: "Não é possível enviar mensagem com quantidade superior a 140 caracteres!"; 1.1.2 - Finaliza o caso de uso.
Exceção 01	Durante a execução do passo 1.2, caso ocorra algum erro, enviar mensagem para alertar o telespectador: "Ocorreu erro durante o envio, a mensagem não será enviada!", e finalizar o caso de uso.
Pós-condições	Uma mensagem é enviada ao Twitter.

Quadro 7 – Detalhamento do caso de uso `Enviar *tweet*`

O sétimo caso de uso (Quadro 08), denominado UC07 - Encaminhar um *tweet*, permite ao telespectador encaminhar um *tweet* previamente enviado por algum contato da sua rede. Este conceito é conhecido como *Retweet*.

UC07 – Encaminhar um *tweet*: Efetuar o encaminhamento para seus contatos de um *tweet*.	
Pré-condições	O aplicativo deve estar conectado ao Twitter. A rotina chamadora deve enviar como parâmetros o *tweet* que será encaminhado.
Cenário principal	1.1 - O aplicativo pergunta ao usuário se ele realmente deseja encaminhar o *tweet* através da mensagem: "Você deseja encaminhar este tweet para sua rede?"; 1.2 - O telespectador confirma com o clique no botão verde; 1.3 - O aplicativo efetua o encaminhamento do *tweet* (API); 1.4 - Finalizar o caso de uso.
Cenário alternativo 01	Durante a execução do passo 1.1, caso o telespectador não confirme o encaminhamento através do clique no botão vermelho: 1.1.1 - Sair do caso de uso.
Exceção 01	No passo 1.3, caso ocorra erro durante a chamada do encaminhamento de *tweets* da API: 1.3.1 - Apresentar mensagem: "Não é possível conectar-se a API, impossível encaminhar o tweet"; 1.3.2 - Finalizar o caso de uso.
Pós-condições	Uma mensagem é encaminhado ao Twitter.

Quadro 8 – Detalhamento do caso de uso `Encaminhar um *tweet*`

O oitavo caso de uso (Quadro 09), denominado UC08 - Responder a um contato, permite ao telespectador iniciar uma mensagem de resposta ao contato passado como parâmetro.

UC08 – Responder a um contato : Inicia uma mensagem de resposta ao contato.	
Pré-condições	O aplicativo deve estar conectado ao Twitter. A rotina chamadora deve passar como parâmetro o nome do contato.
Cenário principal	1 - Cenário Principal 1.1 - O aplicativo limpa o campo de edição; 1.2 - O aplicativo transporta uma @ concatenando ao nome do contato parametrizado para o campo de edição; 1.3 - Executa UC05 para redação de um novo *tweet*.
Pós-condições	O aplicativo preenche o campo de edição com o nome do contato enviado.

Quadro 9 – Detalhamento do caso de uso `Responder a um contato`

3.2.2 Diagrama de classes

Nesta seção apresenta-se o diagrama de classes, que permite representar a estrutura e relações das classes que servem de modelos para os objetos definidos para a aplicação. Ao todo foi analisada a necessidade de cinco pacotes para agrupamento das classes conforme sua natureza e funcionalidade. São eles: `business`, `service`, `events`, `view` e `viewControl`.

O pacote `business` (Figura 11) contém as classes específicas ao negócio da aplicação, cada uma das classes mantém variáveis estáticas que armazenam informações específicas como os caminhos das imagens da aplicação, o mapeamento da cadeia de caracteres relacionada a cada tecla do controle remoto, tempo para atualização da listagem de *tweets* da rede, entre outras.

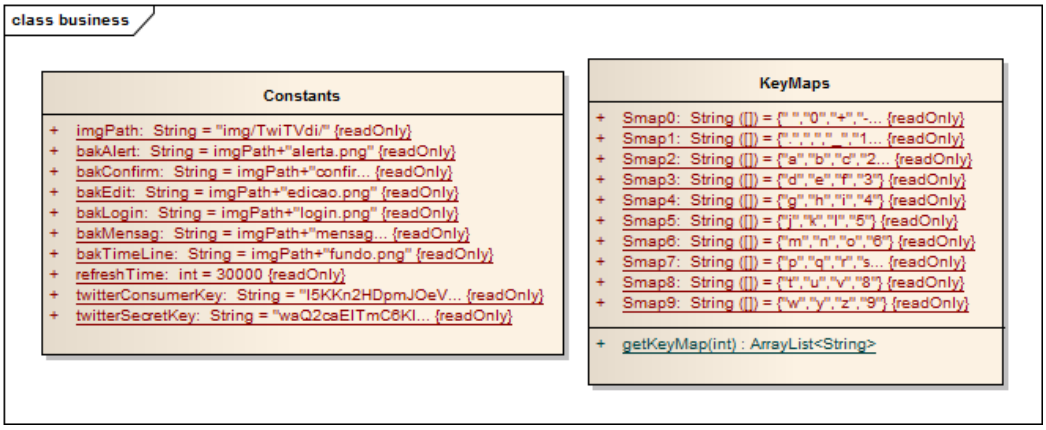


Figura 11 – Diagrama de classes do pacote `business`

Na seqüência apresenta-se o diagrama de classes do pacote `view` (Figura 12). Este pacote contém as implementações específicas da classe `com.sun.dtv.lwuit.Label`, a classe base é a `HTMLLabel`, extende as funcionalidades da `Label` e adiciona funcionalidades padrão como a agregação de código HTML no conteúdo do texto do componente para garantir a quebra de linha, não encontrada na definição padrão deste componente.

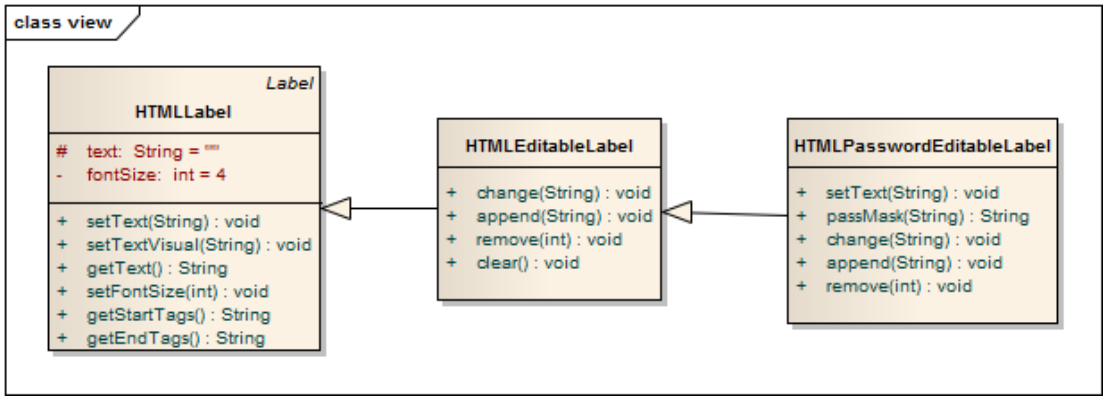


Figura 12 – Diagrama de classes do pacote `view`

Além do `HTMLLabel` criaram-se duas classes derivadas desta para simular a edição do texto do campo, simulando assim um simples *TextField*. A `HTMLEditableLabel` contém métodos padrão para a troca do último caractere de texto, adição e remoção do texto e limpeza do campo, e a `HTMLPasswordEditableLabel`, derivada da anterior, possui os mesmos métodos porém utiliza máscara padrão de senhas antes de retornar o texto a ser visualizado pelo telespectador.

O diagrama de classes do pacote `events` (Figura 13) compreende as funcionalidades para tratamento dos eventos disparados pelo telespectador durante a execução do aplicativo. A classe principal é a `KeyEventController` que extende as funcionalidades da classe `com.sun.dtv.ui.event.UserInputEventListener` e efetua o tratamento de todas as teclas pressionas pelo telespectador através do controle remoto ou do dispositivo que o mesmo esteja utilizando para interação com o aplicativo. As teclas são mapeadas e de acordo com seu evento, previsto na lista de eventos da classe `com.sun.dtv.ui.event.KeyEvent`, efetuam chamadas as funcionalidades do serviço do aplicativo.

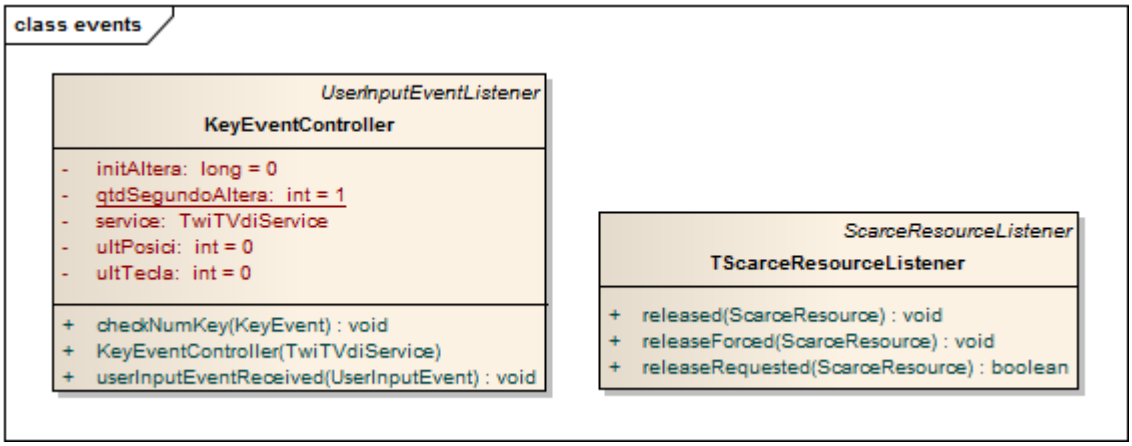


Figura 13 – Diagrama de classes do pacote `events`

O diagrama de classes do pacote `service` (Figura 14) contém as classes responsáveis pelo controle e execução dos recursos e serviço da aplicação. Este é um dos pacotes principais

e apresenta a classe inicial da aplicação, a `TwiTVdiService` onde implementa-se a classe `xjavax.tv.xlet`.

