UNIVERSIDADE FEDERAL DE SANTA CATARINA

GINGA SOCIAL:

SISTEMA DE INTEGRAÇÃO ENTRE

TV DIGITAL INTERATIVA E FACEBOOK

FIODOR CASTRO

GUSTAVO RUPP SANTOS

FLORIANÓPOLIS - SC

2012/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

GINGA SOCIAL:

SISTEMA DE INTEGRAÇÃO ENTRE

TV DIGITAL INTERATIVA E FACEBOOK

FIODOR CASTRO

GUSTAVO RUPP SANTOS

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

FLORIANÓPOLIS - SC

2012/2

FIODOR CASTRO

GUSTAVO RUPP SANTOS

GINGA SOCIAL:

SISTEMA DE INTEGRAÇÃO ENTRE

TV DIGITAL INTERATIVA E FACEBOOK

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador

Prof. Dr. Frank Augusto Siqueira

Banca Examinadora

Prof. Bruno Cavaler Ghisi

Prof. Dr. Roberto Willrich

Dedicamos esse trabalho aos nossos Pais.

AGRADECIMENTOS

Fiodor Castro

Agradeço a todos aqueles que estiveram presentes e contribuiram, direta ou indiretamente, para a realização deste trabalho. Em especial agradeço aos meus pais, Terezinha e Santiago, que me ensinarem a ler e valorizar os estudos, ao professor Frank pela acolhida e sua dedicação e ao colega e amigo Gustavo que decidiu prontamente encarar este desafio ao meu lado.

Gustavo Rupp Santos

Agradeço em especial a minha mãe Glacy e meu pai Arion por todo o amor, carinho e apoio incondicional dados em todos os dias de minha vida.

À minha namorada Dani pelo seu amor, carinho, dedicação e incentivo durante toda a minha graduação, ao meu irmão Ramiro por ter inspirado-me a entrar na área da tecnologia e à minha tia Caren por ter oferecido a oportunidade de viver essa experiência em Florianópolis.

Agradeço também a minha avó Noeli pelo seu carinho e ajuda nos momentos precisos.

Também agradeço à minha família, amigos, colegas de graduação, professores e demais pessoas envolvidas que estiverem presentes até o fim desse projeto.

Por último, mas não menos importante, agradeço ao meu amigo e colega Fiodor pela oportunidade que me deu para trabalhar ao seu lado nesse projeto. Agradeço também ao nosso professor orientador Frank pela sua paciência e dedicação para conosco.

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

AJAX - *Asynchronous* Javascript and XML

Amazon EC2 - Amazon *Elastic Cloud Computing*

API - *Application Programming Interface*

ATSC - *Advanced Television System Committee*

BD - Banco de Dados

CSS - *Cascading Style Sheets*

DASE - *DTV Application Software Environment*

DTO - *Data transfer object*

DVB - *Digital Video Broadcasting*

EPG - *Electronic Programming Guide*

GS - Ginga Social

GSTV - Ginga Social TV

GSW - Ginga Social *Web*

HDTV - *High-definition television*

HTML - *HyperText Markup Language*

HTTP - *Hypertext Transfer Protocol*

IDE - *Integrated Development Environment*

ISDB-T - *Integrated Services Digital Broadcasting Terrestrial*

JDBC - Java *Database Connectivity*

JDK - Java *Development Kit*

JEE - Java *Enterprise Edition*

JRE - Java *Runtime Environment*

JS - JavaScript

JSE - Java *Standard Edition*

JSON - JavaScript *Object Notation*

JSP - Java *Server Pages*

LDTV - *Low-definition television*

MVC - *Model-view-controller*

NCL - *Nested Context Language*

ORM - *Object-relational mapping*

POJO - *Plain Old* Java *Object*

REST - *Representational State Transfer*

REST - *Representational State Transfer*

SBTVD - Sistema Brasileiro de Televisão Digital Terrestre

SDTV - *Standard-definition television*

SGBD - Sistema de Gerenciamento de Banco de Dados

SOA - *Service-Oriented Architecture*

SQL - *Structured Query Language*

STB - *Set-top box*

TV - Televisão

TVD - Televisão Digital

TVDI - Televisão Digital Interativa

URI - *Uniform Resource Identifier*

URL - *Uniform Resource Locator*

XML - *eXtensible Markup Language*

RESUMO

O padrão brasileiro de TV Digital trouxe uma nova realidade. Podemos citar a possibilidade de transmissão de até quatro programações diferentes, melhor qualidade de sinal e a interatividade. No Brasil a interatividade será possível graças ao *middleware* Ginga e a um canal de retorno. Paralelamente a este processo de surgimento da TV digital no Brasil presenciamos, também, a ascensão da chamada Web 2.0 e com ela as Redes Sociais. A relação entre redes sociais e TV Digital tem-se intensificado abrindo novas oportunidades de estudos. Após uma abordagem sobre a TV Digital e seus diferentes padrões, bem como as redes sociais, em especial o Facebook, o presente trabalho aborda o desenvolvimento prático de um sistema que proporciona a integração entre a rede social Facebook e a TV Digital.

Palavras chave: TV Digital Interativa, Rede Social, Ginga-J, Facebook

1 INTRODUÇÃO

1.1 Contextualização

O processo de definição do padrão para teledifusão digital no Brasil durou cerca de 15 anos, compreendendo os anos de 1991 a 2006. Somente neste último ano, após inúmeros debates sobre qual dos modelos já existentes adotar, o governo brasileiro, através do decreto presidencial nº 5.820 (BRASIL, 2006), definiu oficialmente sua posição. A decisão tomada foi a de que o Sistema Brasileiro de Televisão Digital (SBTVD) teria como base o padrão de sinais do ISDB-T - *Integrated Services Digital Broadcasting Terrestrial* (Serviço Integrado de Transmissão Digital Terrestre) - com a possibilidade de incorporações tecnológicas, ou seja, um novo padrão era criado. Definido os rumos da TV digital no Brasil, deu-se início por parte das emissoras, ao processo de implementação do novo padrão. Neste processo, ainda em andamento, muito destaque está sendo dado ao sinal de alta definição em detrimento das demais características da televisão digital, praticamente ainda desconhecidas, como por exemplo, multiprogramação e interatividade.

O padrão brasileiro prevê a possibilidade de transmissão de até quatro programações diferentes (com a resolução de imagem padrão) no mesmo espectro (canal), entretanto o governo federal editou a norma 001/2009[1] que deixa dúvidas quanto à possibilidade das emissoras privadas aderirem à multiprogramação. No Brasil a interatividade será possível graças ao Ginga e a um canal de retorno. O Ginga[2] é uma camada de *software* intermediário (*middleware*) que permite o desenvolvimento de aplicações interativas para a TV Digital de forma independente da plataforma de

[1] Norma 001/2009 publicada através da Portaria do Ministério das Comunicações nº 24 de 11/02/2009.

[2] Ginga: http://www.ginga.org.br

hardware dos fabricantes de terminais de acesso (*set-top boxes*). Aplicações interativas podem integrar serviços *Web*, complementar o conteúdo de programas sendo veiculados, jogos, aplicações para o governo (*T-Gov*), bancárias (*T-Banking*), comércio eletrônico (*T-Commerce*) e aplicações que permitam a integração com outros dispositivos que não o controle remoto. O canal de retorno é um meio que permitirá os televisores enviarem dados às emissoras de TV e, assim, de fato propiciar a interatividade.

Paralelamente a este processo de surgimento da TV digital no Brasil presenciamos, também, a ascensão da chamada *Web* 2.0. A *Web* 2.0 é um termo que passou a ser usado após o estouro da bolha da Internet em 2001. Depois de muitas discussões sobre sua definição Tim O'Reilly a definiu como algo de fronteiras não bem definidas, num contexto onde a Internet é uma plataforma de serviços dos quais os usuário geram e controlam o conteúdo (O'REILLY, 2005). Dentre estes serviços se destacam as redes sociais, como o Youtube[3], Facebook[4] e Twitter[5]. Estas redes sociais são serviços que conectam indivíduos e seu conteúdo consiste somente em informações disponibilizados por estes indivíduos. O grande número de pessoas adeptas às redes sociais, bem como o alto nível de interação entre seus usuários, fez com que estas se tornassem uma importante fonte de informação, combinando qualidade de informação com celeridade para disseminá-las.

As redes sociais passaram a influenciar fortemente as emissoras de televisão e os seus programas e, por outro lado, os conteúdos televisivos são os principais e mais frequentes temas repercutidos nas redes. Recentemente a respeitada Nielsen, empresa de

[3] Youtube: https://www.youtube.com
[4] Facebook: http://www.facebook.com
[5] Twitter: https://twitter.com

medição de estatísticas, publicou um relatório afirmando que 60% dos estadunidenses veem TV enquanto acessam a Internet (NIELSENWIRE, 2010). Um caso interessante foi o uso do Twitter durante a transmissão do evento esportivo Super Bowl nos Estados Unidos, onde 50% da atividade nesta rede social foi relacionada ao evento (GHISI; LOPES; SIQUEIRA, 2010). No Brasil aconteceu algo parecido durante a exibição do programa BBB10. A audiência não foi recorde, pelo contrário foi a mais baixa de todas as edições, mas de longe foi a edição com o maior número de votos e rentabilidade. Durante o programa era impossível acessar o Twitter ou Orkut[6] sem deparar-se com dezenas de mensagens sobre o BBB (SILVA, 2011).

Esta interação entre redes sociais e TV Digital tende a se intensificar no Brasil a medida que o padrão SBTVD vai se concretizando já que, podemos observar, ambos possuem grande aceitação por parte dos brasileiros. Fala-se em uma convergência entre estas duas plataformas e esta nova realidade torna-se um interessante tema a ser estudado.

1.2 Objetivo Geral

O objetivo geral do trabalho consiste no desenvolvimento de uma aplicação, dentro do ambiente de TV Digital contendo o *middleware* brasileiro Ginga, que proporcione o acesso a funcionalidades da rede social Facebook, promovendo a convergência entre as duas plataformas.

1.3 Objetivos Específicos

O presente trabalho possui os seguintes objetivos específicos:

[6]Orkut: http://www.orkut.com.br

1. Estudo sobre a TV Digital, seus padrões e *middlewares*;
2. Estudo detalhado sobre o *middleware* Ginga com ênfase na máquina de execução;
3. Estudo referente às redes sociais e suas influências;
4. Estudo das API's do Facebook abrangendo as características pertinentes ao tema proposto;
5. Implementação de uma aplicação integrando Facebook e o Ginga.

1.4 Justificativa

Os temas escolhidos pelos autores divergem quanto à sua abrangência. Por um lado tem-se a TV Digital ainda incipiente e no outro as redes sociais em alta, com milhões de usuários ao redor do mundo. Através de leituras dos mais variados temas e das próprias experiências pessoais, os autores estão do lado daqueles que entendem que caminha-se para uma convergência entre mídias. Trazer funcionalidades de uma rede social, o Facebook, que praticamente é uma unanimidade no Brasil (SERASA EXPERIAN, 2013), para um ambiente de TV Digital, torna explicíto as capacidades desta nova realidade que toca o mundo televisivo.

1.5 Metodologia

O presente trabalho foi desenvolvido da seguinte forma:

- Pesquisa bibliográfica: seleção de normas brasileiras (ABNT), artigos, trabalhos acadêmicos e correlatos visando a compreensão do estado da arte dos temas propostos e, assim, definir o objetivo bem como confirmar sua viabilidade.

- Tecnologias a serem utilizadas: definidos os objetivos, procedeu-se a uma análise afim de identificar quais alternativas de ferramentas estavam a disposição para a realização do projeto. As alternativas adotadas são descritas no decorrer do texto.
- Definição das funcionalidades da rede social Facebook a serem providas no ambiente de TV bem como a infraestrutura necessária para tanto.
- Implementação das funcionalidades.

1.6 Organização do Texto

O capítulo 2 trata da fundamentação teórica, abordando temas relacionados à TV Digital, como por exemplo, TV Digital Interativa, padrões de TV Digital e TV Digital no Brasil. É tratado também sobre as redes sociais, mais específicamente sobre o Facebook e algumas de suas características importantes no escopo do trabalho. O capítulo 3 fornece detalhes da implementação da solução idealizada pelos autores. A seguir, no capítulo 4, conclusões e trabalhos futuros são descritos pelos autores.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Breve Histórico da Televisão

Pode-se dizer que a maioria das invenções humanas evoluem ou se tornam obsoletas e caem em desuso. O aparelho televisor, pelo menos por enquanto, tem evoluído ao longo da história passando por diversas transformações até atingir a forma como o conhecemos hoje. Juntamente com sua evolução tecnológica ocorre, também, o crescimento da sua importância na sociedade, passando de um aparelho acessível apenas

para uma minoria até se tornar um imprescindível meio de comunicação presente em quase todos os lares.

Segundo afirma (MATTOS, 2002), a cronologia do aparelho televisor começou no ano de 1873, quando o norte-americano Willoughby Smith descobriu que o elemento químico selênio possuía propriedades fotocondutoras, constatando que sua condutividade elétrica variava dependendo da quantidade de luz. Dois anos depois outro norte-americano, George Carey, propôs a criação de um aparelho de transmissão de imagens por meio de circuitos elétricos. Estas descobertas foram as precursoras de todas as demais que fizeram com que a televisão chegasse aos dias atuais.

Um grande marco no que se refere à TV foi o surgimento da TV em cores. Para a transmissão em cores na televisão foram criados três padrões principais, entre as décadas de 50 e 60, que deram origem a outros. (Montez; Becker, 2005) os descrevem juntamente com o padrão brasileiro:

> **NTSC [*National Television Systems Committee*]**
>
> O primeiro padrão de difusão de TV em cores adotado nos EUA durante os anos 1953-54, e posteriormente no Canadá, Japão e em muitos outros países com sistemas elétricos de 60Hz. Possuía alguns problemas na apresentação das cores e, por isso, começou a ser designado pejorativamente de NTSC – *Never Twice the Same Color* (Donnelly, 1995 apud MONTEZ; BECKER, 2005). Ou seja, nunca conseguia a mesma cor duas vezes. Esse padrão emprega uma taxa de trinta quadros por segundo (na realidade o valor exato é de 29,97) e 525 linhas (MONTEZ; BECKER, 2005).

PAL [*Phase Alternating Line*]

Esse padrão, desenvolvido pela Telefunken da Alemanha nos anos 1960, corrigia o problema de distorção de cores do NTSC. Foi adotado em muitos países da Europa, Ásia e sul da África. O padrão PAL possui uma taxa de 25 quadros por segundo, e 625 linhas. A taxa de 25 quadros por segundo é uma pequena desvantagem desse padrão, pois pequenos tremores na tela (*flickers*) podem se tornar perceptíveis (MONTEZ; BECKER, 2005).

SECAM [*Sequential Couleur Avec Memoir*]

Padrão francês, adotado no início dos anos 1960, que, apesar de usar a mesma resolução do PAL – 625 linhas e taxa de 25 quadros por segundo –, não mantém compatibilidade com nenhum outro padrão. Foi também adotado nos países do Leste Europeu, predominantemente por uma decisão política, pelo fato de os televisores não poderem receber transmissões originadas pela maioria dos países ocidentais (MONTEZ; BECKER, 2005).

PAL-M [*Phase Alternating Line* - Modificado]

Variação do padrão PAL, desenvolvido e adotado apenas no Brasil. Apesar de usar codificação de cores do PAL, apresenta 30

> quadros por segundo (60 Hz) com 525 linhas (MONTEZ; BECKER, 2005).

No ano de 1985, o sistema digital de transmissão de TV começou a substituir a TV analógica tanto na América do Norte quanto na Europa. No período de 1998 a 2000, foram iniciados os processos de integração entre a televisão e a Internet, abrindo-se novas perspectivas para maior interação entre as mídias que compõem as novas tecnologias de comunicação (MATTOS, 2002).

2.2 TV Digital

A TV digital nada mais é do que a transmissão digital dos sinais audiovisuais, conceito bem diferente de TV interativa (MONTEZ; BECKER, 2005). A grande diferença, tão aclamada pelos meios de comunicação, é a conservação na qualidade do sinal, acarretando em ausência de interferências. Além disso com a transmissão digital há um melhor aproveitamento do espectro de transmissão fazendo com que seja possível transmissões em alta definição, multiprogramação e o envio de aplicações juntamente com o programa televisivo.

Atualmente as transmissões predominantes são aquelas que propagam o sinal analógico. (MONTEZ; BECKER,2005) definem um sinal como uma medida de grandeza física - seja acústica, ótica ou elétrica - que veicula algum tipo de informação. Um sinal de TV, por exemplo, corresponde a uma onda eletromagnética que veicula informações sobre áudio, vídeo e dados de sincronização, usadas pelo aparelho receptor. (MONTEZ; BECKER, 2005) definem ainda que um sinal analógico é qualquer tipo de sinal em que a amplitude varia continuamente no tempo (Figura 1-a) e que a

digitalização do sinal analógico é obtida através de três etapas: amostragem, quantização e codificação. (FOROUZAN, 2006) por sua vez comenta que um sinal analógico possui infinitos níveis de tensão num certo período de tempo. Quando uma onda evolui do valor A para o valor B, ela passa através de um número infinito de valores ao longo do caminho. Contrariamente, um sinal digital possui apenas um número limitado e definido de valores, simplificados frequentemente como 1 e 0 (Figura 1-b).

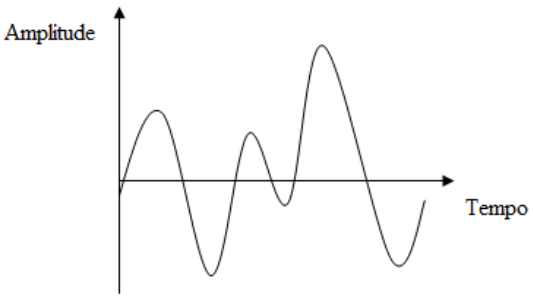


Figura 1-a - Sinal analógico

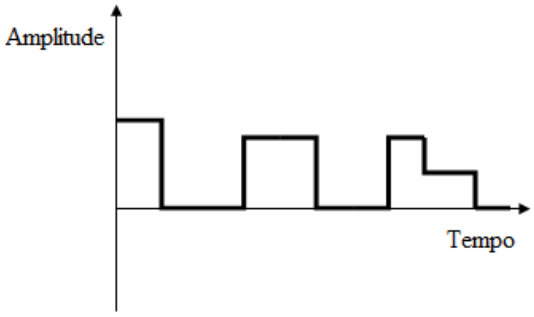


Figura 1-b - Sinal Digital

Em um sistema de TV digital, o sinal transmitido pode conter, além de áudio e vídeo, um fluxo de dados. A transmissão de dados adicionais junto com fluxos de áudio e vídeo é chamada de *datacasting*. Um *datacasting* fortemente acoplado é aquele em que o fluxo de dados é temporalmente relacionado ao áudio e ao vídeo. O *datacasting* fracamente acoplado é aquele em que os dados se relacionam com o áudio e vídeo mas não de maneira sincronizada. Já no *datacasting* descacoplado os dados são totalmente independentes podendo fluir entre os telespectadores e o provedor de forma equivalente ao que ocorre na internet, quando um internauta navega na web.

A forma padronizada em TV digital para *datacasting* é a do carrossel de dados e carrossel de objetos. A idéia básica é permitir a transmissão periódica de dados sobre um fluxo de transporte (*Transport Stream* - TS) como mostra a Figura 2. Com o envio periódico de dados, o receptor/telespectador apenas aguarda o próximo envio quando precisar de uma determinada informação adicional (MONTEZ;BECKER, 2005).

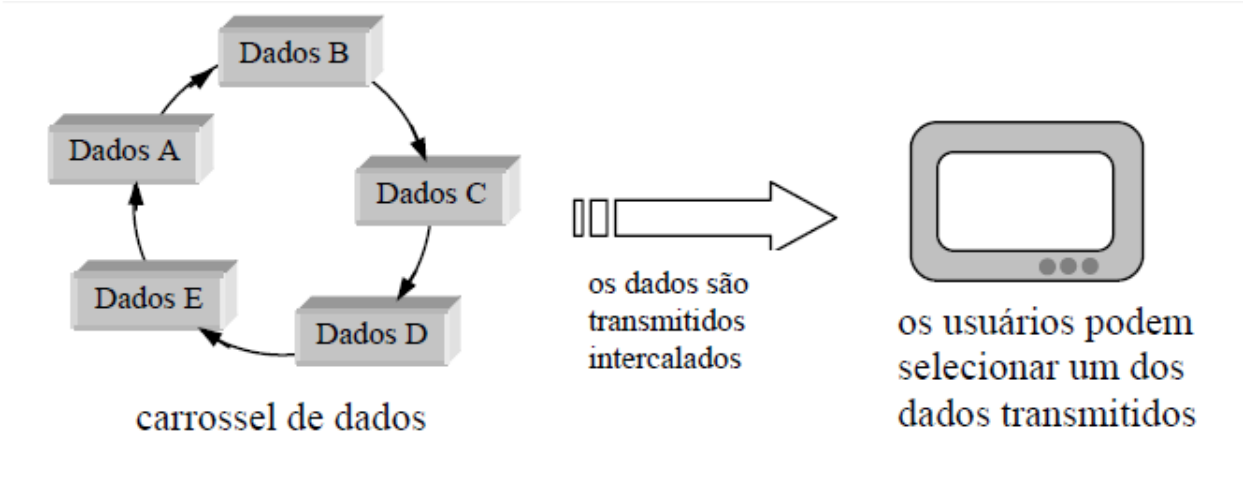


Figura 2 - Carrossel de dados. Fonte: (GAWLINSKI, 2003).

Todos os tipos de arquivos, tais como páginas *web*, imagens JPEG, músicas em MP3, programas de computadores e bases de dados, podem ser transmitidos dessa forma, utilizando o fluxo de dados. Guias de programação eletrônica (EPG – *Electronic Program Guide*), aplicativos em Java (denominados Xlets) ou NCL e *softwares* novos para o *set-top box* são os exemplos mais citados de uso para essa tecnologia. A implementação do carrossel de dados em fluxos de transporte MPEG-2 é baseado no protocolo DSM-CC (*Digital Storage Media Command and Control Protocol)*. O DSM-CC foi criado originalmente visando uma forma de suportar a entrega de vídeos sob demanda usando um fluxo de transporte MPEG-2 (MONTEZ; BECKER, 2005).

Um sistema de TV digital pode ser caracterizado como um sistema cliente/servidor, no qual o lado do servidor é representado pela emissora (radiodifusora)

e o cliente é representado pelo telespectador. A Figura 3 da uma visão geral deste sistema quanto à difusão e recepção do sinal:

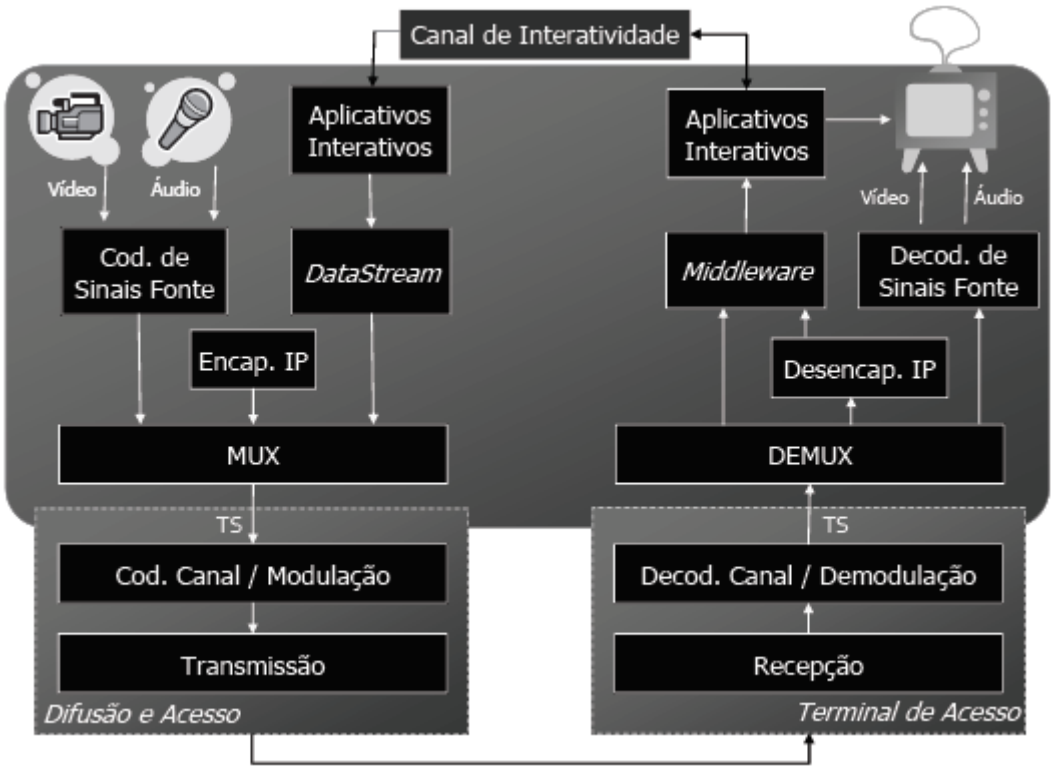


Figura 3 - Sistema de TV Digital. Fonte: (BARBOSA; SOARES, 2008).

A geração do conteúdo televisivo (servidor) está representada no lado esquerdo da figura. O áudio e vídeo dos programas passam por um codificador digital para gerar os fluxos de áudio e vídeo principais comprimidos. Em seguida, juntamente com os dados adicionais, que podem ser encapsulados no formato IP ou outro, são multiplexados em um único sinal, denominado fluxo de transporte. Este sinal é modulado para um canal frequência e transmitido.

No lado do telespectador (cliente) o sinal recebido passa pelo demodulador, em seguida, o demultiplexador (demux) separa os fluxos de áudio, vídeo e de dados. Os fluxos de áudio e vídeo seguem para os decodificadores apropriados e os dados passam

por um processamento. O processamento destes dados pode implicar na necessidade de novos dados que podem ser adquiridos através do canal de interatividade (canal de retorno). Este canal pode servir também como um canal de comunicação entre a emissora e o telespectador.

Este modelo de TV Digital é desenvolvido em cinco camadas e são descritas abaixo com suas funções:

- **camada de aplicação** - executa e controla o ciclo de vida das aplicações;
- **camada de *middleware*** - esta camada é responsável por abstrair os detalhes de hardware, ou seja, as aplicações devem estar de acordo com o *middleware*;
- **camada de compressão** - responsável por realizar a compressão e descompressão dos fluxos sendo transmitidos (áudio, vídeo e dados);
- **camada de transporte** – as operações de multiplexacão e demultiplexação são realizadas nesta camada, multiplexar significa transmitir simultaneamente dois ou mais fluxos em um mesmo canal; e
- **camada de transmissão** - Composto pela codificação de canal, modulação e transmissão, no lado da difusão e acesso e pela demodulação e decodificação no lado do terminal de acesso (CPqD, 2006 *apud* BITTENCOURT; BENNERT, 2007).

A ideia central da arquitetura em camadas é cada uma oferecer serviços para a camada superior e usar os serviços oferecidos pela inferior. Dessa forma, as aplicações executadas na TV digital interativa usam uma camada de *middleware*, que intermedia toda a comunicação entre a aplicação e os demais serviços são oferecidos pelas camadas inferiores (MONTEZ; BECKER, 2005).

2.2.1 TV Digital Interativa

O grande potencial da TV digital, pelo menos no Brasil, é, sem dúvida, a possibilidade de interatividade. Segundo (HOUAISS; VILLAR, 2001) interatividade é o "ato ou faculdade de diálogo intercambiável entre o usuário de um sistema e a máquina, mediante um terminal equipado de tela de visualização".

Para se ter um melhor entendimento do conceito de interatividade, podemos classificá-lo em três níveis, em ordem crescente de abrangência (REISMAN, 2002 *apud* MONTEZ; BECKER,2005):

- **Reativo -** nesse nível, as opções e realimentações (*feedbacks*) são dirigidas pelo programa, havendo pouco controle do usuário sobre a estrutura do conteúdo (nível de padronização elevado);

- **Coativo -** apresenta-se aqui possibilidades do usuário controlar a sequência, o ritmo e o estilo;
- **Pró-ativo -** o usuário pode controlar tanto a estrutura quanto o conteúdo.

Também é possível classificar a interatividade das mídias em termos de mídia quente ou fría (MCLUHAN,1964 *apud* MONTEZ; BECKER,2005):

- **Mídias quentes:** São aquelas altamente ou totalmente padronizadas, não havendo espaço para a interação. Distribuem mensagens prontas, sem possibilidade de intervenção (rádio, cinema, fotografia, dentre outros);

- **Mídias frias:** São as que permitem a interatividade, que deixam lugar livre onde os usuários poderão preencher e interagir (telefone, computadores e a rede mundial de informação).

Podemos dizer que a televisão ao longo da historia é classificada nestes dois tipos de mídias, se considerarmos o seu estado mais rudimentar até aquele em que é possível chegar com a TV Digital.

Como mencionado anteriormente o sinal em um sistema de TV digital pode conter dados adicionais além do áudio e vídeo. O fluxo de dados (*datacasting*) pode ser utilizado para transportar músicas, imagens, páginas web, aplicações etc. Somente com o fluxo de dados já é possível prover interatividade ao telespectador ainda que de forma limitada. (LEMOS, 1997) classifica a interatividade na televisão em cinco níveis de interação baseados na evolução tecnológica dessa mídia:

- **Nível 0:** estágio em que a televisão expõe imagens em preto e branco e dispõe de um ou dois canais. A ação do espectador resume-se a ligar e desligar o aparelho, regular o volume, brilho ou contraste e trocar de um canal para outro;
- **Nível 1:** televisão em cores, maior número de emissoras e controle remoto. Ele facilita o controle que o telespectador tem sobre o aparelho, mas, ao mesmo tempo o prende ainda mais à televisão;
- **Nível 2:** acoplamento de alguns equipamentos periféricos à televisão, como o videocassete, as câmeras portáteis e os jogos eletrônicos. Novas tecnologias possibilitam novos usos para o objeto televisão, podendo agora também ver vídeos e jogar, e das emissões, podendo gravar programa e vê-los quando e quantas vezes quiser;

- **Nível 3:** já aparecem sinais de interatividade de características digitais. O telespectador pode então interferir no conteúdo a partir de telefones (“Você Decide” da Rede Globo de Televisão) por fax ou correio eletrônico;
- **Nível 4:** é o estágio da chamada televisão interativa em que se pode participar do conteúdo a partir da rede telemática em tempo real, escolhendo ângulos de câmera, diferentes encaminhamentos das informações etc.

(Montez; Becker, 2005) argumentam que ainda no nível 4 o telespectador não tem o controle total sobre a programação. Ele apenas reage a impulsos e caminhos pré-definidos pelo transmissor o que vai contra uma característica da interatividade que é a não padronização. Sendo assim estes autores propõem mais três níveis:

- **Nível 5:** o telespectador pode ter uma presença mais efetiva no conteúdo, saindo da restrição de apenas escolher as opções definidas pelo transmissor. Passa a existir a opção de participar da programação enviando vídeo de baixa qualidade, que pode ser originado por intermédio de uma webcam ou filmadora analógica. Para isso, torna-se necessário um canal de retorno ligando o telespectador à emissora, chamado de canal de interação;
- **Nível 6:** a largura de banda desse canal aumenta, oferecendo a possibilidade de envio de vídeo de alta qualidade, semelhante ao transmitido pela emissora. Dessa forma, a interatividade chega a um nível muito superior à simples reatividade, como caracterizado no nível quatro;
- **Nível 7:** neste nível, a interatividade plena é atingida. O telespectador passa a se confundir com o transmissor, podendo gerar conteúdo. Esse nível é semelhante

ao que acontece na internet hoje, onde qualquer pessoa pode publicar um site, bastando ter as ferramentas adequadas. O telespectador pode produzir programas e enviá-los à emissora, rompendo o monopólio da produção e veiculação das tradicionais redes de televisão que conhecemos hoje.

Cabe ressaltar que o conceito de TV interativa ainda é muito discutido, entretanto, há um consenso no sentido de que a TV deixa de ser unidirecional e alguns estudiosos tentam definir ao menos o seu alcance. Apesar de uma ampla abrangência, é possível classificar toda variedade de informações incorporadas pelo termo TV interativa em sete grandes grupos segundo o Emarketer (MACLIN, 2001 *apud* MONTEZ; BECKER, 2005):

- **TV Avançada (*Enhanced TV*) :** tipo de conteúdo televisivo que engloba texto, vídeo e elementos gráficos, como fotos e animações. Na sua forma mais simples, é a apresentação integrada desses elementos, organizada por uma grande de programação. A principal diferença para a TV analógica consiste justamente na integração desses elementos e no aumento da qualidade do vídeo e do som. A resolução do monitor deixa de ser na proporção 4:3 para ser 16:9, igual à resolução da tela de cinema.
- **Internet na TV:** também conhecida como TV conectada, é capaz de acessar a internet e todas as suas funções.
- **TV Individualizada:** o telespectador é capaz de adaptar a TV às suas preferências. Escolher ângulos de câmeras, personalizar a interface com escolha

de cores e fontes e repetição de cenas perdidas são exemplos de uma funcionalidades de uma TV Individualizada.

- **Vídeo Sob Demanda:** dada a grade de programação de uma emissora o telespectador é capaz de escolher qual programa assistir em um determinado horário.
- ***Personal Video Recorder*** **(PVR):** também conhecido como *Personal TV* ou *Digital Video Recorder*, especificando apenas alguns dados dos programas, como por exemplo, título, horário ou ator é possível gravar programas e armazená-los em um disco rígido. Esta função permite, inclusive, parar a transmissão de programa ao vivo e retomá-la do ponto de parada fazendo com que o telespectador assista ao programa na íntegra.
- ***Walled Garden*****:** é um guia das aplicações interativas que informa ao usuário o que se pode fazer, o que tem disponível e serve como um canal de entrada para essas aplicações.
- **Console de Jogos:** a televisão assume o papel de um console de jogos, seja localmente, contra a própria TV/computador ou em rede.

Aos sete grupos propostos podem ser acrescentados mais dois grupos levando em consideração a TV Digital Interativa (GAWLINSKI, 2003):

- **Guia de programação eletrônica:** um portal contendo a programação, análogo à revista contendo a programação da TV a cabo, entretanto, o usuário navega pelo guia com o uso do próprio controle remoto. Semelhante ao *Walled Garden* sendo que um trata da programação e o outro das aplicações;

- **Serviços de teletextos:** refere-se a informações textuais adicionadas à programação. Seu conteúdo pode ser variado, complementando a própria transmissão ou trazendo informações meteorológicas, econômicas, etc.

2.2.2 Padrões de TV Digital

As pesquisas em TV digital iniciaram no final da década de 1980 e vieram a se consolidar na década seguinte com o lançamento de dois padrões de TV digital: ATSC (*Advanced Television System Committee*) e DVB (*Digital Video Broadcasting*). O Japão, pioneiro nas pesquisas em TV de alta definição digital, somente lançou comercialmente seu padrão em 2003.

Atualmente há três principais padrões de TV digital vigentes, excetuando-se o padrão brasileiro, são eles: DVB, o europeu, ATSC, o estadunidense e ISDB (*Integrated Services Digital Broadcasting*), o japonês. A adoção destes padrões nos países ao redor do globo pode ser vista na Figura 4. A região da america do sul aparece como ISBD, pois, o padrão brasileiro teve como base o padrão japonês e foi expandido para os demais países vizinhos.

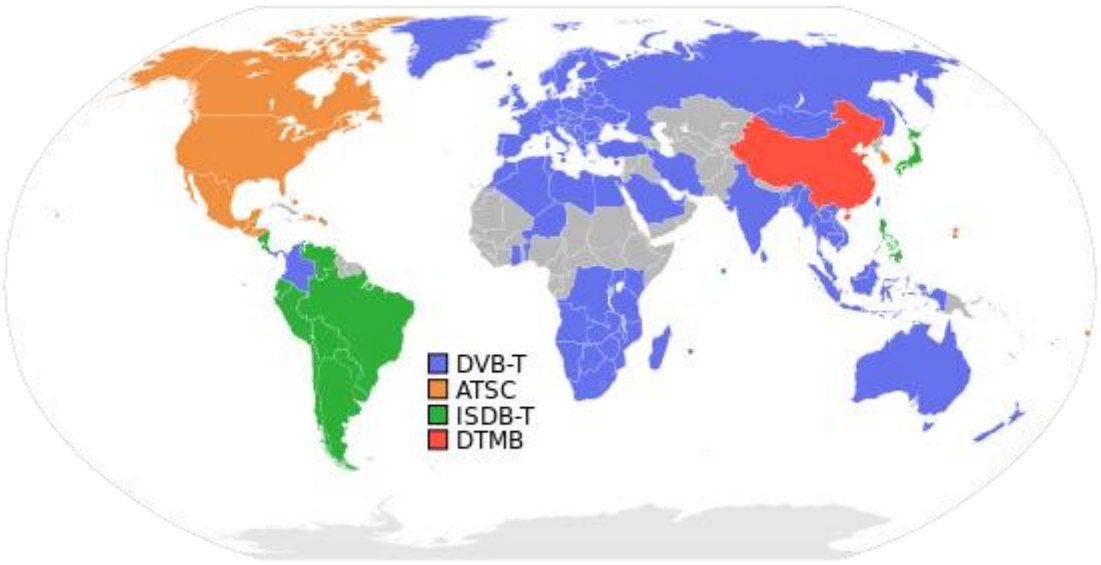


Figura 4 - Sistemas de TV Digital. Fonte: WIKIPEDIA[7].

Como mencionado na seção anterior um sistema de TV digital é composto por diferentes camadas e cada um dos sistemas possui a sua implementação para elas. A Figura 5 mostra as tecnologias utilizadas nas diferentes camadas dos três padrões:

[7] Disponível em: http://en.wikipedia.org/wiki/Broadcast_television_systems#cite_note-5. Acesso em: Agosto de 2012

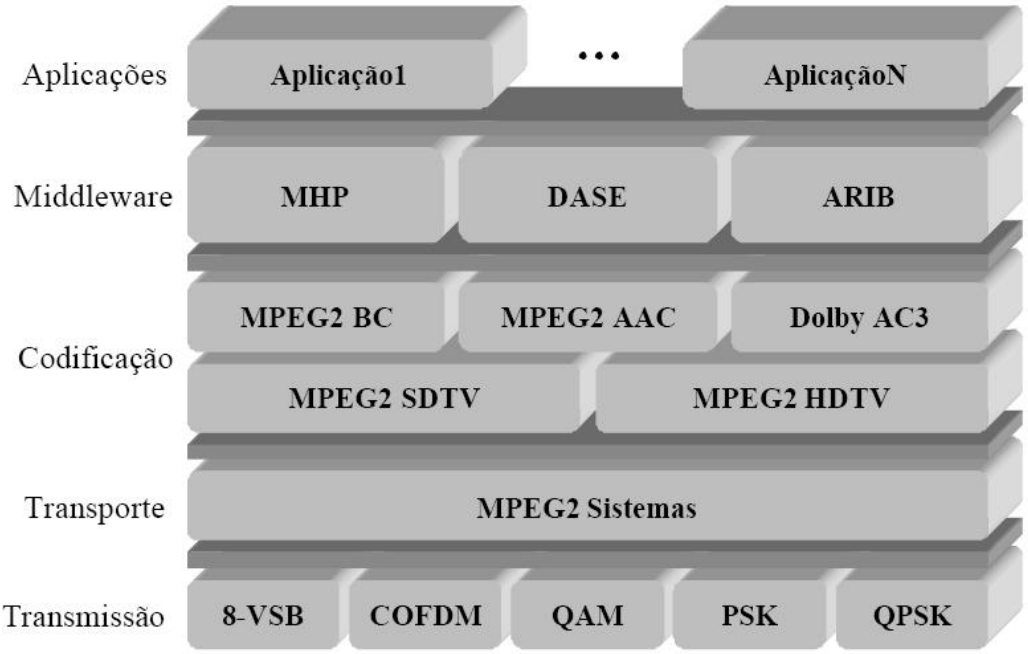


Figura 5 - Opções de padrões para um sistema de televisão digital interativa. Fonte: (FERNANDES; LEMOS; SILVEIRA, 2004).

Na sequência desse trabalho, os principais padrões são detalhados.

2.2.2.1 ATSC - *Advanced Television Systems Committee*

Foi o primeiro dos três a ser desenvolvido, começa em 1990 e termina em 1995, iniciando seu funcionamento a partir de 1998. A Figura 6 mostra a arquitetura do ATSC (ATSC,2011) bem como os componentes presentes em cada camada.

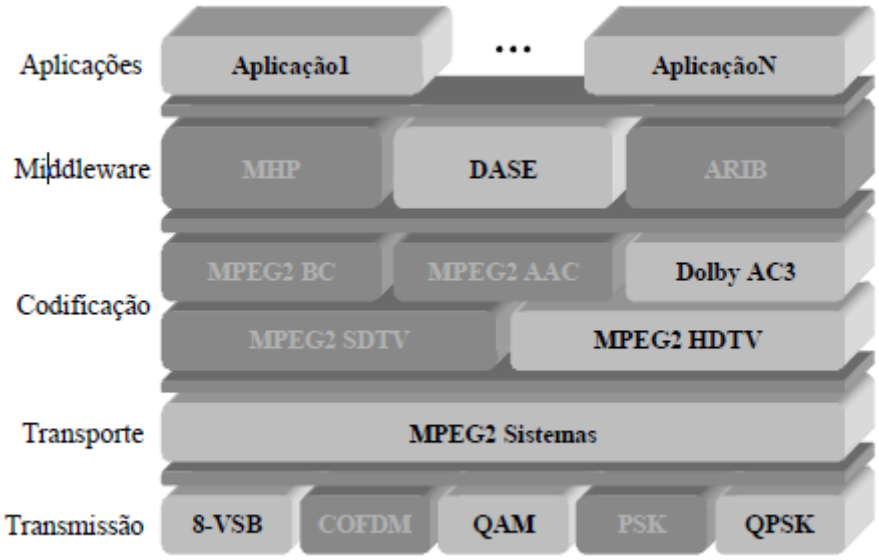


Figura 6 - Arquitetura ATSC. Fonte: (FERNANDES; LEMOS; SILVEIRA, 2004).

O padrão ATSC permite diversas configurações para a camada de transmissão, definindo diferentes esquemas de modulação para transmissão terrestre, via cabo e via satélite. Na radiodifusão terrestre, o padrão ATSC pode operar com canais de 6, 7 ou 8MHz e utiliza a modulação 8-VSB, cuja taxa de transmissão é de 19,8 Mbps. Em função deste esquema de modulação, um sistema ATSC apresenta problemas na recepção por antenas internas e não permite a recepção móvel. Para as redes de televisão a cabo e as transmissões via satélite, da mesma forma que o padrão DVB, o padrão ATSC adota as modulações 64-QAM e QPSK, respectivamente (FERNANDES; LEMOS; SILVEIRA, 2004).

O padrão ATSC prevê diversos modos de transmissão com diferentes níveis de resolução da imagem e formatos de tela. No entanto, o modelo de negócios americano foi direcionado para a televisão de alta definição (HDTV) (FERNANDES; LEMOS; SILVEIRA, 2004).

Na camada de codificação, o sinal de áudio é codificado usando o padrão Dolby AC-3 (ATSC, 2001) e o sinal de vídeo é codificado usando a recomendação MPEG-2 Vídeo (ISO, 1996b) com qualidade HDTV. Na camada de transporte, da mesma forma que o padrão DVB, o padrão ATSC multiplexa e demultiplexa os fluxos elementares de áudio, vídeo e dados (aplicações) usando a recomendação MPEG-2 Sistemas (ISO, 1996a).

2.2.2.2 DVB - *Digital Video Broadcasting*

O padrão europeu teve origem a partir da criação de um consórcio de mesmo nome DVB (DVB, 2011). Este padrão é formado por um conjunto de normas que definem padrões de transmissões, sendo os mais conhecidos: DVB-T: (transmissão terrestre por radiodifusão); DVB-C (transmissão via cabo); DVB-S (transmissão via satélite); DVB-MC (transmissão via microondas operando em frequências de até 10GHz); e DVB-MS (transmissão via microondas operando em frequências acima de 10GHz). Estes padrões adotam diferentes métodos de modulação. Os componentes do padrão DVB são mostrados na Figura 7.