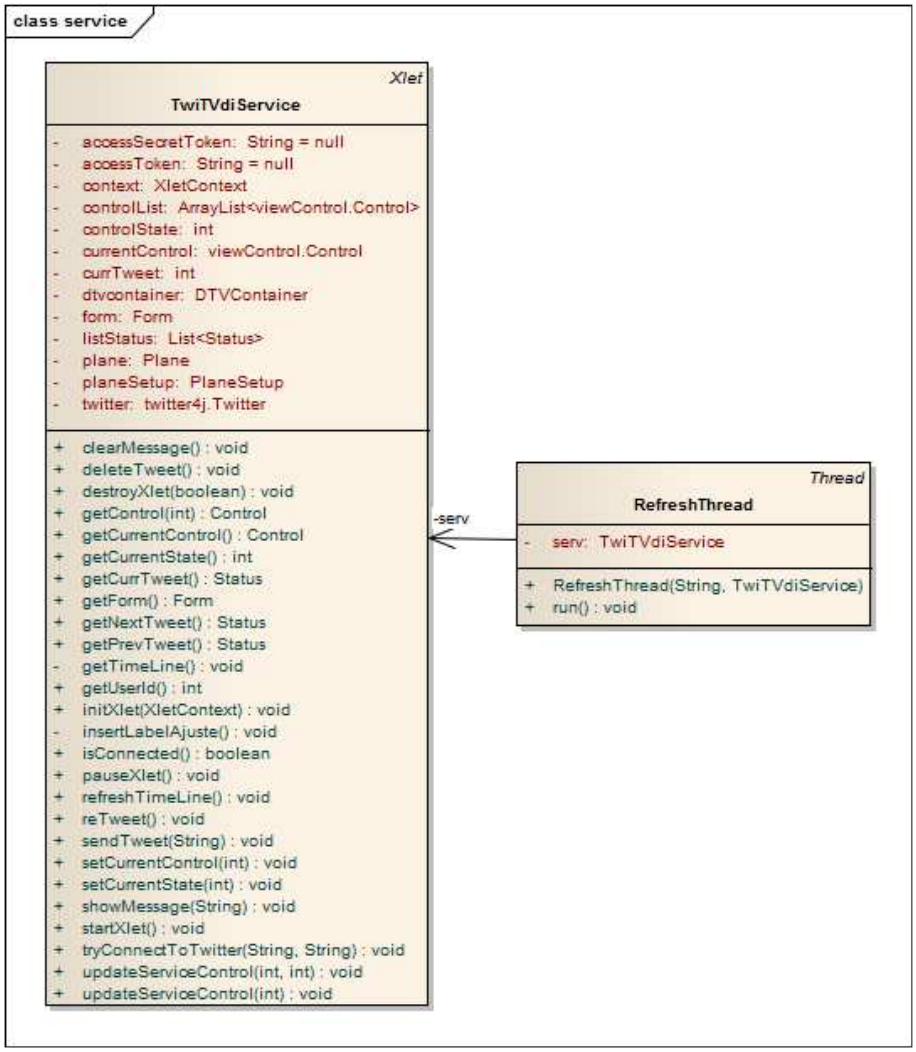


Figura 14 – Diagrama de classes do pacote `service`

A `TwiTVdiService` prevê: todos os métodos do ciclo de vida do *Xlet*, a interação com a Twitter API, a manutenção das variáveis para controle do estado da aplicação e desta forma auxiliar as classes controladores da interface, a criação dos componentes iniciais da aplicação, e por fim, o registro dos objetos para mapeamento das interações do telespectador através do controle remoto.

Além da classe inicial existe ainda a classe `RefreshThread` que consiste na implementação de uma `java.land.Thread` que é iniciada assim que o serviço conecta-se e que é disparada enquanto a aplicação estiver executando. Esta *Thread* atualiza de acordo com o tempo configurado nas variáveis de negócio a listagem dos *tweets* da rede automaticamente.

Por fim, o pacote `viewControl` (Figura 15), mantém interfaces com variáveis estáticas para a definição de tipos de estado e de controladores possíveis a aplicação. Além das interfaces citadas anteriormente, existe a classe base `Control` que representa um tipo genérico

de controlador da aplicação e que mantém funções básicas abstratas que serão implementadas em controladores específicos.

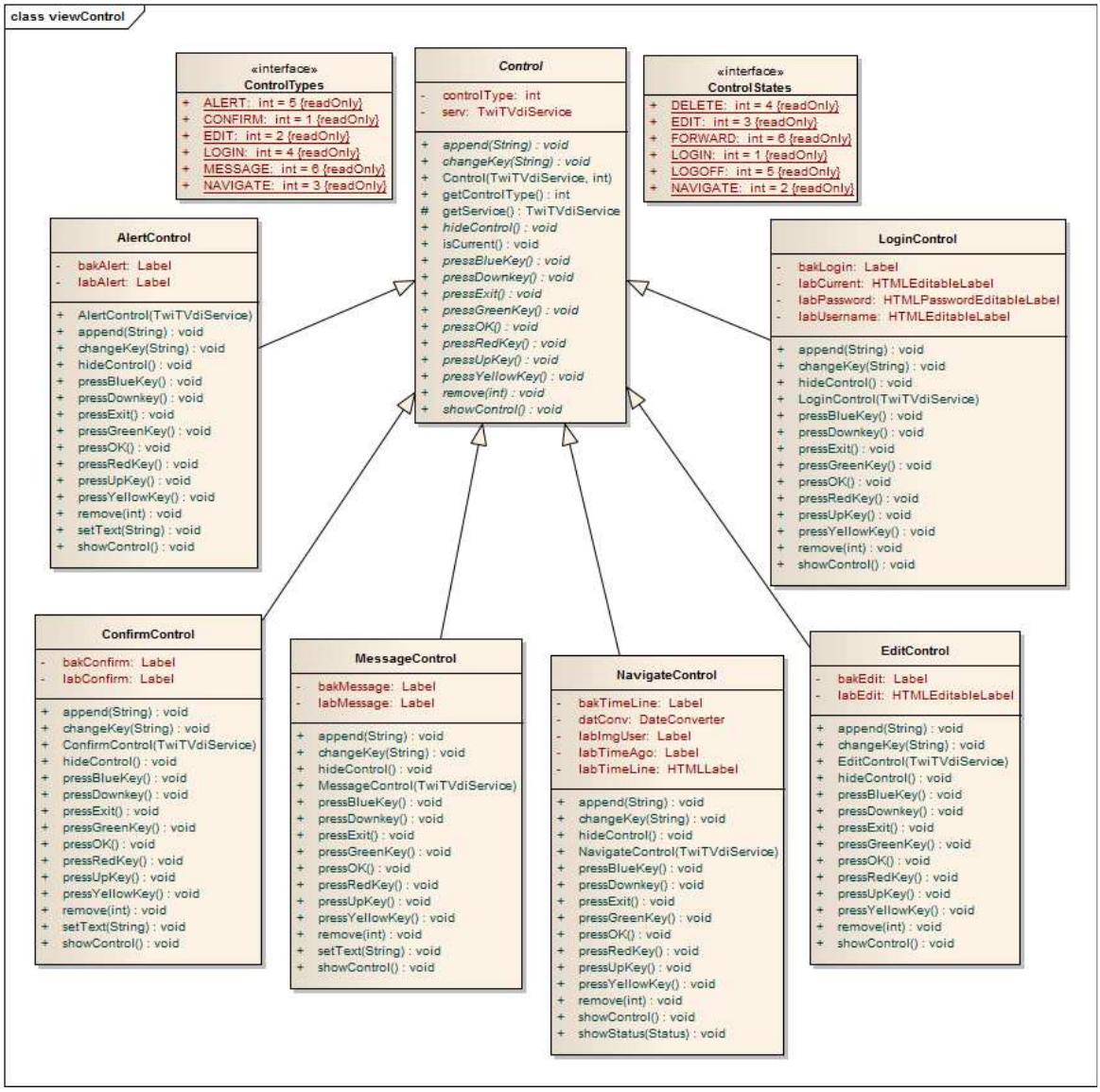


Figura 15 – Diagrama de casos de uso do aplicativo

Através dos valores atrelados ao tipo do controlador e do estado da aplicação que o aplicativo fará a chamada ao controlador específico, e dentro de cada controlador específico os métodos base serão executados e repassados ao aplicativo. Os controladores terão controle sobre o serviço do aplicativo e desta forma poderão mudar o estado atual da aplicação, renderizar imagens no aplicativo, mostrar alertas e mensagens, solicitar confirmações, chamar o processo de *login* ao Twitter e navegar na aplicação.

3.2.3 Diagrama de estados

Na figura 16 apresenta-se o diagrama de estados, onde é possível verificar todas as possíveis transações de estado e de funcionalidade efetuadas durante a execução do aplicativo. O diagrama apresenta também quais botões devem ser acionados para a transferência de estado, demonstrando desta forma o mapeamento completo da utilização das teclas por um único diagrama.