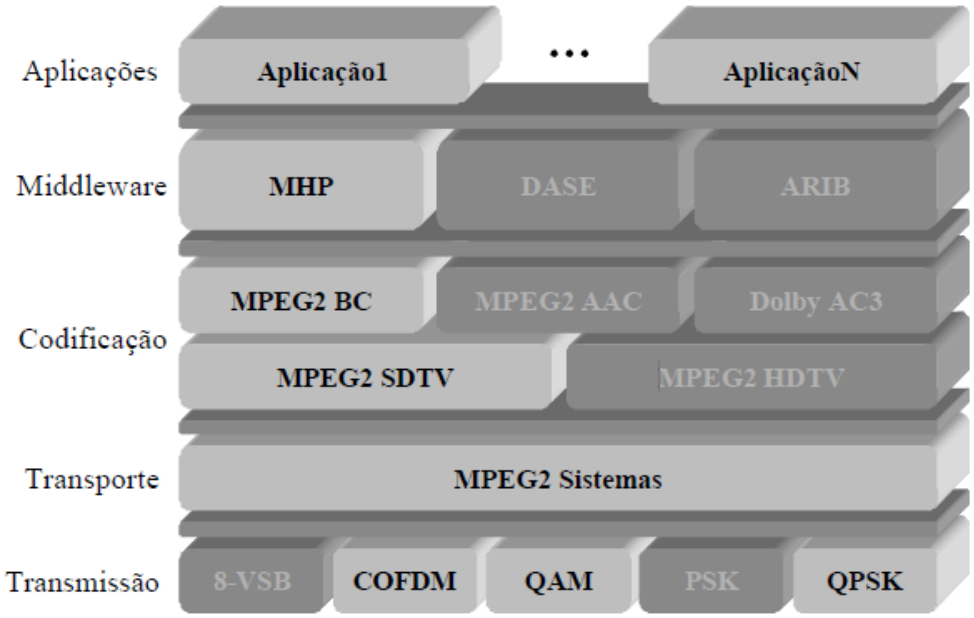


Figura 7 - Arquitetura DVB. Fonte: (FERNANDES; LEMOS; SILVEIRA, 2004)

O DVB-T (ETSI, 2001) pode operar em canais de 6, 7 ou 8 MHz e adota a modulação COFDM (*Coded Orthogonal Frequency Division Multiplexing*), cuja taxa de transmissão pode variar entre 5 e 31,7 Mbps, dependendo dos parâmetros utilizados na codificação e modulação do sinal. Este padrão utiliza seis modos de transmissão que abrangem desde sistemas de alta definição (HDTV – *High Definition Television*) até sistemas de baixa definição (LDTV – *Low Definition Television*). No entanto, alguns estudos apontam que o funcionamento não é satisfatório quando ocorrem transmissões simultâneas para sistemas de alta definição e sistemas móveis (FERNANDES; LEMOS; SILVEIRA, 2004).

Para as redes de televisão a cabo, o DVB-C (ETSI, 1998) adota a modulação 64-QAM ou QAM. Para a difusão via satélite, o DVB-S (ETSI, 1997a) recomenda a modulação QPSK. Para a radiodifusão terrestre utilizando microondas, são previstos dois tipos de modulação. Para frequências abaixo de 10 GHz (MMDS), o DVB-MC

(ETSI, 1997c) recomenda a utilização da modulação 16, 32 ou 64-QAM. Para frequências acima de 10 GHz (LMDS), o DVB-MS (ETSI, 1997b) recomenda o mesmo mecanismo de modulação que o DVB-S, ou seja, QPSK.

Nas camadas de transporte e codificação, o DVB é fundamentalmente baseado no MPEG-2 e suas recomendações. Na camada de codificação, o sinal de áudio é codificado usando a recomendação MPEG2-BC e o sinal de vídeo é codificado usando a recomendação MPEG-2 Vídeo (ISO, 1996b) com qualidade SDTV. Na camada de transporte, os fluxos elementares de áudio, vídeo e dados (aplicações) são multiplexados no produtor de conteúdo e demultiplexados nos *set-top boxes* dos usuários usando a recomendação MPEG-2 Sistemas (ISO, 1996a).

2.2.2.3 ISDB - *Integrated Services Digital Broadcasting*

O padrão ISDB teve sua especificação lançada no ano de 1999 no Japão pelo grupo DiBEG - *Digital Broadcasting Experts Group* (DiBEG, 2011). Assim como os demais padrões o ISDB é formado por diversos documentos que definem os diversos padrões adotados. A Figura 8 apresenta as tecnologias encontradas nas diferentes camadas.

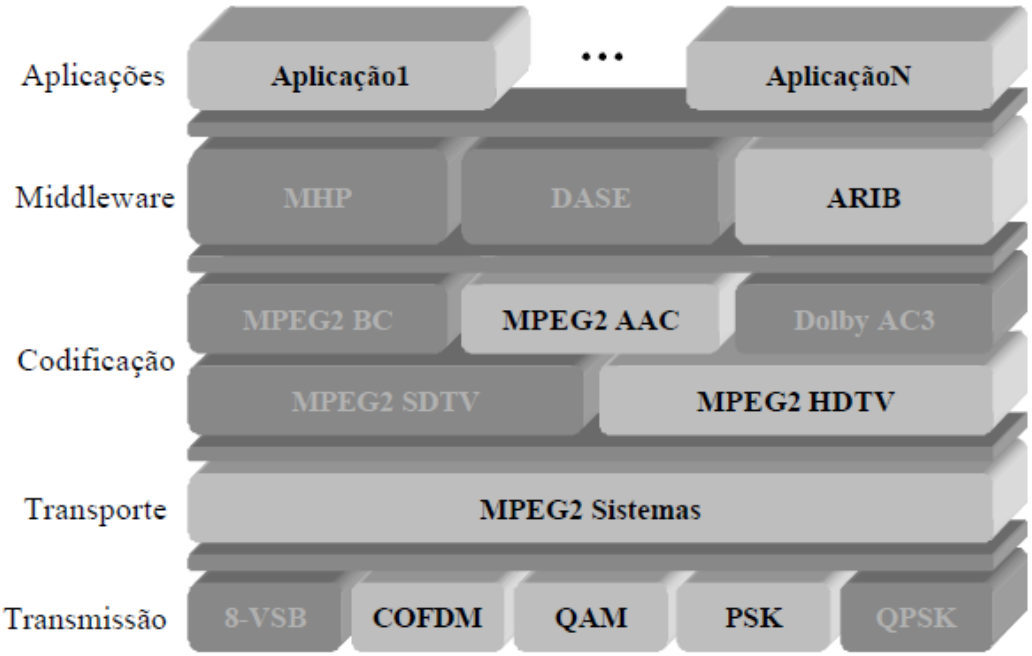


Figura 8 - Arquitetura ISDB. Fonte: (FERNANDES; LEMOS; SILVEIRA, 2004).

O ISDB utiliza a modulação COFDM, com algumas variações; possui uma taxa de transferência que varia entre 3,65 a 23,23 Mbps, e uma largura de banda de 6, 7 ou 8 MHz. As suas maiores vantagens são a grande flexibilidade de operação e o potencial para transmissões móveis e portáteis. A multiplexação e a codificação de vídeo, como nos dois padrões anteriores, também são realizadas em MPEG-2. A codificação de áudio utiliza o MPEG-2 ACC (MONTEZ; BECKER, 2005).

O ISDB é projetado para suportar sistemas hierárquicos com múltiplos níveis, podendo ser usado, por exemplo, para prover simultaneamente recepção de baixa taxa de dados sob condições móveis excepcionalmente difíceis, taxa de dados intermediária (SDTV) para recepção estática e alta taxa de dados (HDTV) para boas condições de recepção (FERNANDES; LEMOS; SILVEIRA, 2004).

Embora seja baseado no sistema de transmissão europeu, o ISDB-T é superior ao DVB-T quanto à imunidade a interferências, permitindo a convivência da televisão de alta definição com a recepção móvel (FERNANDES; LEMOS; SILVEIRA, 2004).

2.2.3 *Middlewares*

Neste capítulo seão apresentados os *middlewares* dos padrões de TV Digital citados anteriormente. Segundo (RNP, 2011) *middleware* é o neologismo criado para designar camadas de software que não constituem diretamente aplicações, mas que facilitam o uso de ambientes ricos em tecnologia da informação. A camada de middleware concentra serviços como identificação, autenticação, autorização, diretórios, certificados digitais e outras ferramentas para segurança. A importancia *middleware* para o contexto da TV Digital torna-se clara se levarmos em conta a heterogeneidade dos aparelhos existentes.

2.2.3.1 DASE

DASE (*DTV Application Software Environment*) (DASE, 2003) trata-se do *middleware* especificado para o padrão ATSC. A Figura 9 apresenta uma arquitetura de referência do *middleware*, que basicamente é divido em *DASE Content Model* (*DASE applications*) e o *DASE Environment Model* (*DASE System*).

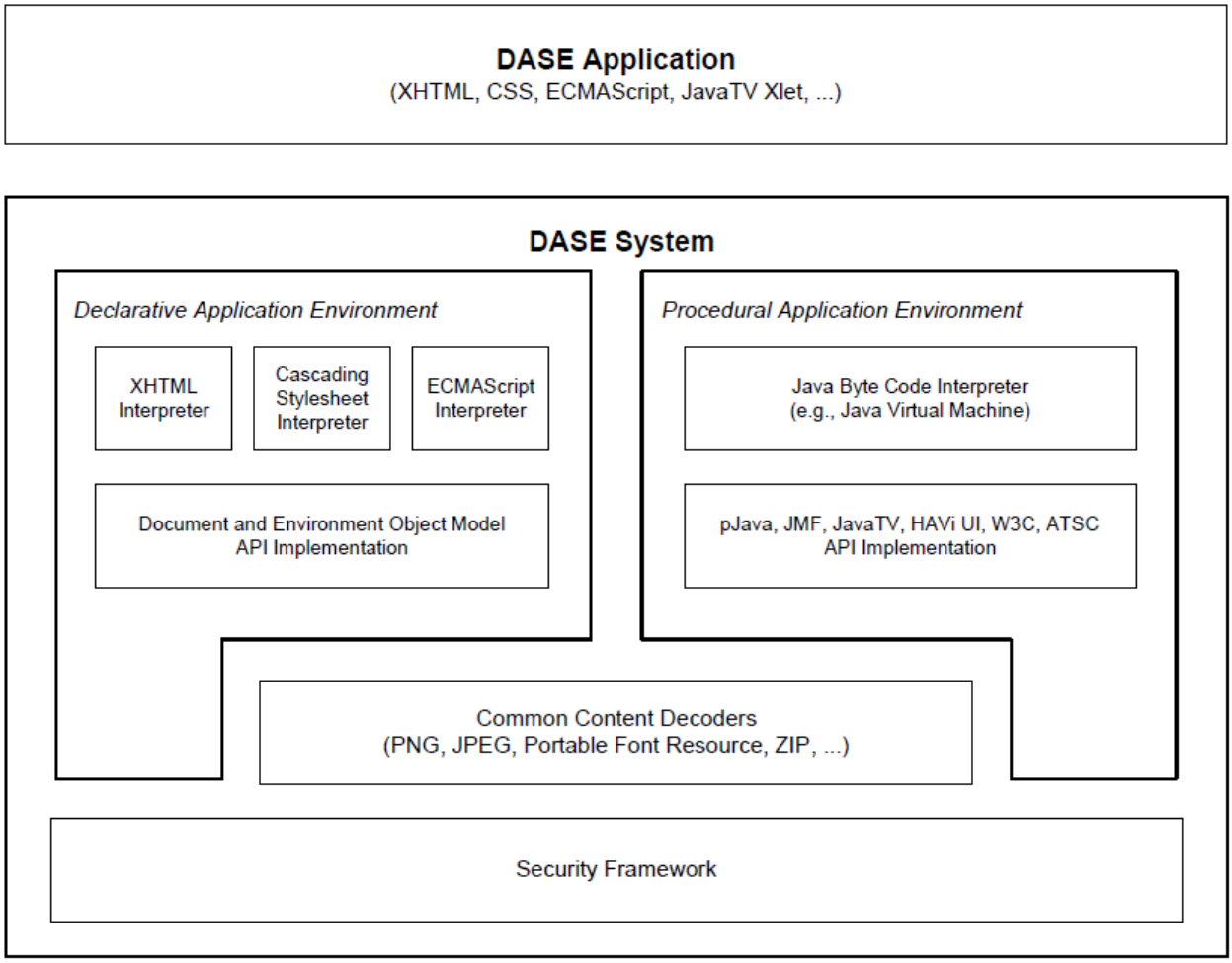


Figura 9 - Arquitetura DASE. Fonte: (DASE, 2003)

De acordo com (PAES; ANTONIAZZI, 2005), alguns componentes são definidos abaixo:

- **Aplicação DASE** (***DASE Application***) – é a coleção de informações que expressa um conjunto específico de comportamentos observáveis.

- **Ambiente de Aplicações Declarativas (*Declarative Application Environment*)**– é basicamente o *browser* de documentos multimídia (*User Agent*).

  Aplicações Declarativas – é o documento multimídia composto por regras de estilo, *scripts, markups*, gráficos, vídeo e áudio.

- **Ambiente de Aplicações Procedurais (*Procedural Application Environment*)** – é a *Java Virtual Machine* e a implementação de suas APIs.

  Aplicações Procedurais – é o aplicativo JavaTV Xlet, composto por código binário compilado em Java em conjunto com outros conteúdos como gráficos, vídeos e áudios.

O módulo Decodificação de Conteúdo Comum, nas fases de decodificação e apresentação da aplicação tem a responsabilidade de prover os serviços comuns aos dois ambientes. Já o *Framework* de Segurança tem a função de criptografar os dados trafegados entre o receptor do usuário e a transmissora (SASAKE, 2007).

Aplicações DASE podem ser híbridas, ou seja, uma mescla entre aplicações procedurais e declarativas. As aplicações declarativas podem utilizar *scripts* com o intuito de agregar caraterísticas de aplicações procedurais.

As aplicações procedurais são implementadas através da linguagem Java (*Personal Java*, *Java Media Framework*, e Java TV), realizando tarefas complexas dinamicamente e um conjunto de APIs específicas do DASE. Um exemplo de aplicação procedural é o Xlet Java em conjunto com outros conteúdos multimídia como gráfico, vídeo e áudio (DASE, 2003 *apud* SASAKE, 2007). As aplicações declarativas são desenvolvidas através de coleções de páginas escritas em linguagens de marcação

(HTML, XML, SGML, etc), folhas de estilos ou linguagens baseadas em scripts (JavaScript ou VBScript) (SASAKE, 2007).

O *middleware* DASE adota um ciclo de vida idêntico para ambos modelos de aplicação, diferentemente do *middleware* MHP, tratado mais adiante, que possui um ciclo de vida para cada um dos paradigmas.

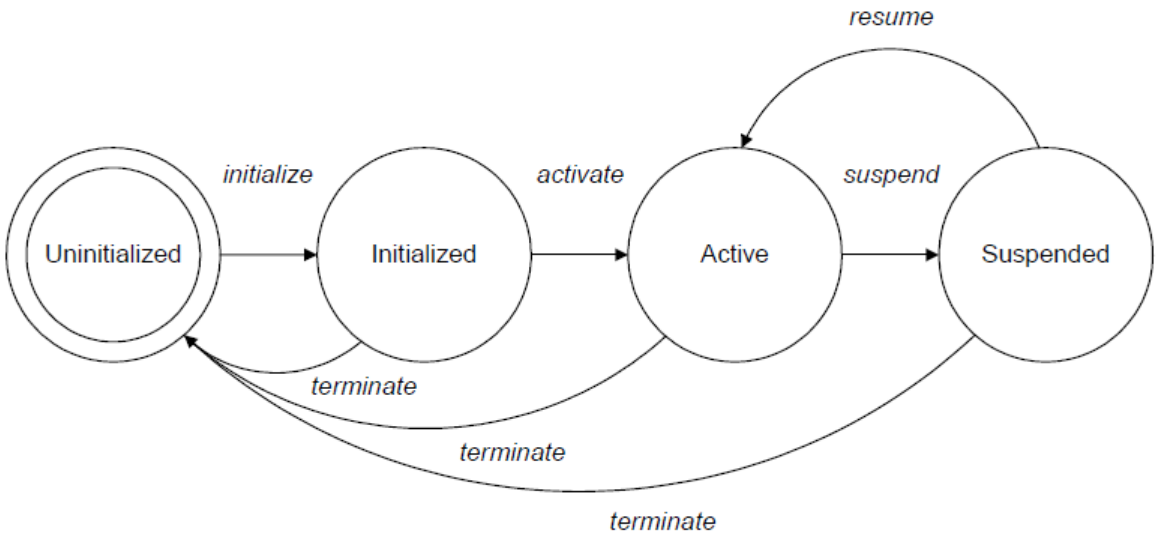


Figura 10 - Ciclo de vida de aplicações DASE. Fonte: (DASE, 2003)

Como podemos observar na máquina de estados ilustrada da Figura 10, o ciclo de vida de uma aplicação inicia e termina no estado *Uninitialized*. Enquanto no estado não inicializado, um aplicativo não deve consumir qualquer recurso do ambiente. Entretanto, o DASE *System* pode consumir recursos de *cache* para reter recursos de um aplicativo ao mesmo tempo que está em um estado não inicializado. Múltiplas aplicações podem estar no estado não inicializado em um determinado momento dentro de um DASE *System*. Uma vez que a aplicação deixa este estado ela só retorna através de um evento *terminate*, que pode ser gerado pela própria aplicação ou pelo *DASE System*.

Ao passar para o estado *Initialized* uma aplicação já pode consumir recursos do ambiente, porém, ainda não está sendo executada. Ao passar para o estado *Active* uma aplicação pode consumir qualquer recurso do ambiente, sendo que apenas uma aplicação pode estar neste estado. Ao passar para o estado *Suspended,* uma aplicação libera todos os recursos dos quais estava fazendo uso e bloqueia todas as suas *threads*.

2.2.3.2 MHP - *Multimedia Home Platform*

O DVB-MHP (*Digital Video Broadcasting – Multimedia Home Platform*) (ETSI, 2006) é o *middleware* do padrão europeu. O objetivo do MHP é prover um *middleware* que suporte um grande número de serviços, inclusive *Web Browsing*. Interoperabilidade e segurança de informação são observadas com um maior grau de atenção por este padrão. Além dos objetivos citados, o MHP é um padrão aberto, podendo assim ser usado por empresas, países e fundações sem o pagamento de *royalties* (PAES; ANTONIAZZI, 2005).

O MHP possui um ambiente de execução que utiliza-se de uma máquina virtual Java (JVM) e um conjunto de interfaces de programação de aplicações. Uma aplicação DVB usando API Java é denominada aplicação DVB-J. No ambiente declarativo, o DVB utiliza uma linguagem semelhante ao HTML, denominada DVB-HTML.

Dentro da especificação do MHP encontra-se a arquitetura básica do mesmo mostrada na Figura 11.

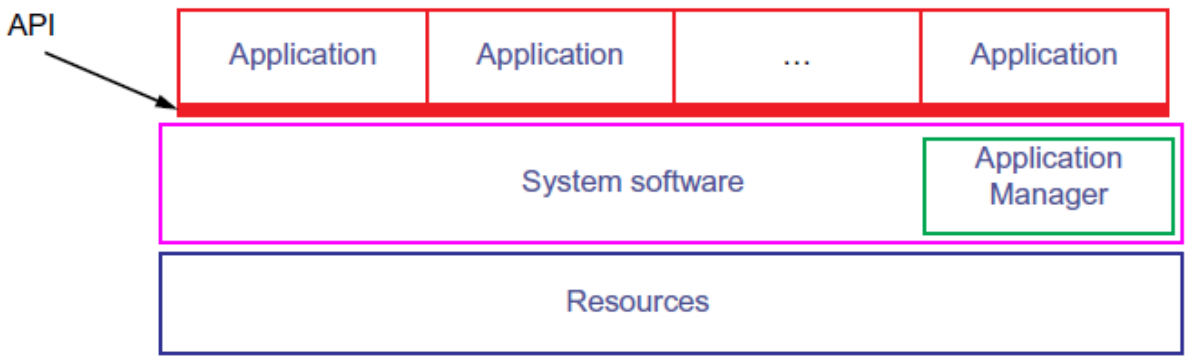


Figura 11 - Arquitetura básica MHP. Fonte: (ETSI, 2006).

A camada *Resources*, de mais baixo nível, abrange os recursos de *hardware* e *software* para fornecimento de serviços do *middleware*. Estes recursos devem ser acessados de maneira transparente pelas diferentes aplicações suportadas. A camada *System software* visa o isolamento do *hardware* das aplicações interoperáveis residentes na camada superior. Dentro desta camada existe, também, o *Application Manager* responsável por gerenciar o ciclo de vida de todas as aplicações inclusive as interoperáveis. A figura 12 detalha a camada *System sofware*:

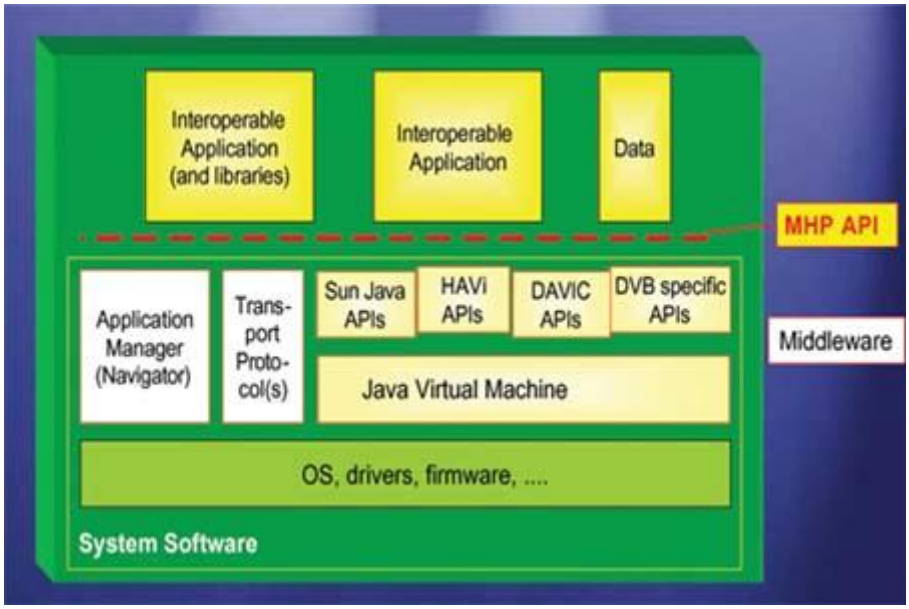


Figura 12 - Camada *System Software* do MHP. Fonte: (PAES; ANTONIAZZI, 2005)

O modelo procedural é composto por 4 blocos fundamentais (Java API, DAVIC, HAVi e DVB API) sobre uma camada de interface baseada em máquina virtual Java e denominada como DVB-J API. No contexto dos padrões DVB, uma aplicação interpretada pelo *user agent* é chamada de aplicação DVB-HTML. Um *user agent* pode ser implementado como um *plug-in* do *middleware* procedural, a ser enviado pelo provedor de conteúdo, como um Xlet, quando for necessária a apresentação de uma aplicação DVB-HTML.

É importante notar que as especificações MHP não possibilitam a implementação de um ambiente que possua apenas o *middleware* declarativo DVB-HTML. Ele é considerado, pelo padrão DVB, dependente do *middleware* procedural MHP. O *user agent* utiliza a API genérica definida pelo *middleware* procedural, ou seja, um *user agent* tem acesso aos recursos do terminal de acesso ou mesmo da máquina virtual Java, através da API procedural MHP.

As aplicações MHP correspondem em sua maioria à categoria DVB-J desenvolvidas na linguagem JAVA (Xlets). As aplicações DVB-HTML não são muito difundidas, principalmente, devido ao nível de complexidade de implementação encontrado pelos fabricantes de receptores, emissoras e desenvolvedores de conteúdo (SASAKE, 2007).

O ciclo de vida das aplicações DVB-J corresponde ao próprio ciclo de vida dos Xlets, mostrado mais adiante seção 2.3.1.4.1. As aplicações DVB-HTML possuem o ciclo de vida de acordo com a Figura 13.

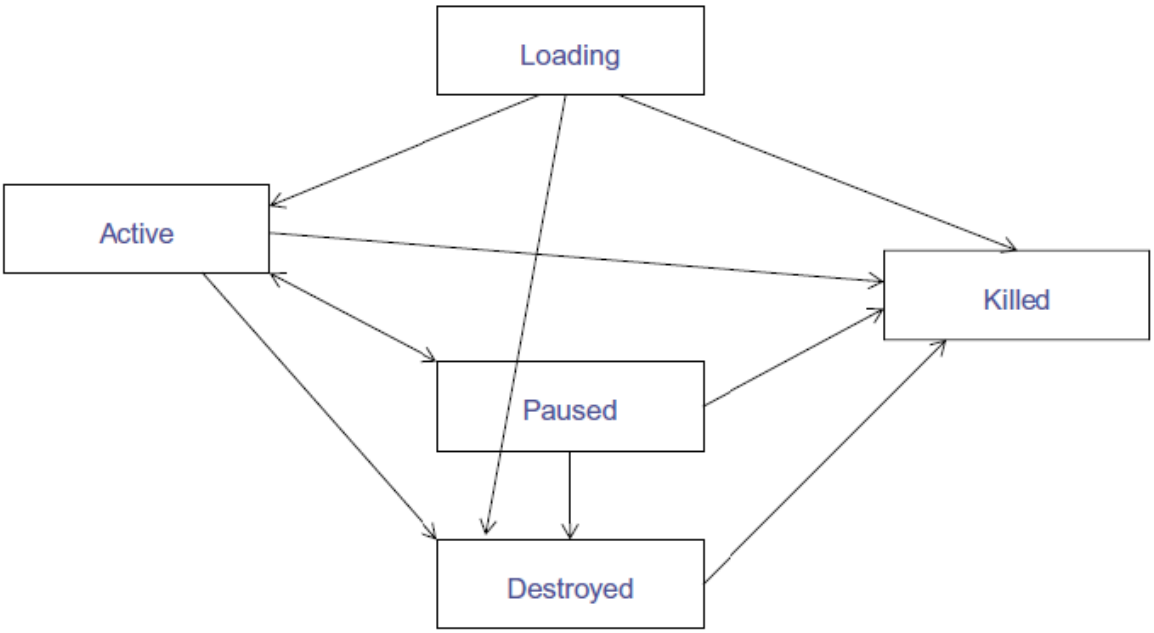


Figura 13 - Ciclo de vida de aplicações DVB-HTML. Fonte: (ETSI, 2006)

O estado inicial é o *Loading* e possui a responsabilidade de acessar aos recursos do sistema inclusive aqueles que dizem respeito à aplicação. Neste estado o espectador ainda não visualiza a aplicação. Já no estado *Active* assume-se que se tem disponível todo o conteúdo do documento sendo processado bem como os recursos do MHP. Quando há perda de todos os recursos a aplicação passa para o estado *Destroyed* mas, mesmo assim, existe a possibilidade de a aplicação continuar executando. Quando a aplicação de fato é finalizada passa-se ao estado *Killed*.

2.2.3.3 ARIB - *Association of Radio Industries and Businesses*

Na camada de *middleware*, o padrão ISDB adota a plataforma padronizada pelo ARIB (*Association of Radio Industries and Businesses*) (ARIB, 2002), definindo uma

camada de *software* que permite a programação de conteúdo e aplicações. O ARIB adota um modelo de aplicação baseado na linguagem declarativa denominada BML (*Broadcast Markup Language*), que é baseada na linguagem XML (*eXtensible Markup Language*) (FERNANDES; LEMOS; SILVEIRA, 2004).

Neste sistema, áudio, vídeo e todos os serviços de dados são multiplexados e transmitidos via broadcasting de rádio, em um fluxo de transporte, especificado pelo MPEG-2. Canais para a interatividade das comunicações são disponibilizados através dos canais interativos da rede, tanto fixas quanto móveis. Este *middleware* suporta três tipos de transmissões de dados:

- Sistema de transmissão de dados que utiliza o armazenamento dos pacotes como um fluxo de pacotes no PES (*Packetized Elementary Stream*). Este sistema é muito usado para aplicações em tempo real e, basicamente, para serviços de dados que necessitem de controle de tempo na decodificação e reprodução – vídeo, áudio e legendas – ou ainda naqueles que precisam ser sincronizados com outros fluxos. Este sistema é especificado como '*data stream*' (PAES; ANTONIAZZI, 2005).
- Sistema de transmissão de dados que utiliza as seções. Sistema utilizado para serviços de armazenagem de informação (*data storage services*). Dados que serão transmitidos repetidas vezes podem ser armazenados no receptor após o primeiro *download* e reutilizados sempre que necessário. Este serviço é especificado como carrossel de dados (PAES; ANTONIAZZI).
- No terceiro sistema suportado pelo ARIB os dados são armazenados diretamente no *payload* do pacote *do* fluxo de transporte.

A Figura 14 ilustra as principais características do *middleware* ARIB.

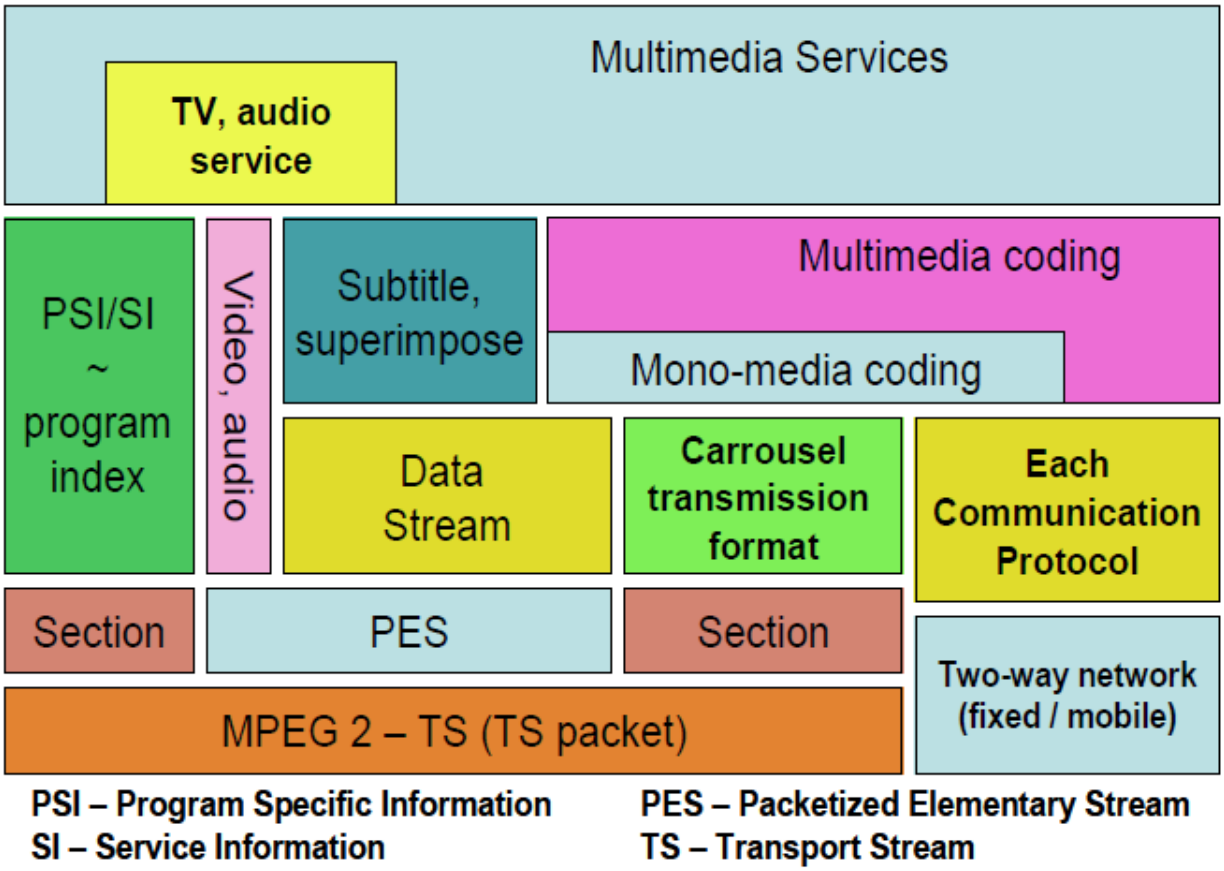


Figura 14 - Principais características do ARIB (PAES; ANTONIAZZI, 2005)

Como mencionado a *Broadcast Markup Language* (BML) é a linguagem declarativa do ARIB, baseia-se em XML e consiste em um conjunto de *tags* para o formato do conteúdo multimídia. As aplicações são apresentadas através de um navegador BML do padrão japonês e seguem um ciclo de vida simples já que somente uma instância do *browser* é ativada. Seus estados são: Ativo, Inicializado, Suspenso e Destruído.

Dos três tipos de *middlewares* apresentados nenhum pode ser considerado compatível com os demais, ou seja, aplicações somente serão executadas no padrão para

o qual foram desenvolvidos devido às suas diferentes APIs. Apesar de incompatíveis os *middlewares* possuem funcionalidades equivalentes.

2.2.3.4 GEM - *Globally Executable* MHP

Geralmente em ambientes computacionais em que há muita diversidade de soluções/implementações, busca-se uma padronização para que haja portabilidade e interoperabilidade entre os componentes dos diferentes sistemas. A padronização contribui também para a redução de custos na implantação de um novo componente no mercado. No ambiente da TV Digital ocorreu o mesmo, (MONTEZ; BECKER; FILHO, 2005) mencionam a padronização do MPEG-2 TS na camada de transporte de sistemas de TV Digital que não só possibilitou o emprego de componentes de *hardware* e *software* já existentes, como também facilitou o desenvolvimento, testes e depuração de novos componentes.

Com o surgimento dos primeiros sistemas de TV Digital constatou-se que os *middlewares* existentes possuíam diversas características em comum, o que induziu à uma concentração dos esforços no sentido de tornar uniformes os padrões.

Por ser o padrão de *middleware* mais maduro, o MHP foi escolhido como referência para essa tentativa de estabelecimento de conformidade. Contudo, o MHP possui diversas especificidades do DVB e, por conseguinte, sua utilização pura e simples não é adequada em sistemas de outros países que adotam, por exemplo, ATSC ou ARIB. Por conseguinte, o GEM (*Globally Executable* MHP) (ETSI, 2005) foi criado para facilitar a criação de especificações de *middlewares* baseados em MHP (MONTEZ; BECKER; FILHO, 2005).

Segundo (LEITE *et al.* 2005) formalmente, o GEM por si só não pode ser considerado uma especificação completa para terminais de acesso. O correto é dizer que GEM é um *framework* a partir do qual uma implementação de um terminal de acesso pode ser instanciada; ou ainda, que GEM é um padrão ao qual implementações existentes devem se adaptar para obter uma conformidade que garante a execução global de aplicações. O padrão define, portanto, um conjunto de APIs, garantias semânticas, protocolos e formatos de conteúdo com os quais as aplicações (agora globalmente interoperáveis) podem contar para a constituição de serviços interativos, executáveis em qualquer plataforma definida pelos padrões internacionais compatíveis.

O GEM referencia características do padrão DVB descrevendo diversas partes do MHP e sinaliza aquelas que devem ser substituídas de acordo com a infraestrutura da plataforma a ser considerada. Uma solução será considerada compatível com o GEM se o consórcio DVB entender que as alternativas funcionais desta são equivalentes àquelas sinalizadas pelo *framework*. A Figura 15 mostra a relação do GEM com os demais *middlewares* dos outros padrões de TV Digital.

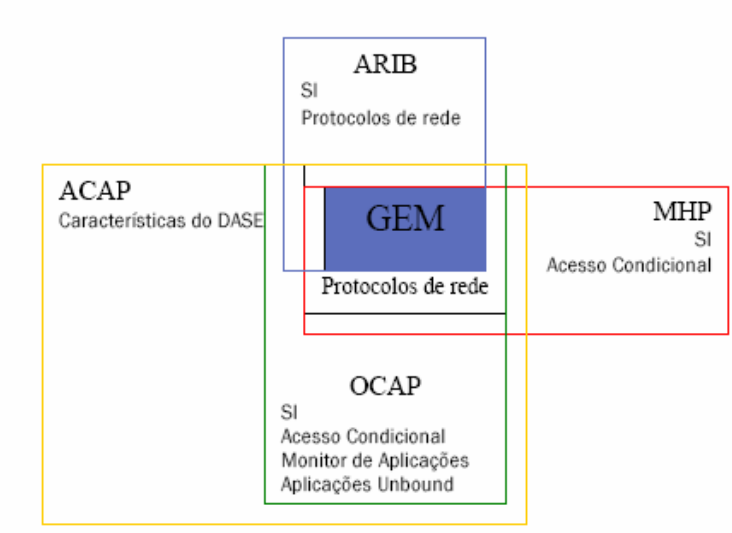


Figura 15 - Relação entre o GEM e demais *middlewares*. Fonte:(DAMASCENO, 2008)

A Figura 16 traça um comparativo entre os diferentes padrões de *middlewares*:

COMPARAÇÃO DOS PADRÕES DE MIDDLEWARE			
CARACTERÍSTICA	MHP	DASE	ARIB
Segurança	SIM	SIM	Não disponível
Decodificação de conteúdo comum (PNG, JPEG, ZIP etc)	SIM	SIM	SIM
Tipos de aplicativos	HTML e JavaTV	XHTML,CSS,ECMA Script, JavaTV	Não disponível
Distinção entre aplicações declarativas e procedurais	SIM	SIM	Não disponível
Interação com usuário	SIM	SIM (teclado, mouse)	SIM
Capacidade de Áudio	MPEG BC	Non-streaming: (audio/basic) Streaming: (Dolby AC-3)	SIM
Capacidade de Vídeo	MPEG 2	Non-streaming: (Multiple Network Graphics) Streaming: (MPEG 2)	MPEG 2
Capacidade Gráfica	LDTV: 320 X 240 SDTV: 640 X 480 EDTV: 720 X 480 HDTV: 1920 X 1080	1920 X 1080 1280 X 720 960 X 540 640 X 480	Alta definição: 1920 X 1080; 1280 X 720 e 960 X 540. Definição Normal: 620 X 480.
Display	Não disponível	Multiplano: Background, vídeo, gráfico e ponteiro/cursor (8 bit pseudo color; RGBA 4444; RGBA 5551; RGBA 6666; RGBA 8880 e RGBA 8888)	Multiplano: vídeo, figura, controle, gráfico e texto e legendas: (Y, Cr, Cb/4:2:2/8bits;Y, Cr, Cb/4:4:4/8bits/ composição do canal α em 256 valores;1920 X 1080 X 1 - 1 bit de controle; 8 bit para endereçamento de mapa de cores) Correção de erros sem perc
Metadados	SIM	SIM	SIM
Receptor (STB)	Receptores Comuns de baixo custo	Receptores Comuns	Receptores Comuns de baixo custo
Extenções/Expansões	SIM	Não disponível	SIM
Serviços	HDTV, SDTV, outros serviços de telecomunicações e de dados.	HDTV, SDTV, outros serviços de telecomunicações e de dados.	HDTV, SDTV, outros serviços de telecomunicações e de dados.
Interatividade	SIM	SIM	SIM, via digital broadcasting, SDTV (terrestre), Satélite, redes de pacotes e redes de telecomunicações.
Controlabilidade	Funções de controle do usuário; Canais de emergência.	Controle do usuário	Funções de controle do usuário; canais de emergência.
Vantagens	Baixo preço do Set Up Box; Maior aceitabilidade mundial.	Possibilidade de contrapartidas comerciais nos EUA.	Melhor para aplicações móveis; Proximidade funcional com DVB.

Figura 16 - Comparativo entre os principais *middlewares* Fonte: (PAES; ANTONIAZZI, 2005)

2.3 TV Digital no Brasil - SBTVD

Pode-se dizer que o início, por parte do governo, dos estudos sobre TV Digital no Brasil foi em 1991 com a criação da Comissão Assessora para Assuntos de Televisão

(Com-Tv), encarregada de estudar e analisar a TV de alta definição que estava sendo desenvolvida em alguns países. Após algumas outras iniciativas, em 1998, o Brasil, através da Agência Nacional de Telecomunicações (ANATEL), inicou testes com os modelos ATSC e DVB que se encontravam em fase de implantação nos Estados Unidos e Reino Unido, respectivamente. Em 1999 o padrão japonês foi concluído e passou a ser testado também.

Em 2003, com as discussões sobre TV Digital cada vez mais presentes, o governo, através do Decreto nº 4.901 (BRASIL, 2003), instituiu o Sistema Brasileiro de Televisão Digital - SBTVD. Além de inovação tecnológica o governo pretendia utilizar-se do SBTVD como uma ferramenta de combate à exclusão digital e consequentemente exclusão social, sendo que esta é uma das características que o distingue dos demais sistemas. O SBTVD tem os seguintes objetivos:

> I – Promover a inclusão social, a diversidade cultural do País e a língua
>
> pátria por meio do acesso à tecnologia digital, visando a democratização da informação;
>
> II – Propiciar a criação de rede universal de educação à distância;
>
> III – Estimular a pesquisa e o desenvolvimento propiciando a expansão de tecnologias brasileiras e da indústria nacional relacionadas à tecnologia de informação e comunicação;
>
> IV – Planejar o processo de transição da televisão analógica para a digital, de modo a garantir a gradual adesão de usuários a custos compatíveis com sua renda;

> V – Viabilizar a transição do sistema analógico para o digital, possibilitando às concessionárias do serviço de radiodifusão de sons e imagens, se necessário, o uso de faixa adicional de radiofreqüência, observada a legislação específica;
>
> VI – Estimular a evolução das atuais exploradoras de serviço de televisão analógica, bem assim o ingresso de novas empresas, propiciando a expansão do setor e possibilitando o desenvolvimento de inúmeros serviços decorrentes da tecnologia digital, conforme legislação específica;
>
> VII – Estabelecer ações e modelos de negócios para a televisão digital adequados à realidade econômica e empresarial do País;
>
> VIII – Aperfeiçoar o uso do espectro de radiofreqüências;
>
> IX – Contribuir para a convergência tecnológica e empresarial dos serviços de comunicações;
>
> X – Aprimorar a qualidade de áudio, vídeo e serviços, consideradas as atuais condições do parque instalado de receptores no Brasil; e
>
> XI – Incentivar a indústria regional e local na produção de instrumentos e serviços digitais.

Com a publicação do decreto veio à tona a questão: “adotar um padrão estrangeiro ou desenvolver um novo padrão nacional?”. Após diversas discussões, em 2006 foi instituído um novo decreto, o de número nº 5.820, onde foi definido que o

SBTVD teria como base o ISDB e seria possível incorporar inovações tecnológicas com o intuito de uma melhor adequação à realidade brasileira.

O sistema brasileiro passou por alterações e de fato não se pode dizer que SBTVD e ISDB são equivalentes. A figura 17 ilustra a arquitetura do padrão brasileiro.

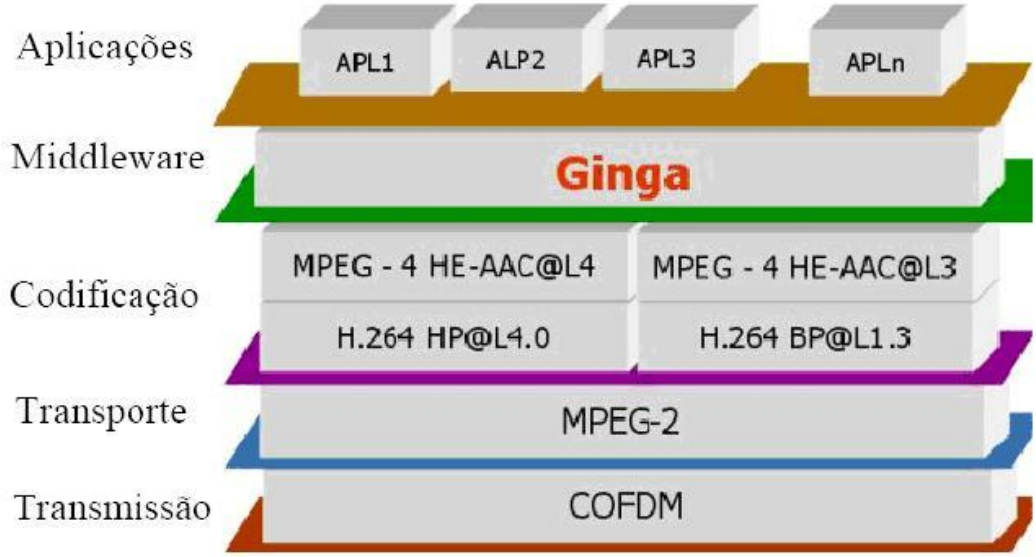


Figura 17 - Arquitetura SBTVD. Fonte: (OLIVEIRA; LACERDA, 2008)

Para compressão de vídeo, o sistema brasileiro inovou ao definir o padrão H.264, também conhecido como MPEG-4 Part 10 ou AVC (*Advanced Video Coding*), no lugar do padrão MPEG-2 utilizado nos outros sistemas de TV Digital, pois o H.264 permite obter a mesma qualidade do MPEG-2 com a metade da taxa de *bits*. Assim, o seu uso permite transmitir uma quantidade duas vezes maior de vídeos por um mesmo canal usado pelo MPEG-2 (OLIVEIRA; LACERDA, 2008).

O padrão H.264 preve transmissões com diferentes taxas de compressão, logo, podem existir diferentes resoluções de tela. Dentre elas encontram-se a HDTV/1080i e 720p, SDTV/480p e LDTV/OneSeg para dispositivos móveis.