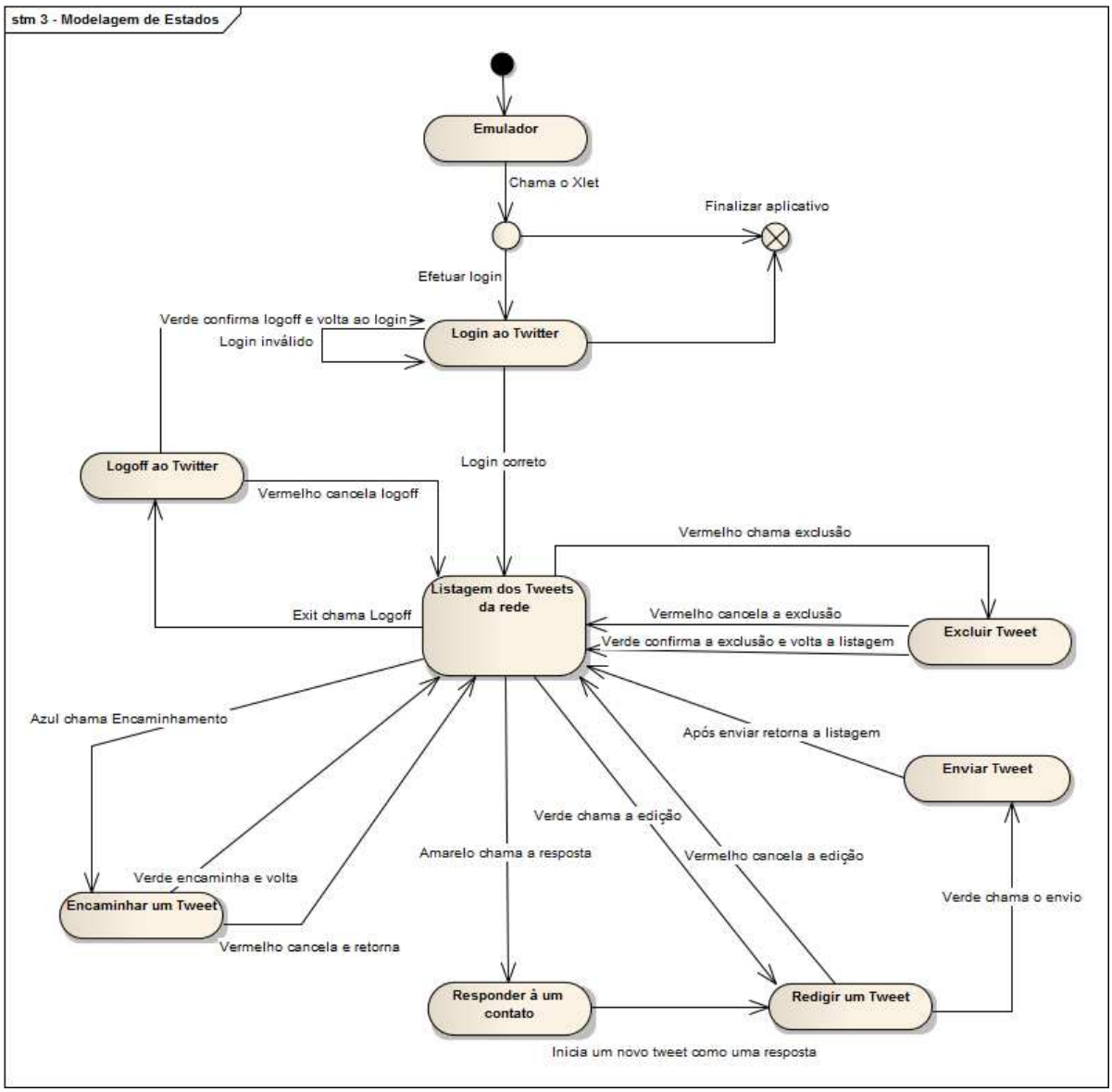


Figura 16 – Diagrama de estados do aplicativo

3.3 IMPLEMENTAÇÃO

Nesta seção são explanadas as técnicas, bibliotecas, ferramentas utilizadas e questões específicas da aplicação. Por fim, faz-se o detalhamento da operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para adequar-se à definição Ginga-J, o aplicativo foi implementado na linguagem Java, integrado com a biblioteca Twitter4j e utilizou-se o ambiente de desenvolvimento Eclipse. Para a execução dos testes e simulação do aplicativo, optou-se pelo emulador Ginga-J, da comunidade OpenGinga. Todos os detalhes dos itens supra-citados, além de detalhes quanto à interface gráfica e tratamento dos eventos, estarão descritos nas seções subsequentes.

3.3.1.1 Ambiente de desenvolvimento Eclipse

Optou-se pelo desenvolvimento através do Eclipe na versão 3.5.1 devido à facilidade na instalação, configuração e utilização. Além das vantagens citadas, o ambiente de emulação, que será apresentado nas seções seguintes, é desenvolvido e distribuído nesta e pode ser facilmente baixado, executado e integrado aos novos aplicativos que possam vir a ser desenvolvidos.

Atualmente ainda não se encontra disponível nenhuma adaptação para o desenvolvimento de aplicativos em Ginga-J no Eclipse, apenas para o Ginga-NCL, conhecido como NCL Eclipse.

3.3.1.2 Biblioteca Twitter4j para a integração com a Twitter API

A Twitter4J é uma biblioteca não-oficial para integração dos serviços da Twitter API em qualquer aplicação com suporte ao Java. É uma biblioteca independente, ou seja, não necessita de nenhuma outra biblioteca para sua execução e possuí código 100% Java.

(TWITTER4J, 2010).

A versão utilizada foi a 2.1.8, que possui código aberto regido sobre a licença *Berkeley Software Distribution* (BSD), o que considera a biblioteca como de domínio público e não apresenta restrições quanto a modificações no código desde que não seja utilizada para fins comerciais. A versão é compatível com qualquer plataforma Java de versão igual ou superior a 1.4.2, podendo ser incorporado ao projeto através da inclusão do arquivo executável *Java ARchive* (JAR) no *classpath*.

A biblioteca cria uma ponte para comunicação entre o aplicativo e as funcionalidades da Twitter API. Integra-se com as principais funcionalidades da API, contém métodos nativos para a autenticação e apresenta analisadores para o processamento do retorno das requisições em todos os formatos previsto pelo serviço. Detalhes pertinentes a autenticação e ao funcionamento da Twitter4J serão apresentados nas sub-seções seguintes.

3.3.1.2.1 Autenticação ao Twitter com a biblioteca

Para a implementação do protótipo, devido a natureza da aplicação em TV digital não possuir o redirecionamento para uma página da forma convencional como numa aplicação em um computador, ocorreu a necessidade de liberação do método de acesso xAuth.

A biblioteca Twitter4J possui suporte nativo a todos as formas de autenticação citadas na fundamentação teórica, e no caso da autenticação xAuth, o Quadro 10 apresenta a codificação do aplicativo para a devida conexão a Twitter API.

```
...
// Criando as configurações para iniciar a aplicação
ConfigurationBuilder configurationBuilder = new ConfigurationBuilder();
configurationBuilder.setOAuthConsumerKey(Constants.twitterConsumerKey);
configurationBuilder.setOAuthConsumerSecret(Constants.twitterSecretKey);

// Chamar factory para criação do objeto de controle do twitter
Configuration configuration = configurationBuilder.build();
this.twitter = new TwitterFactory(configuration).getInstance(uName,uPass);

// Guardando tokens de acesso
String accessToken = null;
String accessSecretToken = null;

// Tentar conectar com as credenciais
AccessToken token = this.twitter.getOAuthAccessToken();
accessToken = token.getToken();
accessSecretToken = token.getTokenSecret();

// Checar se conseguiu conectar
if (accessToken == null || accessSecretToken == null) {
 throw new TwitterException("Login/Senha inválidos!");
}
...
```

Quadro 10 – Utilização da xAuth para autenticação ao serviço

Caso a conexão ocorra com sucesso, a instância da classe `twitter4j.Twitter` possua os métodos de acesso e interação para com a Twitter API.

3.3.1.2.2 Integração as funções da Twitter API

Feita a conexão, a biblioteca pode utilizar diversas funcionalidades previstas para comunicar-se a Twitter API. Para o retorno das informações, por exemplo, é efetuado em diversos formatos, como o XML, o JSON ou ainda o RSSb. A Twitter4j utiliza o XML para o processamento de requisições de menor demanda, como o retorno das atualizações do usuário e de seus contatos, e a JSON para requisições numa demanda superior de informações, como as funcionalidades de busca ou das atualizações globais, ou seja, de todos os usuários do serviço.

A interface de comunicação presente na biblioteca Twitter4J utiliza os métodos POST e GET, onde as requisições são realizadas através de uma instância da classe `java.net.HttpURLConnection`. Desta forma a interação com o canal de retorno é efetuada pela própria biblioteca, abstraindo-se do aplicativo, que em alto nível utiliza apenas as chamadas aos métodos da instância `twitter4j.Twitter`, adquirida após a conexão.

O Quadro 11 abaixo representa um trecho do código do aplicativo onde a instância da classe `twitter4j.Twitter` é utilizada para o retorno das atualizações da rede do telespectador. O que ocorre é uma chamada a função `statuses/friends_timeline` que é

transparente ao aplicativo, e a Twitter4J retorna uma listagem instanciada na classe `java.util.List` com objetos do tipo `twitter4j.Status` que é próprio desta biblioteca e contém todos os atributos das atualizações do Twitter como usuário, data, texto da atualização, entre outros.

```
// Retornar o timeline
public List<Status> getUpdatedTimeLine() throws TwitterException {
  // Guardando os Tweets(Status)
  List<Status> listStatus = null;
  // Solicitando a lista de Tweets
  listStatus = this.twitter.getFriendsTimeline();
  // Retornando a lista encontrada
  return listStatus;
}
```

Quadro 11 – Codificação em nível de aplicativo para retorno das atualizações da rede

Já no Quadro 12 a seguir, encontra-se a utilização da mesma instância (citada acima) para desta vez efetuar uma atualização. Mais uma vez a chamada a função da API `statuses/update` é transparente ao aplicativo.

```
// Enviar um novo tweet
public void sendTweet(String msg) throws TwitterException{
  // Chamar funcionalidade de envio na API
  Status status = this.twitter.updateStatus(msg);
  if (status.getId() == 0) {
    throw new TwitterException
    ("Foi encontrado um erro durante o envio do Tweet!");
  }
}
```

Quadro 12 – Codificação da atualização em nível de aplicativo

3.3.1.3 Emulador Ginga-J: o emulador para simulação do aplicativo

Para o desenvolvimento, execução e testes do aplicativo optou-se pela utilização do emulador Ginga-J. Este é parte do projeto OpenGinga, e é composto de um ambiente mais simples e direcionado para o desenvolvimento de protótipos de aplicações Ginga-J. O emulador Ginga-J tem como base o emulador XletView, disponível em XletView (2003), na versão 0.3.6.