Para compressão de áudio foi adotado o MPEG-4 AAC (*Advanced Audio Coding*), também conhecido como MPEG-2 Part 7 ou MPEG-4 Part 3. Este formato é uma evolução da Camada-3 do MPEG-1 Áudio (também denominada MP3). O AAC consegue taxa de compressão bem superior que seu antecessor.

Na camada de transporte foi adotado o padrão MPEG-2 *System*. Este padrão adiciona aos fluxos elementares de áudio principal e vídeo principal informações para suas exibições sincronizadas. A sincronização é realizada seguindo o paradigma de eixo do tempo (*timeline*) pela adição de carimbos de tempo (*timestamp*) a conjuntos de amostras codificadas de vídeo e áudio baseado em um relógio compartilhado. A geração de fluxos de dados também é determinada pelo padrão (BARBOSA; SOARES, 2008).

No receptor, essa sequência de pacotes será demultiplexada e as sequências elementares de *bits* serão reconstruídas e entregues aos seus respectivos decodificadores. Utilizando informações contidas no cabeçalho dos pacotes de transporte, é possível a realização de operações como sincronização do aparelho receptor, detecção e sinalização de erros (FERNANDES; LEMOS; SILVEIRA, 2004).

O padrão utilizado pelo sistema brasileiro na camada de transmissão é o COFDM modulando em QAM (*Quadrature Amplitude Modulation*) ou PSK (*Phase Shift Keying*). O COFDM é uma técnica de modulação baseada no OFDM (*Orthogonal Frequency Division Multiplex*) o qual utiliza sub portadoras ortogonais para modular os sinais, diferindo no acréscimo da codificação, de onde se acrescenta o “C” ao OFDM. O acréscimo da codificação de canal tem como objetivo corrigir os erros produzidos na transmissão. Embora sua complexidade seja elevada, COFDM possui melhor desempenho sob canais em condições realmente desafiadoras (RODRIGUES; GOMES, 2004 *apud* OLIVEIRA; LACERDA, 2008).

O *middleware* do padrão brasileiro é o Ginga (ABNT 15606), trata-se de uma das principais diferenças em relação aos demais padrões. Dada a importância do *middleware* brasileiro para o presente trabalho, o mesmo será descrito na próxima seção com maior riqueza de detalhes.

2.3.1 *Middleware* Ginga

As principais propostas de *middlewares* para TV Digital oferecem como suporte a execução de aplicações interativas em dois ambientes: declarativo e imperativo (MORRIS, 2005 *apud* KULESZA *et al.* 2010). O Ginga, da mesma forma, possui estes dois ambientes representados pelos componentes Ginga-NCL e Ginga-J respectivamente. A linguagem declarativa utilizada é a *Nested Context Lenguage* (NCL) e a linguagem imperativa é o Java bem como a linguagem Lua de script, esta ultima executa no ambiente delcarativo juntamente com o NCL.

O Ginga foi construído a partir da junção dos *middlewares* FlexTV (LEITE *et al.* 2005) e Maestro (SOARES, 2006), desenvolvidos por consórcios liderados pela UFPB e PUC-Rio no projeto SBTVD, respectivamente.

O FlexTV, proposta de *middleware* procedural do projeto SBTVD, incluía um conjunto de APIs já estabelecidas em outros padrões e funcionalidades inovadoras, como a possibilidade de comunicação com múltiplos dispositivos, permitindo que os mesmos enviassem e recebessem dados do *middleware*. Já o Maestro foi a proposta de *middleware* declarativo do projeto SBTVD. Com foco em oferecer facilidade do sincronismo espaço-temporal entre objetos multimídia, utiliza a linguagem declarativa NCL (*Nested Context Language*) e a linguagem de *script* Lua (PAULINELLI; KULESZA, 2012).

O *middleware* brasileiro é considerado inovador, possui a capacidade de comunicações com múltiplos dispositivos (herança do FlexTV) utilizando protocolos comuns tais como *Bluetooth*, USB, *Wi-Fi*, outros. Existe, também, a possibilidade de atualização do *middleware* em tempo de execução através do canal de retorno ou pela própria emissora, sem a necessidade do conhecimento do usuário (OLIVEIRA; LACERDA, 2008).

Outra inovação do *middleware* brasileiro é a ponte entre o ambiente declarativo e o procedural, no qual uma aplicação pode alterar e/ou executar uma aplicação de outro ambiente. É possível a gravação de aplicativos no *middleware* para serem executados posteriormente. Essa funcionalidade será bastante útil para a utilização na educação, pois o professor poderá salvar uma aplicação para ser executada com os alunos em um horário mais conveniente, ou simplesmente poder aplicar quantas vezes desejar, sem a necessidade de esperar o programa ser exibido novamente pela emissora. Essa funcionalidade foi desenvolvida para atender uma das exigências da inclusão digital (OLIVEIRA; LACERDA, 2008).

Em sua arquitetura o Ginga possui três componentes principais, Ginga-J, Ginga-NCL e Ginga-cc como mostrado na Figura 18.

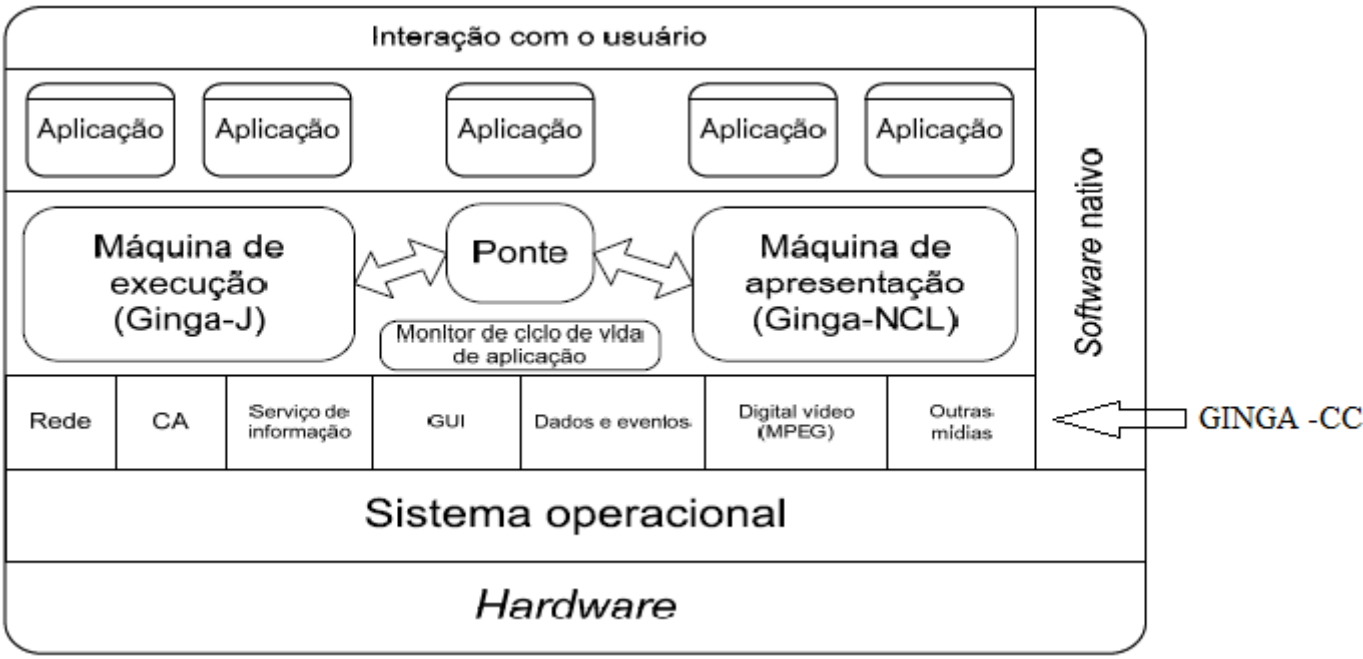


Figura 18. Arquitetura Ginga. Adaptado de (ABNT 15606-1).

Nas camadas inferiores, encontramos o *hardware* e o Sistema Operacional. A arquitetura da implementação de referência do Ginga o divide em três grandes módulos: o Núcleo Comum Ginga (Ginga *Common Core*), a Máquina de Apresentação Ginga-NCL e a Máquina de Execução Ginga-J. Podemos observar também o monitor de ciclo de vida da aplicação, uma aplicação ou recurso do sistema operacional para controle do estado do *software*. Sua função inclui a gerência de todo o ciclo de vida da aplicação, incluindo a inicialização, término e controle. O monitor do ciclo de vida de aplicações deve estar de acordo com o ambiente procedural (ABTN 15606-1). Em seguida encontramos as aplicações e a interação com o usuário. É prevista também a existência de *softwares* nativos, como EPG, aplicações nativas, entre outros.

2.3.1.1 Planos Gráficos

As aplicações Ginga podem ser apresentadas de diferentes formas (ABNT 15606-1), é definido que a função de apresentação deve ser designada baseando-se na representação lógica da tela da televisão, sendo esta composta por cinco camadas: camada de vídeo, camada de imagem estática, camada de seleção vídeo/imagem, camada de texto e gráficos e camada de legendas. A Figura 19 ilustra a disposição dos planos:

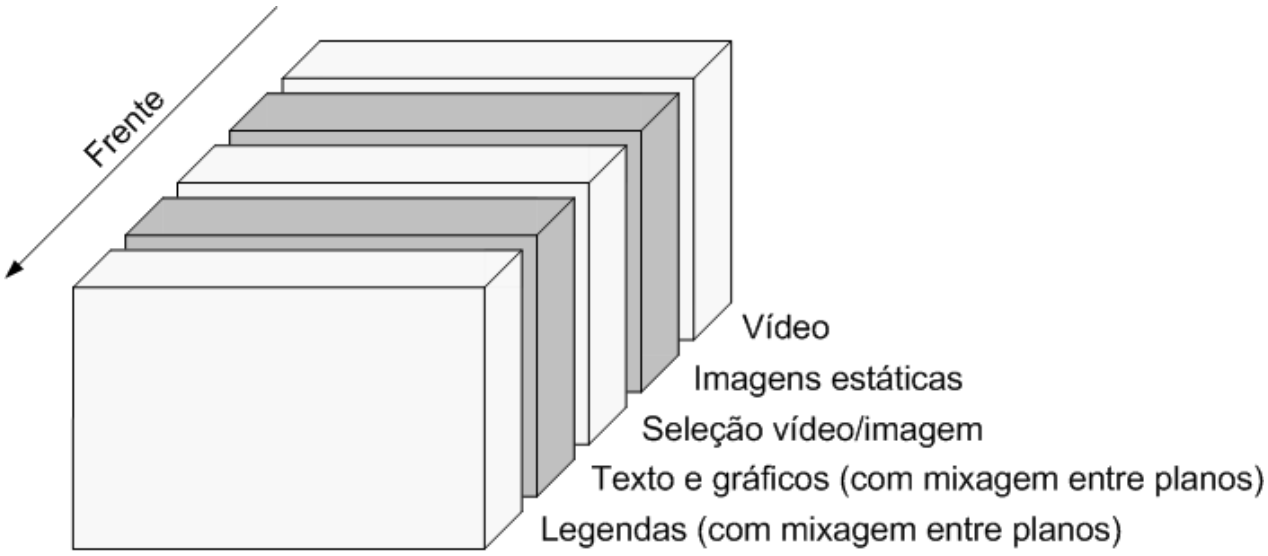


Figura 19 - Planos gráficos do Ginga (ABNT 15606-1)

O plano de legendas não é acessível por aplicações Ginga, sendo uma característica nativa do receptor. Restam, portanto, quatro planos sobre os quais uma aplicação pode operar. Para cada um destes planos, é possível obter suas características e efetuar operações gráficas sobre eles. São eles de acordo com (ABNT 15606-4):

- **Plano de Texto e Gráficos**: é aquele sobre o qual a aplicação pode desenhar elementos gráficos (primitivas geométricas e imagens) com alta definição de cores e canal de transparência, sobre o vídeo.

- **Plano de Seleção Vídeo/Imagem**: permite definir áreas de precedência entre o plano de imagens estáticas e o plano de vídeo. Isto é, em que áreas retangulares da tela o plano de imagens estáticas será exibido sobre o plano de vídeo, ou vice-versa.
- **Plano de Imagens Estáticas**: Este é o plano sobre o qual a aplicação pode exibir imagens de alta resolução e profundidade de cores no formato JPEG. Tipicamente, ele é usado para definir um plano de fundo para a aplicação em telas onde o vídeo é redimensionado. Entretanto, através do uso conjunto com o plano de seleção vídeo/imagem, ele pode ser usado para exibir imagens JPEG em alta resolução sobre o vídeo.
- **Plano de Vídeo**: O plano de vídeo exibe o fluxo elementar de vídeo do serviço, ou opcionalmente, monomídias de vídeo. O conteúdo exibido neste plano pode ser manipulado através de controles JMF, por exemplo.

2.3.1.2 Ginga-CC

O Núcleo Comum do Ginga (*Ginga Common Core*) é o subsistema do Ginga responsável por oferecer funcionalidades específicas de TV Digital comuns para os ambientes imperativo e declarativo, abstraindo as características específicas de plataforma e *hardware* para as outras camadas acima. Como suas principais funções, podemos citar a: exibição e controle de mídias, o controle de recursos do sistema, canal de retorno, dispositivos de armazenamento, acesso as informações de serviço, sintonização de canais, entre outros (KULESZA *et al.* 2010).

A arquitetura do sistema garante que apenas o módulo Ginga-CC deva ser adaptado à plataforma onde o Ginga será embarcado. O Ginga-CC provê, assim, um

nível de abstração da plataforma de *hardware* e sistema operacional, acessível através de APIs bem definidas (SOARES, 2009).

Decodificadores de conteúdo comuns servem tanto às aplicações procedurais quanto às declarativas que necessitam decodificar e apresentar tipos comuns de conteúdo como PNG, JPEG, MPEG e outros formatos. O Ginga-CC é composto pelos decodificadores de conteúdo comuns e por procedimentos para obter conteúdos transportados em fluxos de transporte (*transport streams*) MPEG-2 e através do canal de interatividade. O Ginga-CC também deve obrigatoriamente suportar o modelo conceitual de exibição, conforme descrito na ABNT NBR 15606-1 (DAMASCENO, 2008). Os componentes básicos do Ginga-CC são:

- **Sintonizador:** responsável pela sintonização do canal, escolhendo um canal físico e um dos fluxos de transporte que estão sendo enviados neste canal. É capaz de distinguir os serviços de um canal e receber os conteúdos.
- **Filtro de Seções:** Responsável pelo acesso a partes específicas do fluxo de transporte disponibilizando-as aos demais componentes/aplicações.
- **Processador de Dados:** Oferece suporte para a obtenção de dados, obtidos através de seções especiais MPEG-2. Monitora informações que identificam a exitência de aplicações interativas. As aplicações multiplexadas no fluxo recebidos pelo sintonizador serão obtidas por este módulo.
- **Persistência:** Provê a funcionalidade de armazenar arquivos e dados requisitados pelas aplicações.
- **Adaptador do A/V Principal:** Possibilita que as aplicações acessem o fluxo de áudio e vídeo. Útil quando as ações de uma aplicação dependem do conteúdo sendo veículado.

- **Gerenciador de Gráfico:** Responsável pelo gerenciamento do modelo concetiual do plano grafico de apresentação. É ele que define o plano de exibição do vídeo principal, os planos de exibição dos outros objetos de mídia que compõem uma aplicação e como esses planos se sobrepõem. Em outras palavras, define como as imagens, vídeos, dados e outros elementos são apresentados ao usuário.
- **Gerenciador de Atualizações:** Possui a incumbência de gerenciar a atualização do *middleware* como um todo. Suas ações devem ser transparentes ao usuário.
- **Exibidor de Mídia:** Necessário para exibir as diversas mídias que podem ser áudio, vídeo, texto e imagem.
- **Interface com o Usuário:** Componente que recebe e interpreta os eventos gerados pelo usuário através do controle remoto ou outros dispositivos.
- **Gerenciamento de Contexto:** Responsável por traçar um perfil do usuário através da análise de suas preferências. Este perfil é disponibilizado ao Ginga-NCL e Ginga-J para eventuais adaptações de conteúdos.
- **Canal de Retorno:** Todo o acesso a dados externos bem como a comunicação entre *set-top box* (dispositivo utilizado pelo expectador) deve ser feita através do canal de retorno ou, também chamado, canal de interatividade.
- **Acesso Condicional:** Analisa conteúdos recebidos pelos canais de porgramação, caso seja detectado uma ameaça uma restrição é gerada. Determina também os privilégios de acesso às diversas mídias que compõem uma aplicação.

2.3.1.3 Ginga-NCL

O Ginga-NCL (ABNT 15606-2) é o subsistema "lógico" do *middleware* Ginga que processa documentos NCL. Entre seus módulos chave está o Formatador NCL, que é o responsável por receber um documento NCL e controlar sua apresentação, fazendo com que as relações de sincronismo entre os objetos de mídia existentes sejam respeitadas.

Diferente do HTML, ou XHTML, a linguagem NCL não mistura a definição do conteúdo de um documento com sua estruturação, oferecendo um controle não invasivo, tanto do leiaute do documento (apresentação espacial), quanto da sua apresentação temporal. Como tal, NCL não define nenhum objeto de mídia, mas apenas a "cola" que mantém esses objetos semanticamente juntos em uma apresentação multimídia.

O modelo da linguagem NCL visa um domínio de aplicações mais amplo do que o oferecido pela linguagem XHTML. NCL visa não apenas o suporte declarativo à interação do usuário, mas o sincronismo espacial e temporal em sua forma mais ampla, tratando a interação do usuário como um caso particular. NCL visa também o suporte declarativo a adaptações de conteúdo e de formas de apresentação de conteúdo, o suporte declarativo a múltiplos dispositivos de exibição e a edição/produção da aplicação em tempo de exibição, ou seja, ao vivo (SOARES, 2009). Para os poucos casos particulares, como por exemplo, quando a geração dinâmica de conteúdo é necessária, NCL provê o suporte de sua linguagem de *script* Lua. Alternativamente, as APIs da ponte com o Ginga-J podem ser usadas acionando o suporte imperativo oferecido pela linguagem Java (SOARES, 2009).

Como dito anteriormente esta linguagem não define nenhum tipo de mídia, podendo trabalhar com imagens, vídeo, áudio, texto e código imperativo. Os objetos de mídia suportados são aqueles cujo os exibidores estão acoplados ao Formatador NCL.

Dentro dos exibidores obrigatórios do padrão brasileiro encontra-se o decodificador MPEG-4, implementado em *hardware* no receptor de televisão digital e o exibidor XHTML já que estes podem ser embutidos na NCL.

2.3.1.3.1 Linguagem Lua

Lua é inteiramente projetada, implementada e desenvolvida no Brasil, por uma equipe na PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro). Lua nasceu e cresceu no Tecgraf, o Grupo de Tecnologia em Computação Gráfica da PUC-Rio. Atualmente, Lua é desenvolvida no laboratório Lablua. Tanto o Tecgraf quanto Lablua são laboratórios do Departamento de Informática da PUC-Rio.

Lua é uma linguagem de programação poderosa, rápida e leve, projetada para estender aplicações. Lua combina sintaxe simples para programação procedural com poderosas construções para descrição de dados baseadas em tabelas associativas e semântica extensível. Lua é tipada dinamicamente, é interpretada a partir de *bytecodes* para uma máquina virtual baseada em registradores, e tem gerenciamento automático de memória com coleta de lixo incremental. Essas características fazem de Lua uma linguagem ideal para configuração, automação (*scripting*) e prototipagem rápida (LUA; 2011).

Dentro do ambiente declarativo existe também a máquina de execução Lua responsável pelo processamento do código imperativo Lua.

2.3.1.4 Ginga-J

Tendo o seu desenvolvimento iniciado a partir do FlexTV, o Ginga-J, obrigatório em dispositivos fixos, é composto por um conjunto de APIs definidas para

atender todas as funcionalidades necessárias para a implementação de aplicativos para TV Digital, desde a manipulação de dados multimídia até protocolos de acesso. Sua especificação é formada por uma adaptação da API de acesso a informação de serviço do padrão japonês (ARIB B.23), pela especificação Java DTV (que inclui a API JavaTV), além de um conjunto de APIs adicionais de extensão ou inovação (PAULINELLI; KULESZA, 2012) .

As APIs adicionais incluem o conjunto de classes disponíveis para a ponte entre os aplicativos escritos nas linguagens NCL e Java, funcionalidades adicionais para sintonia de canais e envio de mensagens assíncronas, acesso pelo canal de interatividade e um conjunto de comandos para a integração de dispositivos externos ao *middleware*, viabilizando o suporte a recursos multimídia e interação simultânea de múltiplos usuários em aplicações de TV Digital (SILVA, 2007; SILVA, 2008).

Inicialmente o Ginga-J era baseado no GEM (*Globally Executable MHP*), entretanto, devido a uma incerteza quanto ao modelo *royalties* para uso das API's do GEM, o Forúm Brasileiro de TV Digital em parceria com a Sun Microsystems (atualmente Oracle) desenvolveu AP's equivalentes às do GEM. Como resultado deste trabalho em conjunto chegou-se ao JavaDTV (ABNT 15606-6). Atualmente a arquitetura do Ginga-J está consolidada e é mostrada na Figura 20.