Segundo o Ginga CDN (2010a), o emulador XletView é um emulador código aberto sob licença *General Public License* (GPL). Além de possuir uma implementação de referência da API JavaTV, apresenta suporte aos padrões utilizados nos outros sistemas de TV Digital. O trabalho empregado ao projeto OpenGinga, na criação do emulador Ginga-J, consiste na substituição das bibliotecas dos padrões estrangeiros pelas bibliotecas do padrão brasileiro. Sendo uma das principais alterações a troca da biblioteca gráfica HAVI para a LWUIT e a retirada total das dependências de bibliotecas não especificadas no padrão Ginga-J.

Todo desenvolvedor interessado pode se cadastrar no projeto OpenGinga e contribuir para o desenvolvimento do emulador. O código do emulador pode ser baixado através do *Global Information Tracker* (GIT) ou um cliente compatível com esta plataforma de gerenciamento de fontes. Após a aquisição do código ele deve ser adicionado em um novo projeto do Eclipse, e a partir da configuração do projeto, o desenvolvedor pode estudar o emulador, alterando-o caso necessário e executando-o para testar outras aplicações criadas pelo mesmo. Os detalhes para a aquisição do código fonte podem ser encontrados em Ginga CDN (2010b).

O emulador pode ser executado diretamente do Eclipse, caso esteja configurado como um projeto, ou pela execução do arquivo JAR, caso o mesmo tenha sido gerado a partir do Eclipse. Após o carregamento do emulador, a interface padrão apresenta a área de menus na parte superior, o controle remoto no canto direito, e no centro a área para execução dos canais e dos aplicativos. Detalhes quanto à emulação do mesmo podem ser encontrados em Ginga CDN (2010c). A mesma pode ser visualizada conforma a Figura 17 abaixo.

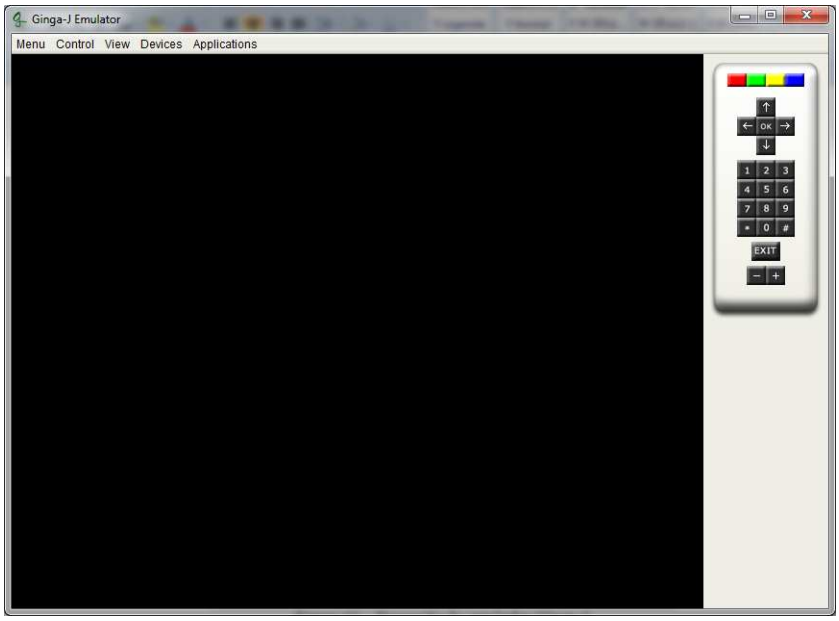


Figura 17 – Execução do emulador Ginga-J

A área de menus apresenta as seguintes opções:

a) *Menu*: no momento dispõe apenas da opção para sair da aplicação;
b) *Control*: apresenta opção para desligar o som do emulador (Mudo);
c) *View*: apresenta opção para visualização do console, funcionalidade muito útil para o desenvolvimento da aplicação;
d) *Devices*: apresenta opções para a configuração de dispositivos remotos que possam vir a ser utilizados como controle remoto alternativo, como um celular ou *Personal Digital Assistants* (PDA). Esta funcionalidade encontra-se em desenvolvimento;

e) *Applications*: apresenta opções para cadastramento de aplicações *Xlets* que possam vir a ser executadas pelo emulador, além de atalhos para a execução das aplicações cadastradas e atalho para recarregamento da aplicação atual.

A configuração do emulador Ginga-J ainda segue o mesmo padrão do *XletView*, onde existem quatro arquivos base localizados no diretório `config`:

a) `applications.xml`: arquivo XML que armazena todas as aplicações *Xlets* mantidas pelo menu Applications. A alteração deste pode ser efetuada tanto pela edição do arquivo, quanto pela seleção da opção Manage applications do menu Applications do emulador;

b) `remote_control.xml`: arquivo XML com todas as configurações para montagem do controle remoto visualizado na interface do emulador. Através deste é possível alterar imagens, posicionamento, *keycodes*, e adicionar novos botões;

c) `channels.xml`: arquivo XML que possibilita simular canais de TV pelo emulador, o emulador busca os canais cadastrados e executa como um vídeo de fundo no emulador. Está prevista a inclusão de menu para manutenção deste diretamente pelo aplicativo, não sendo mais necessário a manutenção do arquivo XML;

d) `settings.txt`: arquivo texto que mantém todas as configurações gerais do emulador, só pode ser alterado através de sua edição e compreende funções como o posicionamento e tamanho da área de vídeo do emulador, a visualização ou não do controle remoto, caminhos para bibliotecas Java externas ao emulador, entre outras.

O projeto ainda está em andamento e ainda apresenta diversas limitações principalmente na área gráfica onde ainda poucos componentes encontram-se implementados. Como exemplo de implementação pode-se citar o *Label* e o *Image*, por outro lado, no momento da implementação do protótipo ainda não é possível utilizar componentes simples como o *TextField, ComboBox, Button*, etc. O status atual do desenvolvimento pode ser checado em Ginga CDN (2010d).

3.3.1.4 Integração entre o aplicativo, a Twitter4J e o Emulador

Em um primeiro momento, para que o aplicativo *Xlet* seja corretamente compilado pelo Eclipse, se faz necessário a adição do jar contendo o código base do emulador e da biblioteca Twitter4J ao *classpath* do aplicativo. Através do Eclipse utiliza-se os métodos de

manutenção do *classpath* para adicionar o arquivo `gingaj-emulator.jar`, encontrado no diretório jars do emulador, e o arquivo `twitter4j-core-2.1.8.jar` disponível em TWITTER4J (2010) para o *classpath* do projeto deste aplicativo.

No segundo momento, para que o emulador execute corretamente as funcionalidades da Twitter API através da biblioteca Twitter4J, é necessário também copiar o arquivo JAR (citado no parágrafo anterior) da mesma para dentro do diretório *jars* contido na pasta raiz do emulador e adicioná-lo ao *classpath* do projeto deste. O simples processo de adicionar qualquer biblioteca ao *classpath* do projeto funciona para a compilação pelo Eclipse, porém ocorre erro ao executá-lo se o mesmo não encontrar a biblioteca em seu diretório padrão ou no diretório `extra.classpath` configurado no arquivo `settings.txt` (citado anteriormente).

Por fim, após a compilação e geração com sucesso do aplicativo por parte do Eclipse se faz necessário a configuração da aplicação através do menu *Applications* do emulador conforme a Figura 18 abaixo.

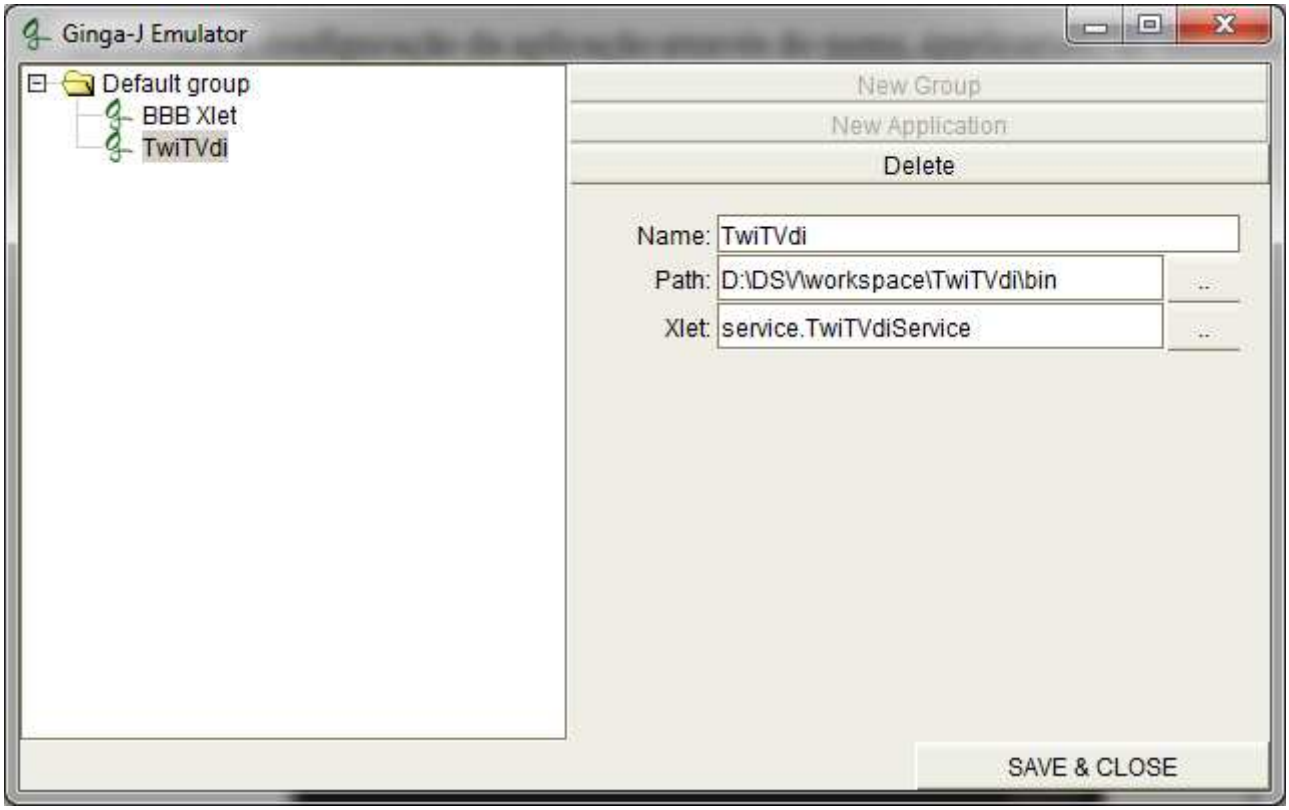


Figura 18 – Configurando a nova aplicação no emulador

Nota-se pela Figura 18, que é necessário o preenchimento no campo Path do caminho relativo da pasta bin que contém os arquivos *.class* gerado, e no campo *Xlet* deve-se indicar a classe que contém a implementação *Xlet* levando em consideração a estrutura de pacotes utilizada.

Finalmente, após a configuração da aplicação, a mesma pode ser executada através do

atalho criado no menu *Applications* conforme o campo `Name` preenchido durante a configuração supra-citada. Detalhes sobre a execução da aplicação pelo emulador podem ser encontrados em Ginga CDN (2010e).

3.3.1.5 Interface Gráfica e interação com o controle remoto

A classe `com.sun.dtv.ui.DTVContainer` é responsável por manter a camada gráfica da aplicação. É criada durante a inicialização do *Xlet* tendo como base métodos para captura dos planos de acordo com o dispositivo onde o aplicativo esteja sendo utilizado. Feita a criação da camada gráfica cria-se o componente principal da área gráfica, representado pela classe `com.sun.dtv.lwuit.form` e que possui características semelhantes ao `java.awt.Frame` de aplicações Java. O código para a criação dos objetos das classes citadas pode ser vislumbrado no Quadro 13 abaixo.

```
public void startXlet() throws XletStateChangeException {
  // Inicializar captura do dispositivo de acordo com detalhes do dispositivo
  Device device = Device.getInstance();
  Screen currentScreen = device.getDefaultScreen();
  ...
  // Capturando todos os planos da tela corrente
  Plane[] planes = currentScreen.getAllPlanes();
  for (int i = 0; i < planes.length; i++) {
    Capabilities cap = planes[i].getCapabilities();
    if (cap.isGraphicsRenderingSupported()) {
      plane = planes[i];
      planeSetup = plane.getCurrentSetup();
      break;
    }
  }
  // reserva o plano para o containner gráfico
  try {
    plane.reserve(false, -1, new TScarceResourceListener());
  } catch (Exception e) {
    e.printStackTrace();
  }
  // Inicializando o formulário
  form = new Form();
  form.setLayout(null);
  form.getContentPane().setLayout(null);
  form.setSize(planeSetup.getScreenResolution());
  dtvcontainer.addComponent(form);
  // Inicializando o containner
  dtvcontainer = DTVContainer.getDTVContainer(plane);
  dtvcontainer.setLayout(null);
  dtvcontainer.setSize(planeSetup.getScreenResolution());
  // setando a visibilidade
  dtvcontainer.setVisible(true);
  // inicializando componentes da tela
  form.show();
  form.repaint();
  this.initComponents();
}
```

Quadro 13 – Codificação para criação do componente gráfico e formulário

Nota-se pelo quadro acima que para a criação do componente gráfico utiliza-se métodos de construção da própria classe partindo de um plano passado. Após a criação do componente gráfico, todos os outros componentes necessários a aplicação devem ser adicionados ao formulário através do método `addComponent()`.

No próximo exemplo (Quadro 14) constata-se a utilização da funcionalidade de criação e adição de componentes ao `Form` na implementação da classe específica de *login* em `viewControl.Control`. O exemplo demonstra a criação de um `view.HTMLLabel` para representar o fundo do formulário de *login*.

```
...
 // Criando os componentes para visualização
 this.bakLogin = new Label();
 try {
       this.bakLogin.setIcon(Image.createImage(Constants.bakLogin));
 } catch (IOException e) {
       System.out.println("Error while loading image " + e.getMessage());
 }
 // Setando o posicionamento
 this.bakLogin.setX(230);
 this.bakLogin.setY(180);
 this.bakLogin.setSize(new Dimension(300, 234));
 this.bakLogin.setVisible(false);
 // Adicionando no formulário
 super.getService().getForm().addComponent(this.bakLogin);
 ...
```

Quadro 14 – Codificação para montar um fundo na tela de *login*

Após o entendimento da criação dos componentes visuais, é necessário entender o funcionamento da entrada de informações por meio dos botões do controle remoto. Conforme já foi citado no diagrama de classes do pacote `events`, a filtragem das teclas são tratadas pelos eventos da classe `java.awt.event.KeyEvent` na classe `view.KeyEventController` e os eventos tratados no método padrão da interface, o `userInputEventReceived`. Independente do estado atual da aplicação esta classe segue dois fluxos de filtragem, o primeiro onde as teclas principais como as setas, os botões coloridos são tratados e diretamente enviam um sinal ao controlador atual do serviço indicando que determinada tecla foi pressionada. Um exemplo pode ser observado no Quadro 15.

```
public void userInputEventReceived(UserInputEvent inputEvent) {
  // Guardando a tecla clicada
  KeyEvent event = (com.sun.dtv.ui.event.KeyEvent) inputEvent;
  switch (event.getKeyCode()) {
  case KeyEvent.VK_UP:
   service.getCurrentControl().pressUpKey();
   break;
  case KeyEvent.VK_DOWN:
   service.getCurrentControl().pressDownkey();
   break;
  case RemoteControlEvent.VK_COLORED_KEY_0:
   service.getCurrentControl().pressRedKey();
   break;
  case RemoteControlEvent.VK_COLORED_KEY_1:
   service.getCurrentControl().pressGreenKey();
   break;
  case RemoteControlEvent.VK_COLORED_KEY_2:
   service.getCurrentControl().pressYellowKey();
   break;
  case RemoteControlEvent.VK_COLORED_KEY_3:
   service.getCurrentControl().pressBlueKey();
   break;
  case 27:
   service.getCurrentControl().pressExit();
   break;
  case KeyEvent.VK_ASTERISK:
   service.getCurrentControl().remove(1);
   break;
  default:
   checkNumKey(event);
   break;
  }
 }
```

Quadro 15 – Codificação do tratamento das teclas padrão

Ao se observar novamente o Quadro 15, quando o evento pressionado não corresponder às teclas padrão, ele entra a opção padrão da instrução, ou seja, a filtragem das teclas numéricas do controle remoto. Este processo é efetuado no método `checkNumKey` e pode ser observado no Quadro 16.

```
public void checkNumKey(KeyEvent KeyCode) {
  // Se a tecla estiver entre 0 e 9
  if(KeyCode.getKeyCode()>=KeyEvent.VK_0 &&
KeyCode.getKeyCode()<=KeyEvent.VK_9){
   // Guardar o array que deve ser utilizado
   ArrayList<String> map = KeyMaps.getKeyMap(KeyCode.getKeyCode());
   // Guardar a hora da alteração atual
   long endAltera = System.currentTimeMillis();
   // Testar se existiu alteração anterior ou se mudou de tecla
   if(initAltera>0&&ultTecla==KeyCode.getKeyCode()){
    // Calcular a diferença
    long diff = endAltera - initAltera;
    // Se a diferença expirou a quantidade de segundos parametrizada
    if((diff/1000)>=qtdSegundoAltera){
     //Adiciona o novo caracter
     service.getCurrentControl().append(map.get(0));
     ultPosici = 0;
    }else{
     // guardar o caracter que será trocado
     String newS = "";
     //Se a ultima posição + 1 estourar o tamanho do array atual
     if(ultPosici==map.size()-1){
      //Pegar o primeiro caracter
      newS = map.get(0);
      ultPosici = 0;
     }else{
      //Adiciona o proximo caracter da tecla
      newS = map.get(ultPosici+1);
      ultPosici+=1;
     }
     // Substitui o ultimo caracter digitado pelo novo
     service.getCurrentControl().changeKey(newS);
    }
   }
   else{
    // Apenas adiciona o primeiro caracter do mapa atual
    service.getCurrentControl().append(map.get(0));
    ultPosici = 0;
   }
   // Guardar a tecla pressionada
   ultTecla = KeyCode.getKeyCode();
   // Zerar a variável de alteração
   initAltera = System.currentTimeMillis();
  }
 }
```

Quadro 16 – Codificação do tratamento das teclas numéricas

O método demonstrado no quadro é executado somente caso o evento pressionado esteja no intervalo das teclas numéricas do controle remoto. O próximo passo que o método executa é verificar se o telespectador apertou o mesmo botão pressionado anteriormente, e em caso positivo, se a alteração está dentro de um período de dois segundos, isto é necessário para que ocorra a troca do último caractere digitado pelo próximo caractere da cadeia relacionada ao botão do controle. Caso o tempo tenha sido superior a dois segundos ou a tecla diferente da anteriormente pressionada, o aplicativo apenas adiciona o primeiro caractere do mapa relacionado à tecla atual.