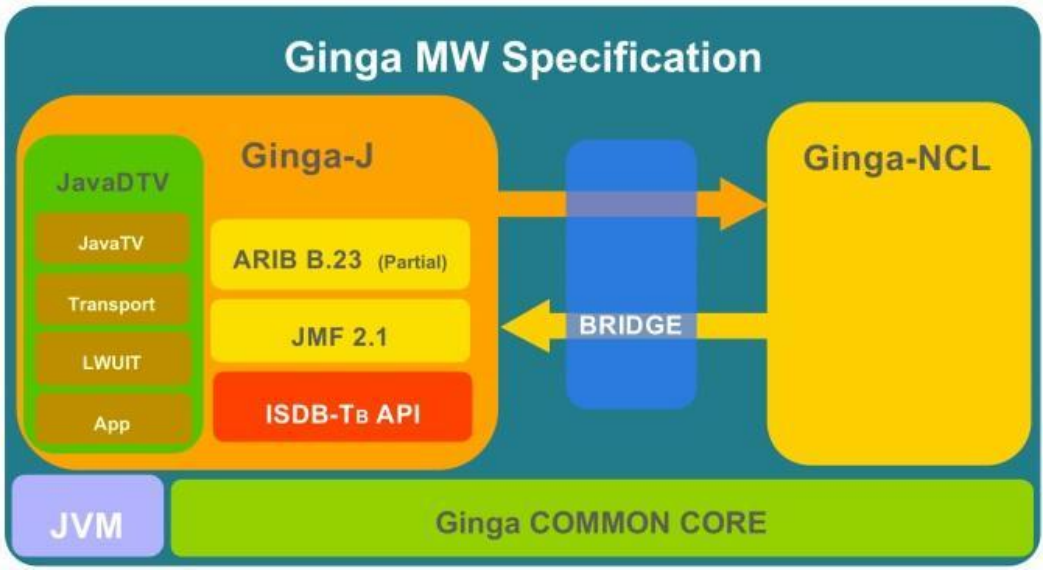


Figura 20 - Arquitetura Ginga-J. Fonte: (PAULINELLI; KULESZA, 2012)

A especificação JavaDTV é uma plataforma aberta e interoperável que permite a implementação de serviços interativos com a linguagem Java. Funcionalmente, a JavaDTV substitui a coleção de APIs utilizada anteriormente e definidas pelo padrão GEM (*Globally Executable MHP*), tais como DAVIC (*Digital Audio Video Council*) e HAVi (*Home Audio Video Interoperability*). O objetivo é prover uma solução livre de *royalties* para os fabricantes de dispositivos e desenvolvedores de aplicações e permitir a produção de aparelhos de TV e/ou conversores por um custo mais acessível. Tal especificação é composta pelas APIs JavaDTV e JavaTV, adicionadas ao ambiente de execução Java (*Java Runtime*) para sistemas embarcados (JavaME), incluindo a configuração *Connected Device Configuration* (CDC), e perfis: *Foundation Profile* (FP) e *Personal Basis Profile* (PBP) como mostrado na Figura 21. Dentre as principais diferenças da JavaDTV quanto ao desenvolvimento de aplicações, podemos citar a API LWUIT (*LightWeight User Interface Toolkit*), responsável por definir elementos

gráficos, extensões gráficas para TVD, gerenciadores de layout e eventos do usuário. O objetivo é substituir a API HAVi do GEM (PAULINELLI; KULESZA, 2012).

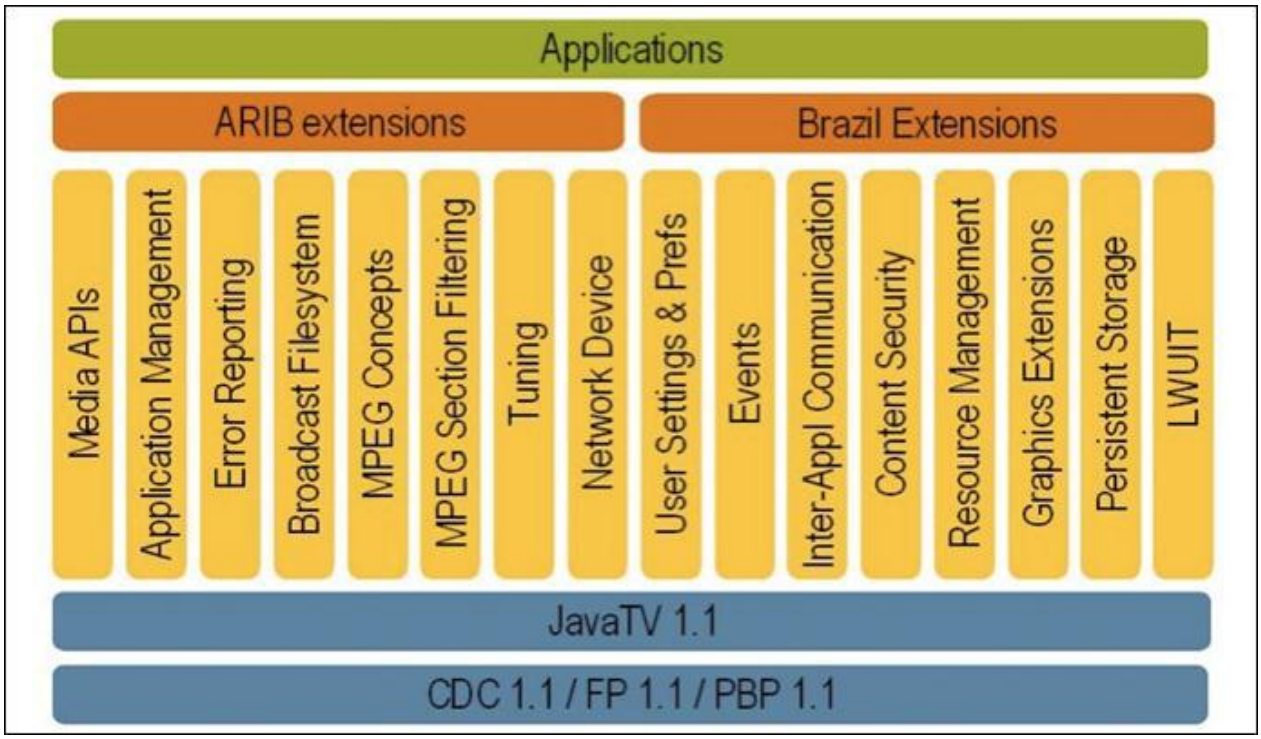


Figura 21 - APIs Ginga-J. Fonte: (PAULINELLI; KULESZA, 2012)

Levando em conta os objetivos do SBTVD, o *middleware* brasileiro teve de prover algumas funcionalidades que não são encontradas no demais *middlewares* de TV digital. Para suprir as necessidades brasileiras bem como manter compatibilidade internacional, o Ginga é baseado em três conjuntos de APIs: verde, amarelo e vermelho. O conjunto de APIs que compõem o módulo verde são aquelas compatíveis com o JavaDTV (substituto do GEM). As APIs do módulo amarelo, por sua vez, são extensões propostas para suprir as necessidades brasileiras e que podem ser implementadas através de uma adaptação das APIs que compõem o módulo verde. Já as APIs que compõem o módulo vermelho são aquelas que não são compatíveis, em software, com as APIs do GEM . Sendo assim, as aplicações "verdes" podem ser executadas em outros

middlewares desde que sejam migradas do JavaDTV para o GEM. As aplicações "amarelas" poderão ser executadas nos demais *middlewares* mediante a presença de um adaptador e, por fim, as aplicações "vermelhas", que representam as inovações brasileiras, poderão ser executadas apenas no Ginga. A Figura 22 expõe a divisão do Ginga-J nos diferentes módulos.

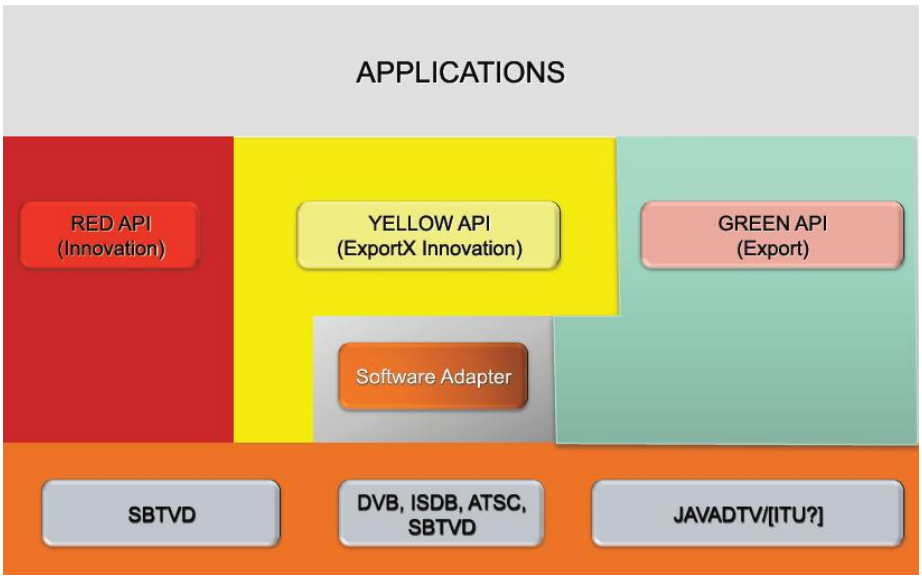


Figura 22 - Módulos Ginga-J. Fonte: (JUNIOR SARAIVA, 2010)

A norma do Ginga-J (ABNT 15606-4) estabelece um conjunto de pacotes mínimos para a execução de aplicações, disponibilizando uma série de APIs que podem ser divididas em sete partes. As mesmas são assim definidas por (JUNIOR SARAIVA, 2010):

- **Pacotes da plataforma JavaME (*Java Micro Edition*)** – funcionalidades da plataforma JavaME providas pela configuração CDC 1.1 (*Connected Device Configuration*) e os perfis FP 1.1 (*Foundation Profile*) e PBP 1.1 (*Personal Basis Profile*);

- **Pacotes da especificação JSSE (*Java Secure Socket Extension*) 1.0.1** – funcionalidades opcionais de segurança para a plataforma básica de Java para TV Digital, como os protocolos de transporte seguro;
- **Pacotes da especificação JCE (*Java Cryptography Extension*) 1.0** – implementa especificamente funcionalidades de segurança relacionadas a criptografia;
- **Pacotes da especificação SATSA (*Security and Trust Services API*) 1.0.1** – permite a comunicação com dispositivos externos, como *smartcards*, utilizando o protocolo APDU (do inglês, *Application Protocol Data Unit*);
- **Pacotes da especificação JavaTV 1.1** – modelo de gerenciamento de aplicações, provendo funcionalidades específicas para TV Digital, além de incluir a API JMF (*Java Media Framework)*;
- **Pacotes da especificação JavaDTV 1.3** – funcionalidades específicas de TV Digital como tratamento de eventos do controle remoto e interface gráfica com LWUIT (*Lightweight User Interface Toolkit*). Os principais pacotes dessa especificação são: com.sun.dtv.application, com.sun.dtv.broadcast, com.sun.dtv.filtering, com.sun.dtv.io, com.sun.dtv.locator, com.sun.dtv.media, com.sun.dtv.net, com.sun.dtv.platform, com.sun.dtv.resources, com.sun.dtv.service, com.sun.dtv.smartcard, com.sun.dtv.transport, com.sun.dtv.tuner, com.sun.dtv.tuner. com.sun.dtv.ui e com.sun.dtv.lwuit;
- **Pacotes específicos do Ginga-J** – funcionalidades exclusivas do sistema brasileiro de TV Digital. São eles: br.org.sbtvd.net, br.org.sbtvd.net.si (ARIB B.23), br.org.sbtvd.net.tuning (*Tuner Control API*), br.org.sbtvd.bridge,

br.org.sbtvd.net.rc (*Return Channel API*), br.org.sbtvd.ui (*Graphics Planes API*) e br.org.sbtvd.interactiondevices (*Device Integration API*).

2.3.1.4.1 Xlet

As aplicações do ambiente Ginga-J devem possuir uma classe principal que implementa a interface **javax.tv.xlet.Xlet.** O conceito de aplicações Xlet é semelhante ao modelo Applet, já amplamente utilizado em páginas *web*, e ao modelo MIDlet, utilizado em aplicações Java para dispositivos móveis (NETO, 2010). A interface estabelece um conjunto de métodos que serão chamados de acordo com o estado do Xlet, ou seja, representam o ciclo de vida da aplicação. A Figura 23 mostra os métodos bem como os estados associados a eles:

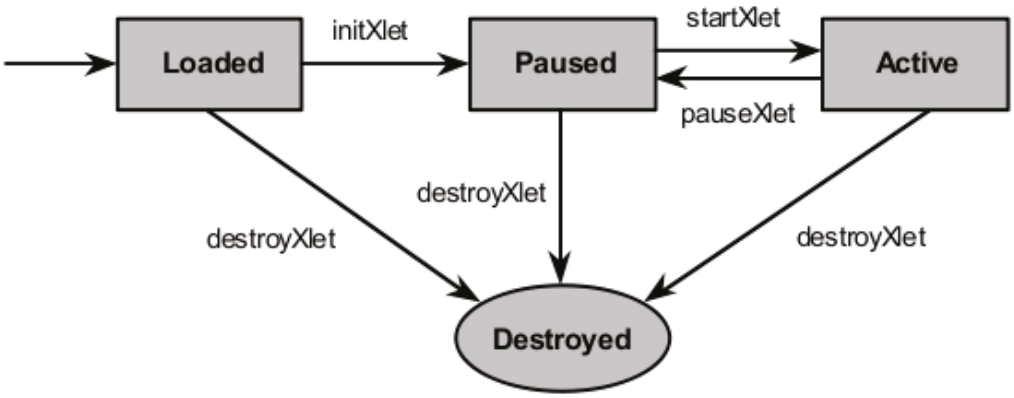


Figura 23 - Ciclo de vida de aplicações Ginga-J (Xlet). Fonte: (ABNT 15606-4).

Os estados não carregada e inválido, podem, também, ser considerados em alguns contextos. Inicialmente o receptor recebe a sinalização de uma nova aplicação, o ciclo de vida da Xlet então começa e é caracterizado como não carregada. A partir deste ponto, o gerenciador da aplicação, que está no receptor, deve detectar a classe principal e criar uma instância dela. Feito isso, o estado passa a ser caracterizado como carregada. Uma aplicação que já foi carregada está apta a ser inicializada pelo método initXlet, que executa suas tarefas e deixa a aplicação no estado pausada. Neste ponto, a Xlet já está pronta para ser executada, pois já carregou e inicializou todos os elementos que eram necessários para começar a executar. O método startXlet é chamado e a aplicação de fato começa a funcionar. O método destroyXlet é chamado toda vez que a aplicação termina suas tarefas (MONTEZ; BECKER; FILHO, 2005). A interface Xlet é mostrada no Anexo A.

Todo Xlet deve possuir um objeto que implementa a interface XletContext que de uma maneira geral serve para isolar a aplicação do resto da máquina virtual. Um objeto XletContext é passado a uma Xlet quando ele é inicializado. É através desta interface que o Gerenciador de Aplicações controla o estado de um Xlet. Entre outras, o contexto permite a Xlet descobrir informações a respeito do ambiente de execução (MONTEZ; BECKER; FILHO, 2005).

Uma aplicação Ginga-J, em geral, além da classe Xlet, que é o ponto de partida, possui outras classes.

2.3.1.4.2 Eventos

O mecanismo de tratamento de eventos do usuário é provido por componentes especializados em televisão e dos componentes providos pela LWUIT 1.1:2008

incorporada ao JAVADTV 1.3:2009. Os pacotes que definem estes componentes são de acordo com (ABNT 15606-4):

com.sun.dtv.ui.event – tem a função de tratamento de eventos de interface gráfica com o usuário específica para televisão digital.

com.sun.dtv.lwuit.events – mecanismo de tratamento de eventos relacionados aos componentes gráficos definidos na LWUIT 1.1:2008 incorporada ao JAVADTV 1.3:2009.

Todo o mecanismo de tratamento de eventos do usuário é baseado no modelo EDT (*Event Dipatch Thread*), dando à plataforma toda a ciência dos eventos e repassando às aplicações o seu tratamento especializado (ABNT 15606-4).

Durante o desenvolvimento da aplicação, é uma boa prática produzir uma classe para cada tela e nela encapsular tanto a confecção da interface quanto o tratamento dos eventos relacionados à tela em questão (JUNIOR GADELHA, 2010).

A classe responsável pelos eventos provenientes do usuário é a com.sun.dtv.ui.event.UserInputEventManager. Esta classe trabalha em conjunto com a interface com.sun.dtv.ui.event.UserInputEventListener que deve ser implementada pelas classes a fim de receber UserInputEvents. A instância da classe de implementação deve ser registrada no UserInputEventManager atribuído à tela, utilizando o método UserInputEventManager.addUserInputEventListener(com.sun.dtv.ui.event.UserInputEventListener, com.sun.dtv.ui.event.UserInputEvent) para fazer com que seja notificado sobre o UserInputEvent especificado.

2.3.1.4.3 Interfaces Gráficas

As aplicações Ginga-J, utilizam-se do pacote **com.sun.dtv.lwuit** para a elaboração de interfaces gráficas. Ao contrário do Swing/AWT um sistema completo de janelas não é aplicável nesse caso e as formas são colocadas usando um DTVContainer passado em abstrações *Plane* e *Screen* (ABNT 15606-6).

Os componentes são colocados em um container com gerenciadores de layout que são usados para determinar o posicionamento dos componentes, containers podem ser aninhados profundamente de maneira similar a Swing/AWT. Todos os componentes são simples e desenhados pelo UIManager o que permite tematizar tudo usando estilos. UIManager é o *singleton*[8] do ponto central gerenciando a aparência do aplicativo. Essa classe permite personalizar os estilos (temas) e a aparência dos todas as instâncias de um dado componente. Também permite personalização de UI elaborada ao se obter o LookAndFeel e sobrepondo-se métodos específicos para desenhar/dimensionar componentes (ABNT 15606-6). A Figura 24 mostra a estrutura do pacote LWUIT do JavaDTV:

[8] Padrão de projeto em que a implementação deve garantir uma única instância de uma determinada classe na aplicação.

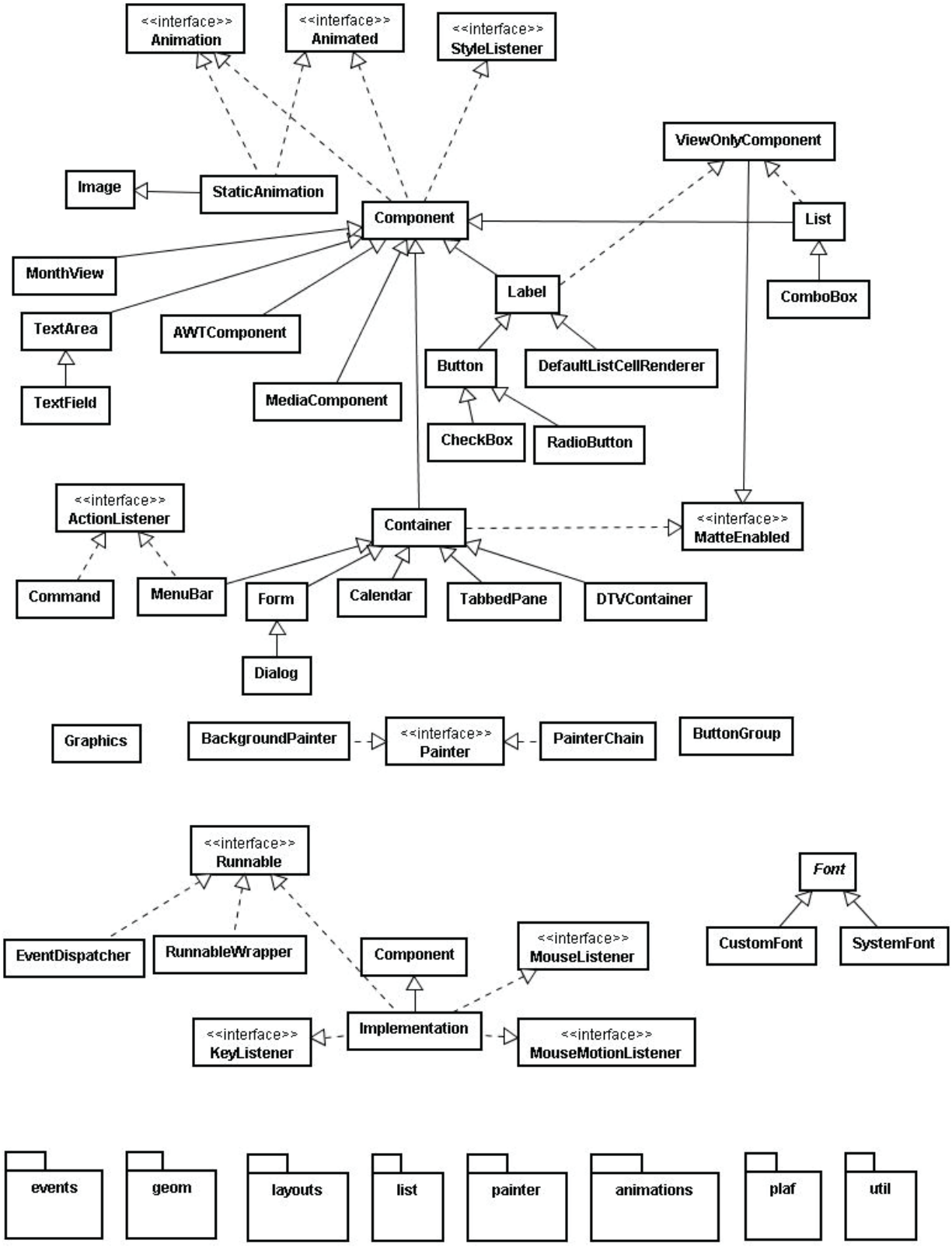


Figura 24 - Pacote LWUIT

Alguns componentes, aqueles utilizados com maior frequência, são assim definidos por (ABNT 15606-6):

- **Container:** Um padrão composto com Component, permite aninhamento e a organização de componentes múltiplos usando uma arquitetura de gerenciador de *layout* plugável. Containers podem ser aninhados uns dentro dos outros para formar interfaces de usuário (UI) elaboradas.
- **Form:** componente de nível mais alto que serve como a entidade mais visível no UI. O formulário contém barra de título, barra de menu e um painel de conteúdo.
- **Dialog:** Um diálogo é um formulário que ocupa parte da tela e aparece como uma entidade modal ao desenvolvedor. Dialogs permitem prontificar usuários para informações e confiar na informação sendo disponibilizada na próxima linha após o método de exibição
- **TabbedPane:** Um componente que deixa o usuário trocar entre um grupo de componentes ao clicar em uma aba com um dado título e/ou ícone.
- **Label:** Rótulos que podem conter textos e imagens.
- **Button:** Botões herdam a classe Label. Usa-se um ActionListener para descobrir quando o botão foi pressionado.
- **RadioButton:** RadioButton é um Button que mantém um estado de seleção exclusivamente dentro de um grupo específico (ButtonGroup).
- **CheckBox:** Checkbox é um botão que pode ser selecionado e desselecionado e que exibe seu estado ao usuário.
- **TextArea:** Uma região editável com opção de possuir múltiplas linhas que pode exibir texto e permitir um usuário a editá-lo. Dependendo da plataforma, a edição pode ocorrer em uma nova tela.

- **TextField:** Permite edição no local usando uma API simples sem necessariamente mover-se a caixa de texto nativa externa. O principal defeito nessa abordagem é que a edição não pode suportar recursos como o T9 e pode não ter o mesmo mapeamento principal ou comportamento da entrada do texto original.
- **List:** Um conjunto de elementos (lista) que são processados usando um ListCellRenderer e são extraídos por meio do ListModel.
- **ListModel:** Representa a estrutura de dados da lista, permitindo assim que a lista represente qualquer fonte de dados potenciais por referenciar implementações diferentes dessa interface.
- **ListCellRenderer:** Componente responsável por desenhar um componente da lista.
- **ComboBox:** Um combo Box é uma lista que permite apenas uma seleção por vez, quando um usuário clica no *combo box*, um botão *popup* com a lista completa de elementos permite a seleção de um único elemento. A combo Box é guiada pelo modelo de lista que também permite todas as funções exibidoras da Lista.