Cada um dos métodos de interação chama diretamente o serviço do aplicativo ou pede para o mesmo a instância do controlador atual, e a partir deste controlador efetuam chamadas

genéricas. A partir desta chamada o controlador toma as ações necessárias de acordo com sua implementação, e em alguns casos, apesar da chamada aos métodos padrão, a instância controladora não executa ação nenhuma, como por exemplo o pressionamento das teclas numéricas em controladores que não possuem campos editáveis.

3.3.2 Operacionalidade da implementação

Nesta seção apresenta-se o funcionamento e o ciclo de vida da aplicação durante a interação com o telespectador. Ao escolher a inicialização do aplicativo, que é gerenciado pelo emulador no caso demonstrado, a tela de *login* ao Twitter é apresentada conforme a Figura 19. A tela apresenta dois campos para o preenchimento das credenciais para o acesso, o *login* e senha, e dois botões relacionados aos botões coloridos do controle remoto para a tentativa de *login* ou cancelamento do mesmo.

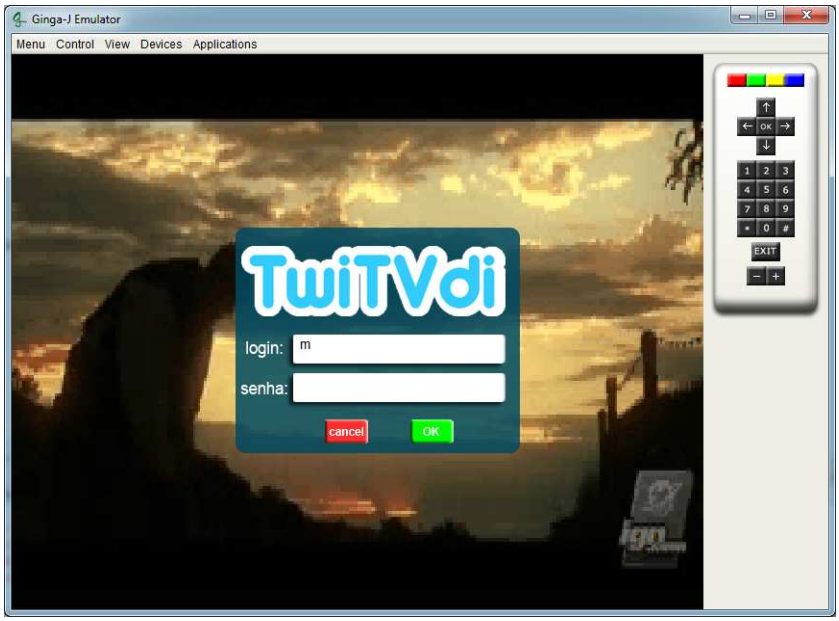


Figura 19 – Tela de *login* ao aplicativo

Para informar o *login* e a senha, o telespectador seleciona um dos dois campos com as setas para cima e para baixo do controle remoto, e a partir da seleção do campo, o telespectador utiliza o teclado numérico para preenchimento das informações. Este processo assemelha-se ao preenchimento de um SMS em um celular.

Ao término do preenchimento, o telespectador tenta validar o acesso clicando no botão verde. Neste ponto ocorrem as validações, primeiramente dos campos obrigatórios, e em caso do esquecimento do preenchimento de alguma informação, o aplicativo lista uma mensagem na área superior da tela informando sobre o não preenchimento do campo obrigatório (Figura

20). Desta forma o telespectador é alertado e pode corrigir as credenciais para tentar novamente o acesso. Após as primeiras validações, o aplicativo tenta a conexão ao serviço da Twitter API, e novamente em caso de erro o alerta é mostrado na área superior da tela.

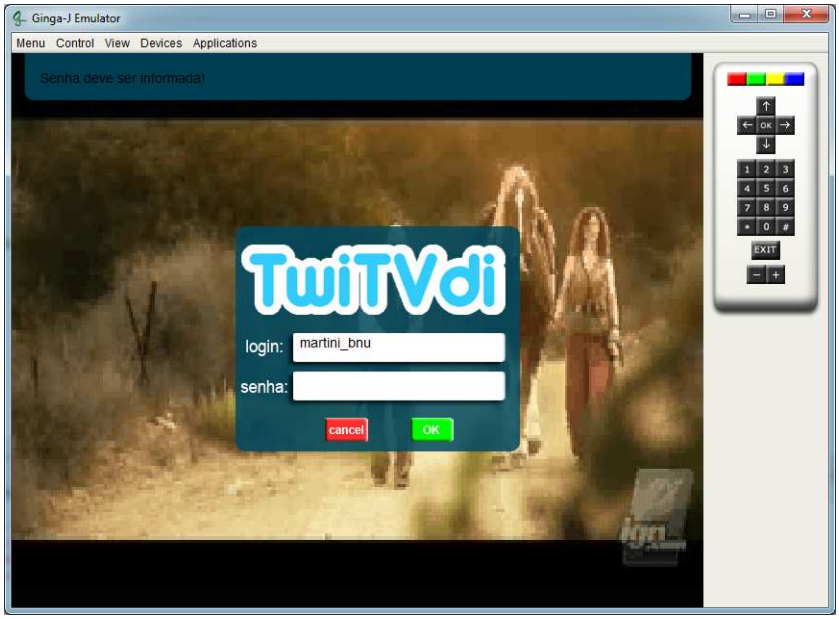


Figura 20 – Mensagem de erro no *login*

Após a conexão com sucesso ao serviço, ocorre o carregamento das últimas 20 atualizações da rede de contatos, e então o painel de visualização das atualizações é mostrado na área inferior do aplicativo conforme a figura 21. O painel apresenta uma atualização por vez, o telespectador pode então navegar entre as atualizações pelo clique nas setas para cima e para baixo do controle remoto. Todas as atualizações apresentam a imagem do contato que a enviou, o seu nome de usuário no serviço, o conteúdo e a linha de tempo, ou seja, há quanto tempo esta atualização foi enviada.

Figura 21 – Aplicativo inicializado com a primeira atualização

O painel de navegação apresenta também quatro botões que servem para a interação com as atualizações do serviço: verde para o envio de uma nova atualização, vermelho para a exclusão de uma atualização enviada pelo próprio telespectador, amarelo para iniciar uma mensagem de resposta a um contato, e por fim, azul para o encaminhamento de uma atualização que o telespectador tenha achado interessante.

Ao clicar nos botões vermelho ou azul, ocorre o processo de confirmação da operação por parte do telespectador (Figura 22). Para confirmação o telespectador usará o botão verde e o vermelho para a não confirmação da operação. É importante salientar que em caso da exclusão, se o telespectador esteja posicionado sobre uma atualização que não seja de sua autoria, ocorre o envio de mensagem de alerta para que o mesmo esteja ciente de que não pode excluir uma atualização que não tenha sido efetuada por ele (Figura 23).

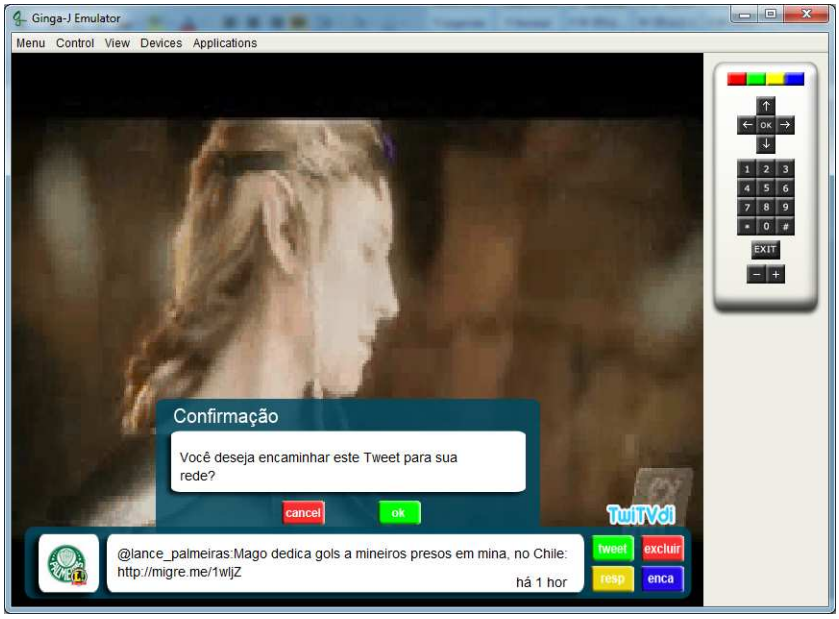


Figura 22 – Confirmação de encaminhamento de *tweet*

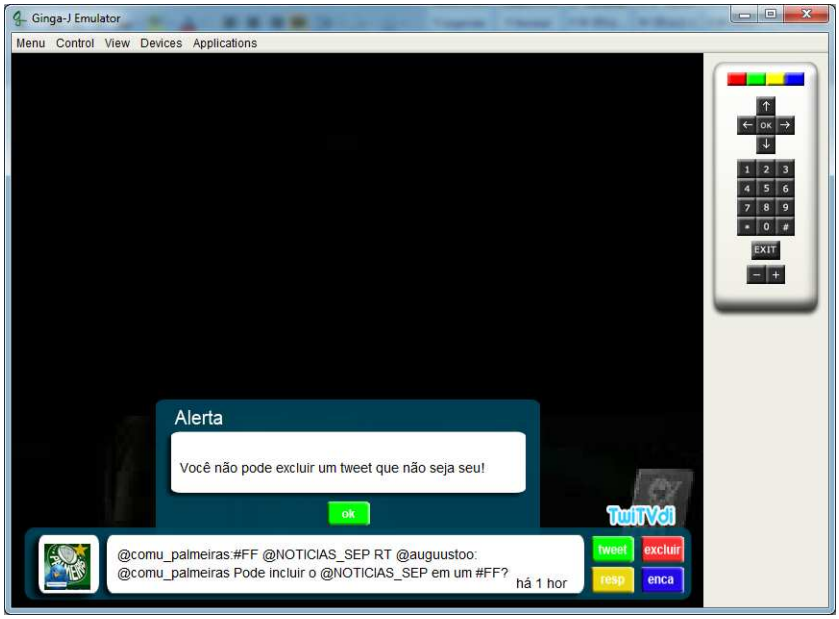


Figura 23 – Alerta ao telespectador

Completando as funcionalidades de interação, os botões verde e amarelo abrem a janela de edição para redação e envio de uma nova mensagem (Figura 24). No caso da resposta a janela é aberta com o início com a citação do nome do contato do *tweet* selecionado atualmente, caracterizando uma resposta, ou em termos do Twitter, uma menção ao contato. Novamente para a composição da mensagem a ser enviada o telespectador utiliza as teclas numéricas do controle remoto.