2.4 Redes Sociais

Em linhas gerais, o estudo das redes sociais é o estudo dos "agrupamentos sociais estabelecidos através da interação mediada pelo computador" (RECUERO, 2006). Uma rede social é composta por dois elementos: os atores (ou seja, as pessoas, instituições ou grupos, que constituem os nós da rede) e suas conexões (ou seja, as relações que se estabelecem entre os indivíduos) (RECUERO, 2006).

Desde o início, sites de redes sociais (SRS), tais como Facebook, Orkut, MySpace, LinkedIn e outros tem atraído milhões de usuários, integrando estas redes em suas práticas cotidianas. Há centenas de SRS disponíveis na internet, com vários recursos tecnológicos, suportando uma gama de interessantes práticas de seus utilizadores. Enquanto que seus principais recursos tecnológicos são bastante consistentes, as culturas que emergem em torno dos SRS são variadas (BOYD; ELLISON, 2007). A maioria dos sites dão suporte na manutenção dos grupos de amigos pré-existentes, entretanto, existem outros que fazem pessoas estranhas se relacionarem entre si baseados em interesses comuns, opiniões políticas ou atividades. Os sites de redes sociais variam na medida em que incorporam novas informações e ferramentas de comunicação, tais como conectividade móvel, blogs e compartilhamento de vídeos/fotos.

Redes sociais possuem conexões constituídas através de diferentes formas de interação e trocas sociais. Na Internet, por exemplo, é possível "assinar" uma lista de discussão, ou seja, participar de um grupo social sem interagir diretamente com seus membros, mas unicamente usufruindo das informações que circulam (RECUERO, 2007).

Sites de redes sociais podem ser definidos como serviços baseados na web que permitem aos indivíduos (1) construir um perfil público ou semi-público[9] dentro de um sistema limitado, (2) articular uma lista de outros usuários com os quais compartilham uma conexão, e (3) ver e pesquisar sua lista de conexões e aquelas feitas por outros dentro do sistema. A natureza da nomenclatura dessas conexões podem variar de site para site (BOYD; ELLISON, 2007).

Enquanto SRS têm implementado uma ampla variedade de características técnicas, seu principal pilar é composto de perfis visíveis que exibem uma lista de amigos, os quais também são usuários do sistema e possuem sua lista de amigos (BOYD; ELLISON, 2007). Os perfis são páginas exclusivas onde podem ser editadas informações sobre sua pessoa. Depois de cadastrar-se em um SRS, é solicitado o preenchimento de formulários contendo uma série de perguntas. O perfil é gerado usando as respostas a estas questões, que geralmente incluem descritores tais como localização, idade, interesses, e um "quem sou eu". A maioria desses sites também incentivam os usuários a fazer o *upload* de uma foto de perfil para que seja facilmente identificável, alguns disponibilizam ferramentas para adicionar conteúdos multimídia e em outros, tal qual o Facebook, é possível instalar aplicativos (*apps*) associados ao seu usuário que permitem executar inúmeras ações dentro e fora da rede social.

2.4.1 Histórico dos sites de redes sociais

[9] O perfil semi-público é aquele em que o usuário restringe o acesso de terceiros a determinadas seções do seu perfil. Exemplo: restringir o acesso ao albúm de fotos apenas aos seus amigos daquela rede social.

O primeiro e reconhecido site de rede social foi lançado em 1997. O SixDegrees.com[10] permitia que os usuários criassem perfis, lista de seus amigos e, a partir de 1998, a possibilidade de navegar nessa lista. Cada uma dessas características existia de alguma forma antes de SixDegrees ser lançado. Os perfis existiam na maioria dos grandes *sites* de namoro e em muitos *sites* de comunidades na *web*. O AIM (acrônimo para AOL *Instant Messenger*) e o ICQ continham listas de amigos, embora tais listas não fossem visíveis para outros. Classmates.com permitiu pessoas autorizadas a afiliarem-se com sua escola ou faculdade e disponibilizou a navegação nessa rede social e a visualização dos que também foram filiados, mas os usuários não podiam criar perfis ou lista de amigos. SixDegrees foi o primeiro a combinar estas características, e a capacidade de enviar mensagens privadas a seus contatos (BOYD; ELLISON, 2007).

Anos depois do pioneiro *site* de redes sociais, muitos outros foram criados, com variadas características e ferramentas que facilitavam a interatividade entre os participantes. A Figura 25 contém a linha do tempo que demonstra a criação desses serviços até o ano de 2006. Pode ser acrescentado também a criação do Google Wave e Google+, do Google, em 2009 e 2011 respectivamente.

[10] SixDegrees: http://www.sixdegrees.com

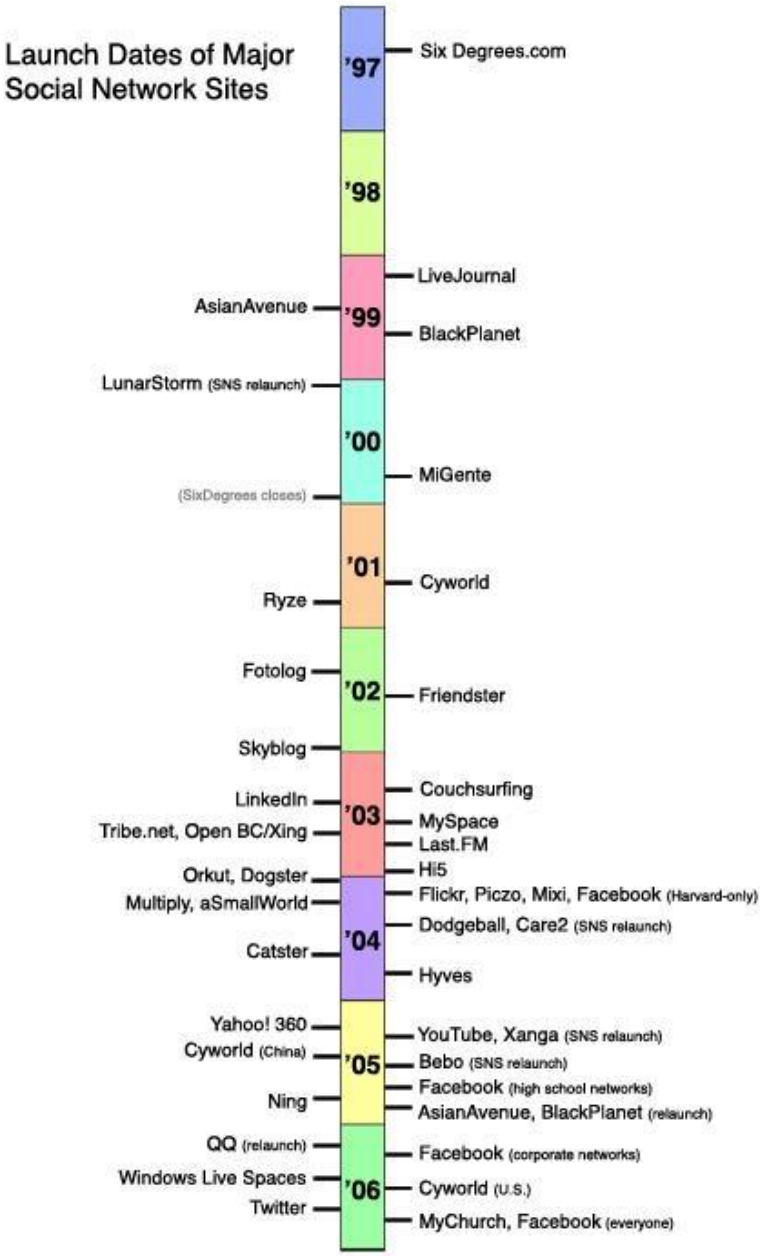


Figura 25 - Data de Lançamento dos maiores sites de redes sociais. Fonte: (BOYD; ELLISON, 2007)

Atualmente, quando falamos de redes sociais no mundo, vem a cabeça o maior SRS utilizado, o Facebook. O infográfico da Figura 26 divulgado pelo portal de notícias

russo Rian Novosti[11] demonstra o domínio desse *website* principalmente na América, Europa, África e Oceania.

Mudando o foco para o Brasil, de acordo com a medição do Instituto de pesquisas IBOPE Nielsen Online[12], o Orkut que esteve na liderança de acessos diários durante 7 anos, perdeu seu posto em meados de agosto de 2011 (GALO, 2011). A partir dessa data, o Facebook ocupa a primeira colocação, assim como o faz em todo o restante do mundo, verificado na Figura 26.

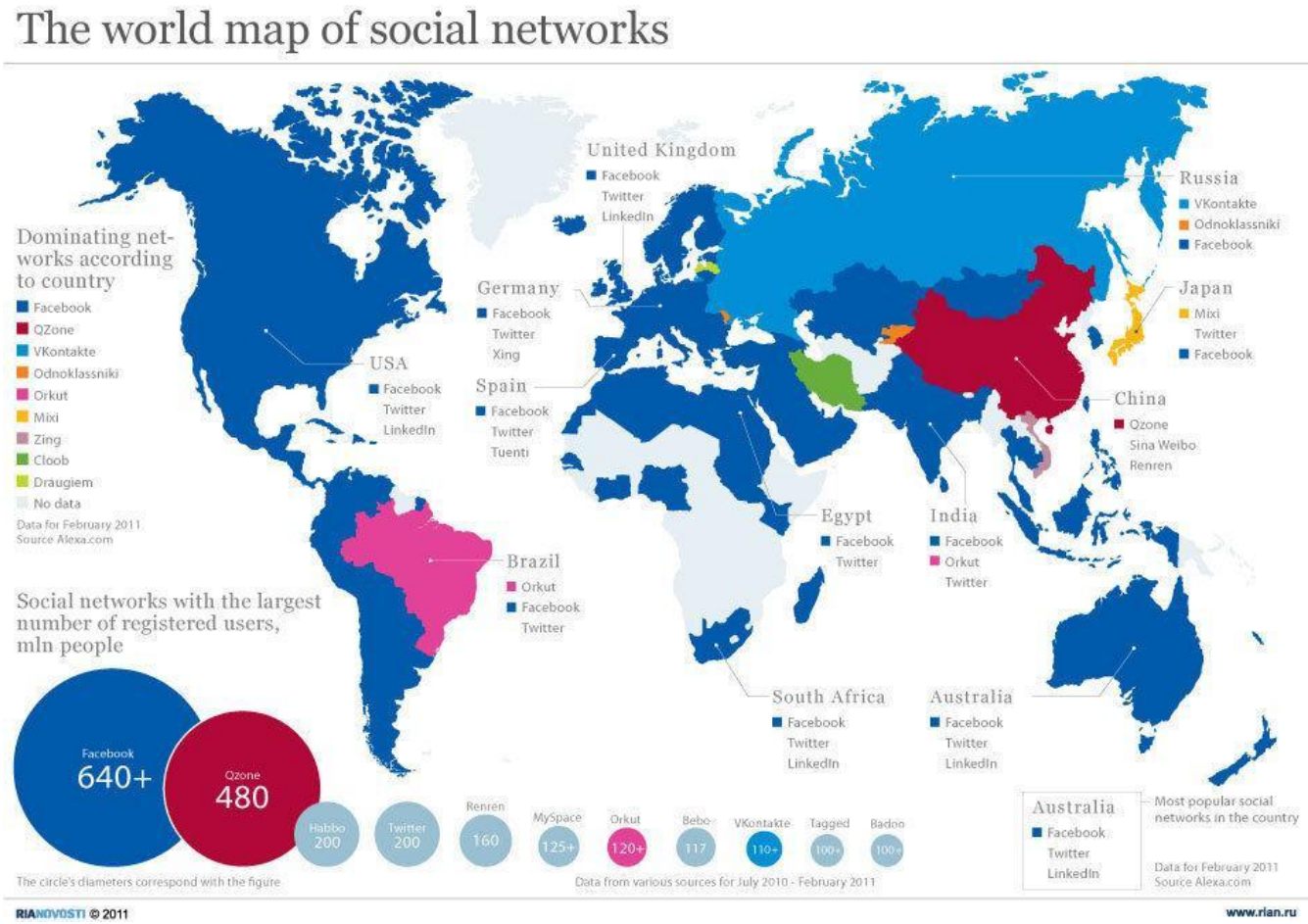


Figura 26 - Infográfico representando o domínio das redes sociais no mundo. Fonte: (RIAN NOVOSTI, 2011)

[11] Rian Novosti: http://en.rian.ru/

[12] IBOPE Nielsen: http://br.nielsennetpanel.com/pnl/br/home

2.4.2 Facebook: o maior Site de Redes Sociais do planeta

2.4.2.1 O que é o Facebook

De acordo com a descrição em seu próprio site:

> *O Facebook é uma rede social que ajuda as pessoas a se comunicarem com mais eficiência aos seus amigos, familiares e colegas de trabalho. A empresa desenvolve tecnologias que facilitam o compartilhamento de informações através do gráfico social, o mapeamento digital das conexões sociais entre pessoas do mundo real. Qualquer um pode se registrar no Facebook e interagir com pessoas conhecidas em um ambiente confiável. Facebook é um pedaço da vida de milhões de pessoas que vivem em todas as partes do mundo. Facebook é uma empresa privada e está sediada em Palo Alto, Califórnia.*

2.4.2.2 Estatísticas Facebook

Atualmente, o Facebook possui sede em diversos países. A tabela 1 apresenta dados interessantes dessa plataforma, obtidos da própria página de estatísticas do Facebook:

Pessoas no FB	- Um bilhão de usuários ativos mensais a partir de outubro de 2012 - Mais de 50% dos usuários acessam o *site* por dia - Um usuário médio possui 130 amigos
Atividade no FB	- Mais de 900 milhões de objetos utilizados para interagir entre as

	pessoas (páginas, grupos, eventos e comunidades) - Um usuário médio está conectado a 80 páginas de comunidades, grupos e eventos - Em média, mais de 250 milhões de fotos são carregadas por dia
Alcance Global	- *Site* disponível em mais de 70 idiomas - Aproximadamente 80% dos usuários estão fora dos EUA - Mais de 300 mil usuários ajudaram a traduzir o site através da aplicação traduções
Plataforma	- Em média, as pessoas no Facebook instalam aplicativos mais de 20 milhões de vezes todos os dias - Todo mês, mais de 500 milhões de pessoas usam um aplicativo no Facebook ou experimentam a Plataforma do Facebook em outros *sites* - Mais de 7 milhões de aplicativos e websites estão integrados com Facebook
Móvel	- Mais de 350 milhões de usuários ativos acessam o Facebook através dos seus dispositivos móveis - Mais de 475 operadoras móveis do mundo trabalham para implantar e promover os produtos Facebook móvel

Tabela 1 - Estatísticas do Facebook (FACEBOOK, 2012a).

2.4.2.3 Gráfico Social do Facebook (*Open Graph*)

No centro do Facebook está o gráfico social (FACEBOOK, 2012b), que representa as pessoas e suas conexões com seus interesses. Quando as pessoas usam a

rede social, elas compartilham suas histórias, as quais aproximam outras pessoas e ajudam na construção de relacionamentos *online* e na vida real.

Quando os usuários do Facebook usam aplicativos online ou em seus dispositivos móveis, eles criam ainda mais histórias: as músicas que ouvem, os filmes que eles assistem, os programas de televisão que assistem e assim por diante. Os aplicativos que conectam ao *Facebook Open Graph* possibilitam aos usuários compartilhar tais histórias com seus amigos.

Tudo isso forma o gráfico social, o qual pode ser acessado através da *Facebook Platform* de inúmeras maneiras (APIs). Por se tratar de um assunto muito amplo, não serão detalhadas todas as APIs fornecidas, limitando apenas às consideradas necessárias e importantes para a conclusão desse trabalho.

O gráfico social em si é um grafo que é definido na ciência da computação e matemática através da Teoria dos Grafos. Não trata-se apenas de uma simples tabela de dados, é uma série de nós os quais conectam-se uns aos outros.

2.4.2.4 Graph API (*Application Programming Interface*) - Facebook

O Graph API (FACEBOOK, 2012c) é a principal ferramenta de acesso ao gráfico social, permitindo ler e gravar dados no Facebook. A API apresenta uma visão simples e consistente do gráfico social Facebook, que representa objetos uniformemente no gráfico (pessoas, fotos, eventos e páginas) e as conexões entre eles (relacionamentos de amizade, conteúdo compartilhado e *tags* de fotos) (FACEBOOK, 2012c).

Ela é uma API de baixo nível com base no protocolo HTTP (*Hypertext Transfer Protocol*) através da qual o usuário pode consultar dados, postar novas histórias, fazer *check-in* (registrar a sua presença em um local) ou qualquer uma das outras tarefas que

um aplicativo possa precisar. O *Open Graph* do Facebook permite que você defina novos objetos e ações no gráfico social de um usuário.

A API é também o mecanismo subjacente que é utilizado pelas ferramentas de desenvolvimento para criar aplicativos para diferentes plataformas e arquiteturas, como por exemplo, iOS e Android, e em diferentes linguagens de programação, como PHP e Javascript.

Todos os tipos de objetos que a API fornece ao desenvolvedor, seguem mencionados na tabela 2.

Achievement (Instância)	A instância desse objeto representa uma conquista (*achievement*) do usuário para uma aplicação específica
Álbum	Álbum de fotos de um usuário
Aplicação	Um aplicativo registrado na Plataforma Facebook
Checkin	Esse objeto representa uma simples visita de um usuário a um local.
Comentário	Um comentário referente a um objeto do Graph API
Domínio	Um domínio de *website* registrado no Graph API
Evento	Um evento registrado no Facebook
Erros	Representa um erro gerado na resposta da Facebook API
Lista de Amigos	A lista de amigos de um determinado usuário
Grupo	Um grupo registrado no Facebook

Insights	Estatísticas sobre as aplicações, páginas ou domínios
Link	Um *link* compartilhado
Mensagem	Uma mensagem individual registada nos grupos de mensagens (*threads)*
Nota	A nota cadastrada no Facebook
Ordem	Um objeto associado a fim Créditos Facebook.
Página	Uma página dentro do Facebook
Foto	Uma foto individual dentro de um álbum
Posto	Uma entrada individual na alimentação de um perfil
Pergunta	A pergunta feita por um usuário, como representado no gráfico API.
QuestionOption	Uma opção permitida como uma resposta a uma pergunta.
Revisão	Uma revisão de um aplicativo
Mensagem de status	Uma mensagem de *status* no mural de um usuário
Assinatura	A assinatura de um aplicativo para obter atualizações em tempo real para um tipo de objeto gráfico.
Thread	Uma *thread* de mensagens (grupo de mensagens)
Ofertas	Representa uma oferta publicada por uma página

Vídeo	Um vídeo individual

Tabela 2 - Objetos acessíveis através da API (FACEBOOK, 2012d)

Cada objeto no gráfico social tem um identificador (ID) único. As propriedades de um objeto podem ser acessadas através de uma requisição a https://graph.facebook.com/ID. Por exemplo, a página oficial para o grupo *Facebook Developers* tem identificador 19292868552, então, para que você possa acessar todos os seus atributos, basta uma requisição para https://graph.facebook.com/19292868552. O serviço irá retornar um objeto no formato JSON (explicado na próxima seção), conforme ilustrado na Figura 27:


```
{
    name: "Facebook Developers",
    is_published: true,
    website: "http://developers.facebook.com",
    username: "FacebookDevelopers",
    company_overview: "Facebook Platform enables anyone to build social apps on Facebook, mobile, and the web. ",
    about: "Build and distribute amazing social apps on Facebook. https://developers.facebook.com/ ",
    talking_about_count: 17345,
    category: "Product/service",
    id: "19292868552",
    link: "http://www.facebook.com/FacebookDevelopers",
    likes: 285328,
  - cover: {
        cover_id: "10151121467948553",
        source: "http://sphotos-f.ak.fbcdn.net/hphotos-ak-ash4/s720x720/299374_10151121467948553_45631061_n.png",
        offset_y: 0
    }
}
```


Figura 27 - Resposta da solicitação ao Facebook em formato JSON

Do mesmo modo, as pessoas e páginas com nomes de usuário podem ser acessadas usando seu nome de usuário como um ID. Uma vez que "FacebookDevelopers" é o nome de usuário para a página acima, https://graph.facebook.com/FacebookDevelopers irá retornar a página correspondente ao grupo.

Com isso, todos os objetos no Facebook podem ser acessados da mesma maneira:

- Perfis de Usuários: https://graph.facebook.com/gustavorupp (Gustavo Rupp Santos)
- Páginas: https://graph.facebook.com/PES (Página do Jogo Pro Evolution Soccer)
- Eventos: https://graph.facebook.com/251906384206 (Facebook Developer Garage Austin)
- Grupos: https://graph.facebook.com/195466193802264 (Desenvolvedores Facebook)
- Aplicações: https://graph.facebook.com/2439131959 (A aplicação Graffiti)
- Mensagens de status: https://graph.facebook.com/367501354973 (A mensagem do *status* de Bret)
- Fotos: https://graph.facebook.com/98423808305 (A foto da página de Coca-Cola)
- Álbuns de fotos: https://graph.facebook.com/99394368305 (*wallpapers* da Coca-Cola)
- Fotos de perfil: http://graph.facebook.com/gustavorupp/picture (Imagem perfil de Gustavo Rupp Santos)
- Vídeos: https://graph.facebook.com/817129783203 (Facebook Tech Talk no Graph API)
- Notas: https://graph.facebook.com/122788341354 (Nota anunciando Facebook para iPhone 3.0)
- Checkins: https://graph.facebook.com/414866888308 (*Check-in* em uma pizzaria)

Algumas dessas URLs necessitam do *Access Token*. De acordo com o Facebook (FACEBOOK, 2012e), um *access token* é uma cadeia de caracteres aleatória que fornece, temporariamente, acesso seguro às APIs do Facebook. Um *token* identifica a sessão de um Usuário, Aplicação ou Página, provê informações sobre as permissões concedidas e também possui dados sobre quando ele irá expirar e qual aplicação gerou esse *token*. Por causa das verificações de privacidade, a maioria das chamadas de API do Facebook precisa estar assinada com um *token*. Todos os *access tokens* são gerados com o protocolo padrão de autorização OAuth 2.0, o qual permite aos usuários compartilhar seus recursos privados armazenados em uma base de dados com um outro site.