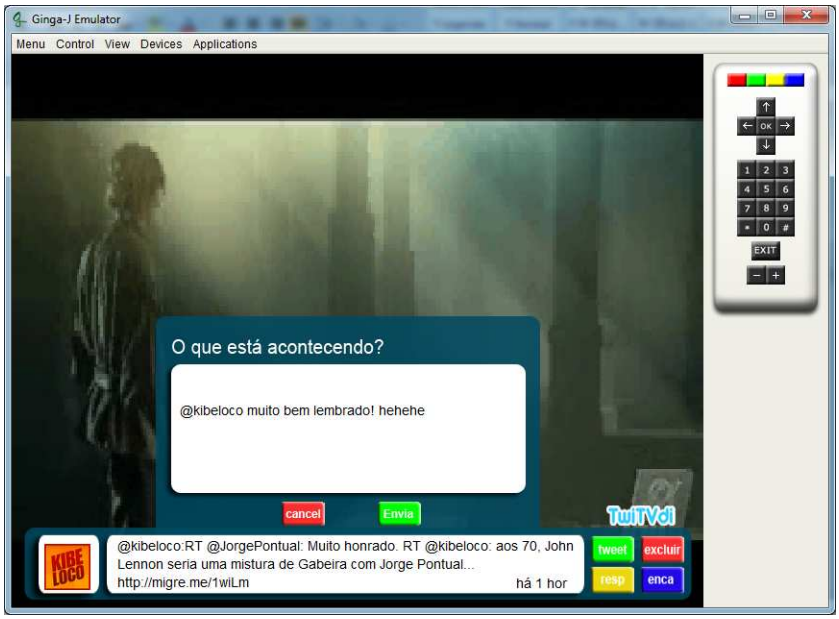


Figura 24 – Edição de um *tweet* de resposta

Após o envio de uma atualização, ou a exclusão de um *tweet*, ou ainda caso haja um período de 30 segundos de inatividade, as últimas 20 atualizações da rede são novamente carregadas e o painel de visualização atualizado com a última atualização da rede.

Por fim, caso o usuário utilizar a tecla *exit* no controle, o processo de *logoff* da conta do Twitter é iniciado e a aplicação novamente efetua o processo de confirmação conforme a Figura 25. E em caso de confirmação com o botão verde a aplicação é desconectada da conta e o processo de *login* descrito no começo dessa seção é chamado novamente.

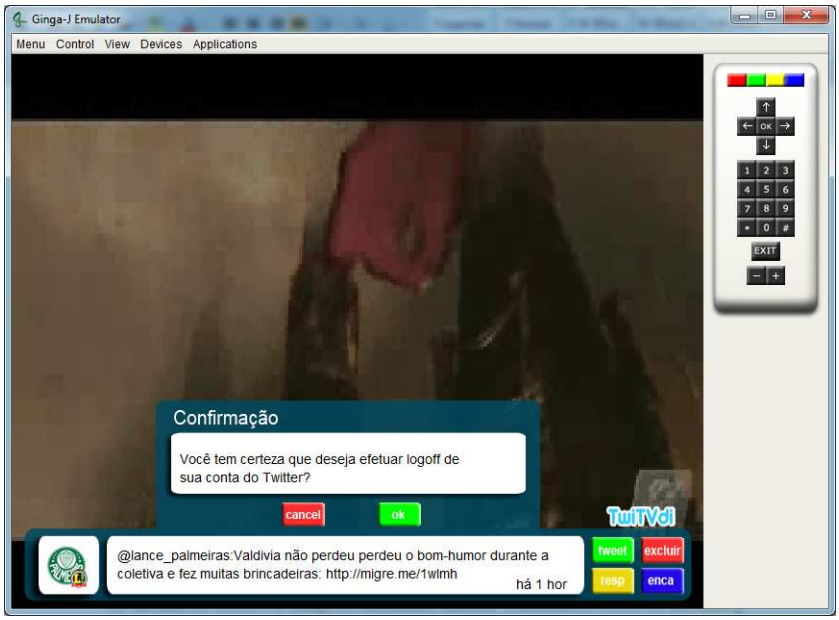


Figura 25 – Confirmação de *logoff* ao Twitter

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos com o término do trabalho foram considerados muito satisfatórios. Com o desenvolvimento deste aplicativo foi possível provar a utilização da norma Ginga-J para desenvolvimento de aplicativos em TVDI. O correto funcionamento do aplicativo durante a execução do mesmo pelo emulador Ginga-J é uma das provas da conformidade do aplicativo para com a norma.

Ao observar a operacionalidade do aplicativo, pode-se concluir que o aplicativo utiliza uma navegação simples e objetiva, em que na maioria das interações os botões coloridos são explorados e sugeridos pela observação a tela onde o aplicativo está rodando.

Durante o amadurecimento da especificação do aplicativo, formulado em primeiro momento durante a proposta deste, decidiu-se pela alteração em alguns requisitos funcionais, onde o detalhamento das atualizações consolidou-se em um único requisito funcional, e pela troca da representação de diagrama de objetos pela representação através de diagrama de casos de uso, pois este melhor se adequava a aplicação.

Já durante a implementação da aplicação, constatou-se que, dentre todas as funcionalidades, a operação de navegação entre as atualizações da rede apresentou o maior consumo de recursos devido ao *download* da imagem de cada contato. Porém, mesmo aplicando exaustivos testes de navegação, a aplicação apresentou um baixo consumo de memória, em torno de 30 *MegaBytes* (MB).

Em redes com baixa velocidade de conexão, novamente a operação que consome o maior tempo é a navegação. Ao desconsiderar o processo de *download* da imagem dos contatos, o tempo de resposta aumenta consideravelmente e a troca entre as atualizações é instantânea desde que a conexão possua uma velocidade mínima de 256 *KiloBytes* (KB).

Infelizmente durante os testes e validações não se encontrou um dispositivo físico de preço acessível e compatível com a norma Ginga-J para a execução da aplicação. Os testes então, foram aplicados unicamente sobre o emulador Ginga-J da comunidade OpenGinga.

Com relação aos trabalhos correlatos, Andreata (2006) apresentou diversos módulos interativos, como enquetes e *chats*, todos observando a utilização do canal de retorno para que a aplicação se adequasse a um ambiente real de TVDI. Apesar da especificação ser compatível apenas com a camada de software MHP, o trabalho se mostrou bastante proveitoso e serviu como uma ótima base conceitual.

Já Schroder (2007) interagiu com um provedor de serviço de difusão, através do canal de retorno e receber informações e notícias no formato RSSb. Este trabalho possui uma estreita relação conceitual com o presente trabalho, pois tem o mesmo conceito de manter o telespectador sintonizado com sua rede virtual.

Por fim, Cruz (2008) demonstrou em diversos testes que os emuladores de dispositivos portáteis Symbian apresentavam resultados corretos e eficientes, sem necessidade de se realizar qualquer modificação na especificação Ginga.

Com exceção de Cruz (2008), os outros trabalhos foram desenvolvidos em um período onde o SBTVD não possuía um nível de maturidade suficiente para ser utilizado como sistema de referência, e utilizou-se então o ambiente MHP. O presente trabalho tem um enfoque específico no padrão Ginga-J, e desta forma, se torna livre e pode ser utilizado como referência para o desenvolvimento de aplicações seguindo a especificação recém liberada e homologada nos padrões da ABNT. O trabalho de Cruz (2008) não segue o domínio procedural, porém serviu de base para provar que um dos principais requisitos do Ginga, a portabilidade, pode ser implementada e viabilizada.

4 CONCLUSÕES

De forma geral, o trabalho desenvolvido atingiu o objetivo de prover um aplicativo *Xlet* para TVDI de acesso ao Twitter e compatível com a especificação Ginga-J. A interação através da utilização do canal de retorno enquadrou-se na especificação. Todos os objetivos específicos foram atingidos com sucesso.

A escolha pela utilização do emulador desenvolvido pelo projeto OpenGinga se deu pela possibilidade do mesmo ser desenvolvido por membros experientes no SBTVD, e inclusive contar com desenvolvedores que fizeram parte do projeto de especificação do *middleware* Ginga. Além destes, o projeto é livre, pode ser estudado por qualquer desenvolvedor interessado e despertou-me interesse em colaborar com o projeto.

Durante todo o desenvolvimento do trabalho, deparei-me com diversas dificuldades. A grande maioria devido à falta de material didático. Ainda existem poucos livros que abordam o SBTVD, e quando são encontrados, falam na grande maioria das vezes sobre o Ginga-NCL. Toda essa falta de conteúdo Ginga-J deve-se principalmente pelas questões burocráticas e de cobranças, que inclusive foram tratadas na fundamentação teórica deste.

Outra grande dificuldade foi devido a falta de suporte aos componentes gráficos da norma pelo emulador. Fez-se necessário a criação de componentes derivados do *Label* para simular *Panels*, *TextFields*, *PasswordFields* e *Buttons*. Apesar da utilização dos componentes citados não gerar erro durante a compilação, na execução os mesmos não eram apresentados. Mesmo com a criação de componentes, o protótipo apresenta a possibilidade de adição de novas funcionalidades e troca dos componentes criados por componentes prontos e que podem ser adicionados tão logo que o emulador possua suporte. Além destes, a utilização da Twitter4J deixa a aplicação flexível a ponto de receber muitas outras funcionalidades da Twitter API sem a necessidade de grandes alterações em seu código fonte.

Por fim, posso concluir que o trabalho demonstra o grande poder da junção TV e Internet, quando deixaremos de ser meros espectadores, e assim como acontece hoje na própria Internet, ajudaremos a construir o conteúdo e reaprenderemos a utilizar o controle remoto para explorar cada vez mais o conteúdo enviado para nossa residência.

4.1 EXTENSÕES

Existem diversas melhorias que podem ser efetuadas partindo do aplicativo deste trabalho, as quais são:

a) trocar os componentes gráficos criados através do *Label* para componentes prontos como *TextField*, *Button* e *List* assim que o emulador possuir o suporte necessário;
b) desenvolver métodos de configuração da aplicação para que o usuário possa escolher questões como o tempo de atualização, a posição do aplicativo no dispositivo, e gravação de senha por exemplo;
c) implementar novas funcionalidades, desde que relevantes quanto a natureza do aplicativo, presentes na Twitter API e que podem ser facilmente integradas graças a utilização da biblioteca Twitter4J que deixa a aplicação flexível e facilmente expansível.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDREATA, Jomar A. **InteraTV**: um portal para aplicações colaborativas em TV digital interativa utilizando plataforma MHP. 2006. 110 f. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Santa Catarina, Florianópolis.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Projeto 00:001.85-006/4**: informação e documentação. Rio de Janeiro, 2007.

BRACKMANN, Christian P. **Usabilidade em TV digital**. 2010. 198 f. Dissertação (Mestrado em Ciência da Computação) – Centro Politécnico, Universidade Católica de Pelotas, Pelotas.

COSTA, Aécio L. V. C.; MELO JR., Mozart. **Desenvolvimento de aplicativos para TV digital:** Comparativo entre módulos do Ginga. Maceió, 2009. Disponível em: <http://www.cesmac.com.br/erbase2010/papers/wticg/65547.pdf>. Acesso em: 12 set. 2010.

CRUZ, Vitor M. **Ginga-NCL para dispositivos portáteis**. 2008. 84 f. Dissertação (Mestrado em Informática) – Centro Técnico Científico, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

CRUZ, Vitor M.; MORENO, Marcio F.; SOARES, Luiz F. G. **TV Digital Para Dispositivos Portáteis –** Middlewares. 2008. 69 f. Monografia (Bacharelado em Ciência da Computação) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

DAMASIO, Leandro. **TV digital**. [Florianópolis], 2007. Disponível em: <http://democratasdigitais.blogspot.com/2007/12/tv-digital.html>. Acesso em: 13 ago. 2010.

DEMONIO DE MAXWEL. **TV digital interativa**. [S.l], 2009. Disponível em: <http://demoniodemaxwell.wordpress.com/2009/01/01/tv-digital-interativa/>. Acesso em: 25 ago. 2010.

FÓRUM DO SISTEMA BRASILEIRO DE TV DIGITAL TERRESTE. [S.l.], 2009. Disponível em: <http://www.forumsbtvd.org.br/materias.asp?id=74>. Acesso em 24 ago. 2010.

GINGA. **Sobre o Ginga**. [S.l.], 2008. Disponível em: <http://www.ginga.org.br>. Acesso em 16 ago. 2010.

GINGADF. **O que é o OpenGinga**? [Brasília], 2009. Disponível em: <http://www.gingadf.com/blogGinga/?p=42>. Acesso em: 8 mar. 2010.

GINGA CDN. **Emulador Ginga-J:** visão geral. [João Pessoa], 2010a. Disponível em: <http://dev.openginga.org/projects/gingaj-emulator>. Acesso em: 1 out. 2010.

____. **Emulador Ginga-J:** usando o GIT. [João Pessoa], 2010b. Disponível em: <http://dev.openginga.org/projects/gingaj-emulator/wiki/Usando_o_git>. Acesso em: 3 out. 2010.

____. **Emulador Ginga-J:** execução do emulador no Eclipse. [João Pessoa], 2010c. Disponível em: <http://dev.openginga.org/projects/gingaj-emulator/wiki/Execu%C3%A7%C3%A3o_do_Emulador_no_Eclipse>. Acesso em: 4 out. 2010.

____. **Emulador Ginga-J:** status atual. [João Pessoa], 2010d. Disponível em: <http://gingacdn.lavid.ufpb.br/projects/gingaj-emulator/wiki/Status_atual>. Acesso em: 10 out. 2010.

____. **Emulador Ginga-J:** execução de uma aplicação pelo emulador. [João Pessoa], 2010e. Disponível em: <http://dev.openginga.org/projects/gingaj-emulator/wiki/Execu%C3%A7%C3%A3o_de_uma_aplica%C3%A7%C3%A3o_pelo_Emulador>. Acesso em: 3 out. 2010.

LANG, Jean P. **O que é o MOStvd?** [São Paulo], 2009. Disponível em: <http://redmine.middlewareopensource.com/wiki/middlewareopensource>. Acesso em: 10 mar. 2010.

MARTINS, Ivan; LEAL, Renata. **O Twitter vê e mostra tudo**: um serviço global de mensagens rápidas desafia os hábitos de comunicação e reinventa o conceito de privacidade. [São Paulo], 2009. Disponível em: <http://revistaepoca.globo.com/Revista/Epoca/0,,EMI63823-15228,00-O+TWITTER+VE+E+MOSTRA+TUDO.html>. Acesso em: 21 mar. 2010.

OPREH. **Nostalgia total**. [S.l.], 2010. Disponível em: <http://opreh.com.br/geral/nostalgia-total-%E2%80%93-02-sem-chororo/>. Acesso em: 20 ago. 2010.

PAULINELLI, Fernanda; KULESZA, Raoni. **Histórico Ginga-J.** João Pessoa, 2009. Disponível em: <http://dev.openginga.org/projects/gingaj-emulator/wiki/Hist%C3%B3rico_Ginga-J>. Acesso em: 24 ago. 2010.

PAWLAN, Monica. **Introduction to digital TV applications programming.** [S.l], 2001. Disponível em: <http://java.sun.com/developer/technicalArticles/javatv/apiintro/>. Acesso em: 24 ago. 2010.

PORTAL DO SOFTWARE PÚBLICO BRASILEIRO. [S.l.], 2009. Disponível em: <http://www.softwarepublico.gov.br/dotlrn/clubs/ginga/file-storage/view/public/posters/Poster2.pdf>. Acesso em: 24 ago. 2010.

SAMPAIO, Lucas D. H. **Análise de middlewares no contexto de sistemas terrestres de televisão digital.** 2008. 58f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Computação, Universidade Estadual de Londrina, Londrina.

SANTOS, Adriana C. O. **Reflexões sobre a convergência tecnológica:** A TV digital interativa no Brasil. [São Paulo], 2002. Disponível em: <http://www.bocc.ubi.pt/pag/santos-adriana-tv-digital-interactiva-no-brasil.pdf>. Acesso em: 22 ago. 2010.

SCHROEDER, Everton. **Desenvolvimento de servidor de RSS para TV digital interativa**. 2007. 73 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOARES, Luiz F. G.; BARBOSA, Simone D. J. **Programando em NCL 3.0**: desenvolvimento de aplicações para o middleware Ginga. Rio de Janeiro: Elsevier 2009.

STRICKLAND, Jonathan. **Como funciona o Twitter**. Tradução HowStuffWorks Brasil. [São Paulo], 2010. Disponível em: <http://informatica.hsw.uol.com.br/twitter.htm>. Acesso em: 01 abr. 2010.

SUN MICROSYSTEMS. [S.l.], 2010. Disponível em: <http://br.sun.com>. Acesso em: 25 ago. 2010.

TELECO. **Estatísticas de domicílios brasileiros (IBGE-PNAD)**. São José dos Campos, 2009. Disponível em: <http://www.teleco.com.br/pnad.asp>. Acesso em: 22 mar. 2010.

TWITTER DEVELOPERS. [S.l], 2010a. Disponível em: <http://dev.twitter.com/pages/every_developer>. Acesso em: 27 jul. 2010.

____. [S.l], 2010b. Disponível em: <http://dev.twitter.com/pages/rate-limiting>. Acesso em: 12 out. 2010.

____. [S.l], 2010c. Disponível em: <http://dev.twitter.com/doc/get/statuses/friends_timeline>. Acesso em: 14 out. 2010.

____. [S.l], 2010d. Disponível em: <http://dev.twitter.com/doc>. Acesso em: 15 out. 2010.

TWITTER4J. [S.l], 2010. Disponível em: <http://twitter4j.org/en/index.jsp>. Acesso em: 30 jul. 2010.

UCHOA, Daniel da C. **Finalmente sai norma Ginga-J**. São Paulo, 2010. Disponível em: <http://www.overmedianetworks.com.br/noticia-overmedia-consulta-nacional-gingaj.html>. Acesso em: 22 mar. 2010.

XLETVIEW. [S.l.], 2003. Disponível em: <http:// www.xletview.org>. Acesso em: 18 out. 2010.

WAISMAN, Thais. **TV digital interativa na educação:** afinal, interatividade para que? São Paulo, 2002. Disponível em: <http://www.abed.org.br/congresso2002/trabalhos/texto25.htm>. Acesso em: 13 ago. 2010.