Todos os objetos no gráfico social estão ligados uns aos outros através de relacionamentos. Gustavo Rupp Santos é um fã da página do jogo Pro Evolution Soccer, Gustavo e Fiodor Castro são amigos. Essas relações são chamadas de "conexões" na API e elas podem ser acessadas usando a estrutura de URL: https://graph.facebook.com/ID/TIPO_CONEXAO.

A API fornece vários tipos de conexões para acesso aos dados, entretanto, todas necessitam do *access token* válido e de autorização para isso. A *url* base para acesso pode ser https://graph.facebook.com/me/ ou ainda https://graph.facebook.com/ID/ sendo que a palavra *me* representa o usuário autenticado na sessão. Na tabela 3 estão as descrições das conexões fornecidas e a forma com que é possível acessá-las:

Amigos	https://graph.facebook.com/me/friends?access_token=...

***Feed* de Notícias (*Home*)**	https://graph.facebook.com/me/home?access_token=...
Mural (*Wall)*	https://graph.facebook.com/me/feed?access_token=...
Curtir	https://graph.facebook.com/me/like?access_token=...
Filmes	https://graph.facebook.com/me/movies?access_token=...
Música	https://graph.facebook.com/me/music?access_token=...
Livros	https://graph.facebook.com/me/books?access_token=...
Notas	https://graph.facebook.com/me/notes?access_token=...
Permissões	https://graph.facebook.com/me/permissions?access_token=...
***Tags* em Foto**	https://graph.facebook.com/me/photos?access_token=...
Álbuns de Fotos	https://graph.facebook.com/me/albums?access_token=...
***Tags* em Vídeo**	https://graph.facebook.com/me/videos?access_token=...
***Uploads* de Vídeo**	https://graph.facebook.com/me/videos/uploaded?access_token=...
Eventos	https://graph.facebook.com/me/events?access_token=...
Grupos	https://graph.facebook.com/me/groups?access_token=...
Checkins	https://graph.facebook.com/me/checkins?access_token=...

Tabela 3 - Conexões fornecidas pela API Facebook

2.4.2.5 JSON (*Javascript Object Notation*)

JSON[13] (*JavaScript Object Notation* - Notação de Objetos JavaScript) é um padrão para troca de dados. Baseia-se na linguagem JavaScript, Standart ECMA - 262 3ª Edição - Dezembro - 1999. Embora seja baseado em JavaScript, o JSON é completamente independente de linguagem, o que o torna ideal para comunicação entre diferentes sistemas.

Possui dois tipos de estruturas:

- Uma coleção de pares nome/valor. Conhecido em outras linguagens como *Object*, *record*, *struct*, dicionário, *hash table*, *keyed list* ou *arrays* associativos.
- Uma lista ordenada de valores. Conhecido em outras linguagens como *array*, vetor, lista ou sequência.

Estas duas estruturas de dados são praticamente universais, estando presentes em quase todas as linguagens. Um objeto JSON possui um conjunto desordenado de pares nome/valor e possui a sintaxe demonstrada nas figuras 28-a e 28-b:

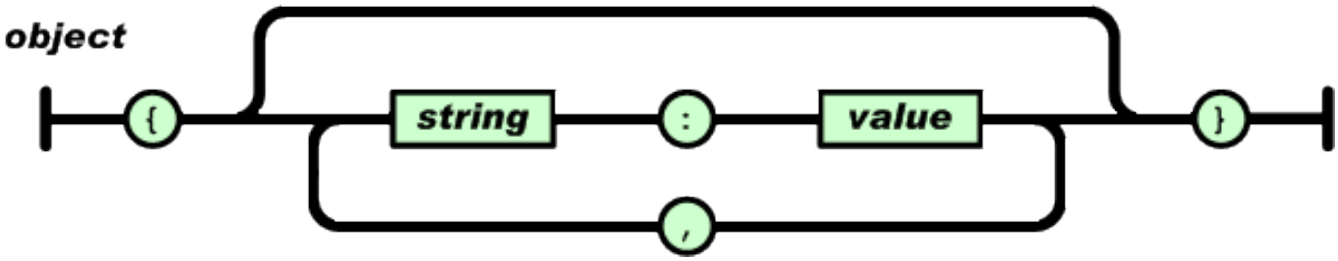


Figura 28-a - Representação de um objeto em JSON. Fonte: (JSON, 2011)

Por sua vez, um *array* é uma coleção de valores ordenados com a sintaxe a seguir:

[13] JSON: http://www.json.org/

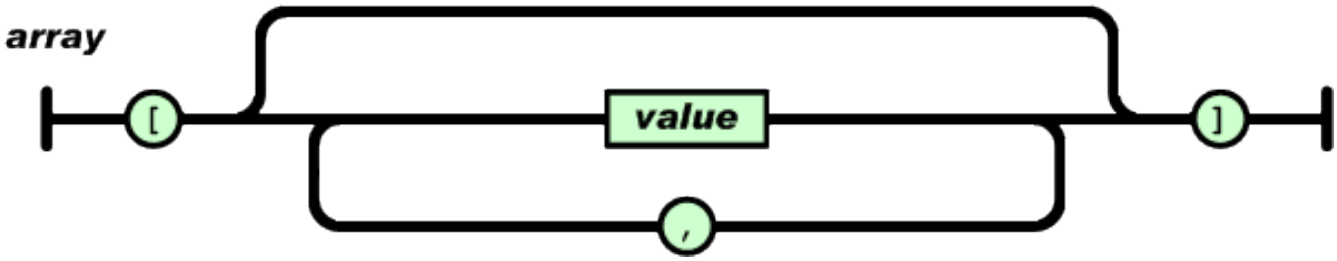


Figura 28-b - Representação de um *array* em JSON. Fonte (JSON, 2011)

As duas estruturas de dados contém valores, e os valores podem ser uma cadeia de caracteres, um número, *true* ou *false*, *nulo*, um objeto ou um array. Estas estruturas podem estar aninhadas.

2.4.2.6 REST

Uma das maneiras que o Facebook utiliza para fornecer seus serviços é através do estilo arquitetural denominado REST (*Representational State Transfer*). Trata-se de uma mescla de outros estilos arquiteturais para sistemas hipermídia distribuídos, acrescentado de características adicionais. De acordo com (TOBALDINI, 2007), os estilos que deram origem ao REST são Cliente com Cache-Servidor sem Estados e Interface Uniforme. No primeiro estilo, o servidor não guarda estados, ou seja, todos os dados referentes a estados devem ser enviados pelo cliente e as respostas podem ser armazenadas para uso em requisições equivalentes (*cache*). O segundo obtêm-se através da aplicação do princípio da generalização de engenharia de *software* à interface dos componentes simplificando e dando melhor visibilidade à arquitetura.

A natureza e o estado dos elementos de dados de uma arquitetura são os aspectos chaves do REST. Quando selecionamos um *link* - na *Web* por exemplo - a informação necessita ser movida do local onde é armazenada para o local onde ela será processada.

Esta característica é distinta de muitos outros paradigmas de processamento distribuído, onde é possível - e geralmente mais eficiente - mover "o processamento" (em forma de uma expressão de busca ou um programa móvel) até os dados, ao invés de levá-los ao processamento (TOBALDINI, 2007).

No REST, os componentes se comunicam através da transferência de representações de um recurso em um formato selecionado dinamicamente de um conjunto de tipos de dados padrão, baseado nas capacidades e desejos do componente que irá receber as informações e da natureza do recurso. Sejam estas representações no mesmo formato que os dados brutos ou derivadas destes, isto permanece transparente (TOBALDINI, 2007).

Em se tratando de REST é fundamental o entendimento do que vem a ser um recurso. Um documento, uma imagem, um serviço podem assumir o papel de um recurso, ou seja, tudo aquilo que pode ser referenciado é um recurso. Uma maneira de se identificar um recurso em REST é através de uma URI utilizada na *web*.

Uma representação em REST é feita através de dados e metadados. Os metadados são apresentados na forma de pares nome-valor, onde o nome corresponde a um padrão que define a estrutura do valor e sua semântica. Mensagens de resposta podem incluir metadados de representação assim como metadados do recurso (informações sobre o recurso que não são específicas à representação fornecida) (TOBALDINI, 2007).

Existem também, nas representações, informações de controle que irão contribuir para definição do propósito de uma mensagem entre componentes ou parametrizar requisições e redefinir o comportamento de alguns elementos. À estrutura dos dados de uma representação dá-se o nome de tipo de informação.

Muitas das características da *Web* podem ser relacionadas ou atribuídas ao REST. Basicamente, toda a estrutura da *Web* consiste em um padrão de localização de recursos (URI), protocolo de comunicação HTTP, linguagens de marcação de conteúdo e outras tecnologias, como servidores de domínio (DNS).

3. SISTEMA DE INTEGRAÇÃO GINGA SOCIAL

O objetivo principal do trabalho consiste na integração entre redes sociais e TV Digital contendo o Ginga. À solução proposta pelos autores para tanto deu-se o nome de **Sistema de Integração entre TV Digital Interativa e Facebook**. Dado o objetivo e uma análise do atual cenário, com ferramentas e tecnologias disponíveis, verificou-se que a aplicação se estenderia para além do ambiente da TV. Detalhes referentes à implementação, bem como justificativa das alternativas adotadas são expostas a seguir.

3.1 Arquitetura da Implementação

Para demonstrar de forma prática o sistema proposto, foi definido uma arquitetura englobando todos os itens desse sistema. A Figura 29 ilustra a arquitetura idealizada pelos autores:

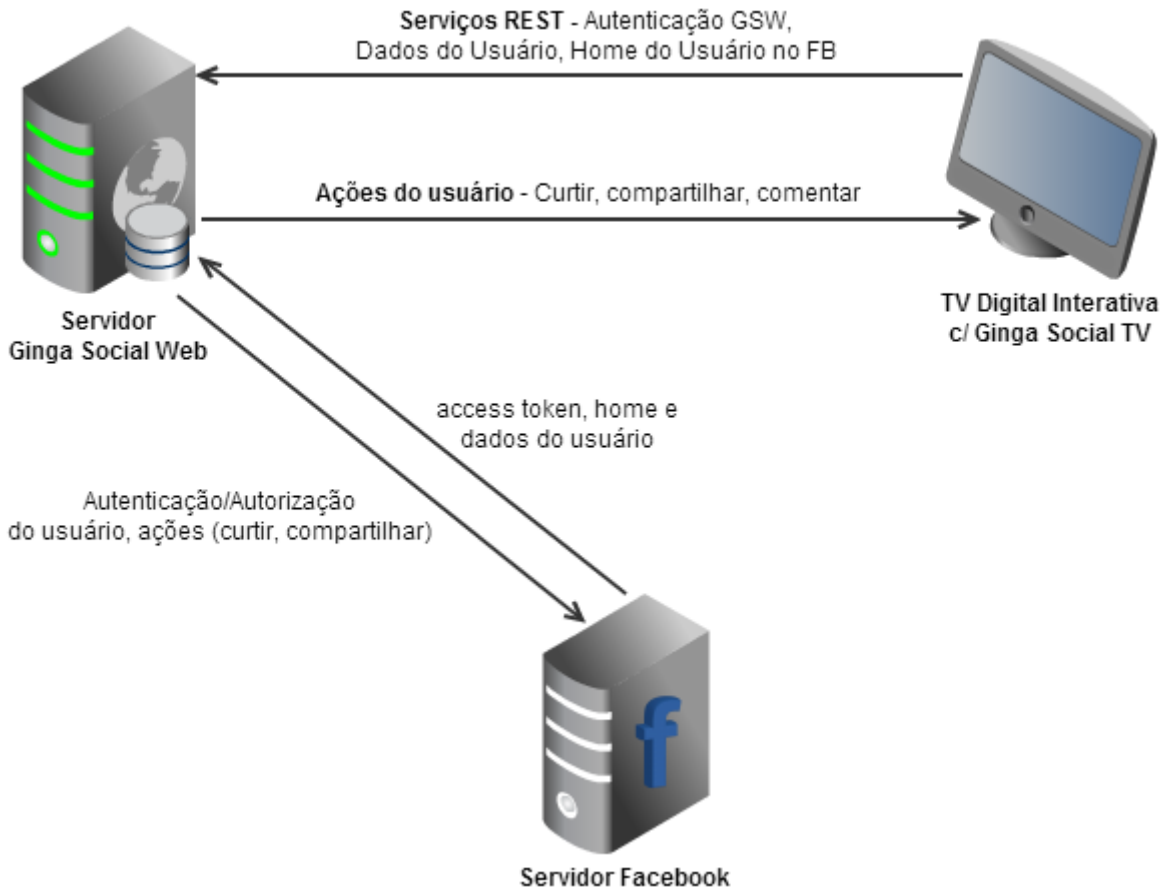


Figura 29 - Arquitetura do Sistema de Integração entre TV Digital Interativa e Facebook

Na figura pode ser observado que há um ponto central de conexão entre TV e Facebook, trata-se do servidor *web* do Ginga Social, intitulado como **Ginga Social Web (GSW)**. O servidor Ginga Social atua como um *gateway* entre a aplicação de TV, doravante chamada **Ginga Social TV (GSTV)**, e o **Facebook** comunicando-se bidirecionalmente com ambos através da Internet.

O tráfego de dados entre o **GSW** e o **Facebook** é realizado utilizando o protocolo HTTP com pacotes em formato **JSON** (Javascript Object Notation) por tratar-se de um padrão definido pelo próprio Facebook. Para a troca dos dados entre o **GSW** e

o **GSTV**, um formato foi escolhido pelos autores e possui explicação detalhada na seção 3.2.5.

O sistema desenvolvido possui os casos de uso presentes na Figura 30.

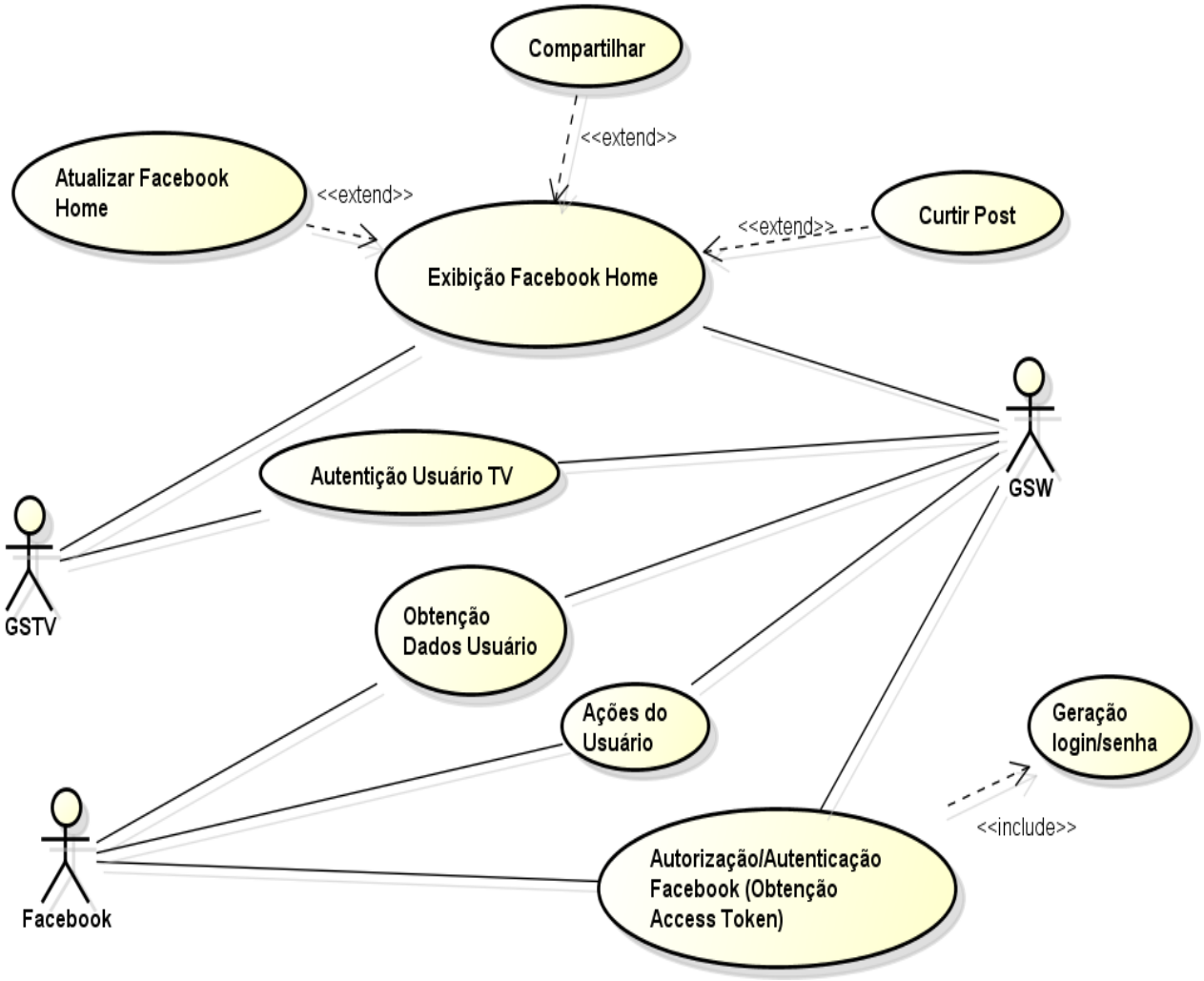


Figura 30 - Casos de uso do Ginga Social

Antes da execução da aplicação na TV o usuário deve permitir que seus dados, residentes no Facebook, sejam fornecidos à terceiros. O caso de uso Autorização/Autenticação Facebook contempla esta etapa, que é realizada através do Servidor GSW. Para tanto o usuário acessa a página do servidor “index.jsp” e autoriza explicitamente à aplicação Ginga Social obter seus dados junto ao Facebook. Após a

autorização, são gerados automaticamente login e senha a esse usuário de forma que os demais casos de uso possam ser utilizados na TV.

Ao executar a aplicação na TV, uma autenticação se faz necessária junto ao GSW que possui o *access token* do usuário persistido no Banco de Dados. Uma vez que o usuário autentique-se junto ao sistema, o Ginga Social Web obtém os dados do usuário junto ao Facebook para que seja possível a Exibição do Facebook *Home* do usuário. Com o *home* disponível em tela, pode-se curtir ou compartilhar um *post* e, a critério do usuário, atualizar seu *home* com os *posts* mais recentes ou mais antigos. Estas etapas podem ser observadas no diagrama de visão geral de interação, contido na Figura 31.

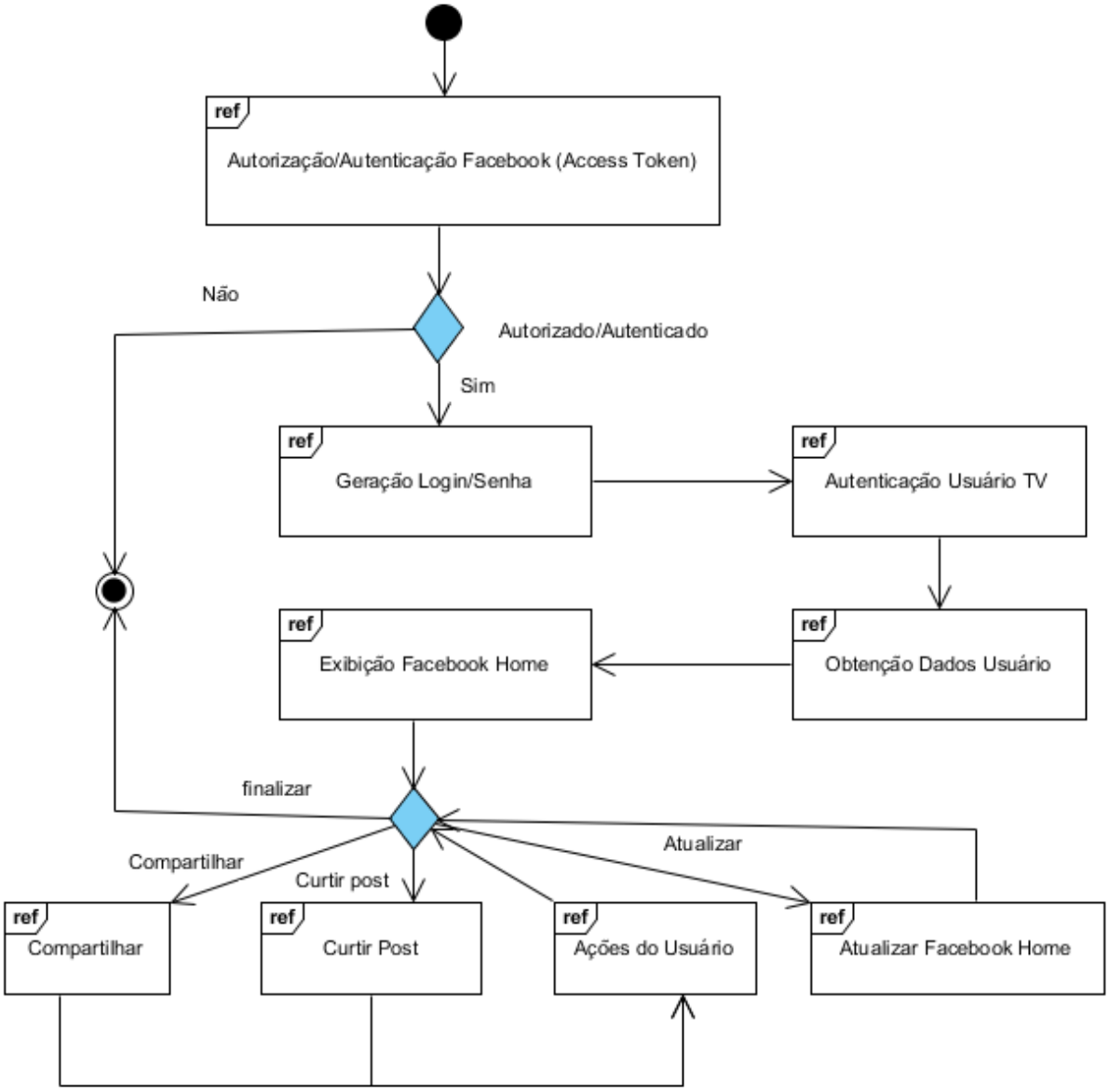


Figura 31 - Diagrama de visão geral de interação

3.2 Servidor Ginga Social Web (GSW)

3.2.1 Tecnologia

A lógica de negócio do sistema, o modelo de dados e sua persistência estão situados nesse servidor, justificando sua presença como elemento fundamental e

principal na arquitetura projetada. Ademais, um servidor intermediando a comunicação elimina uma dependência direta da aplicação com relação ao Facebook, bem como torna o ambiente mais amigável para possíveis extensões do conjunto de funcionalidades.

O GSW foi hospedado no *datacenter* da Amazon EC2 (AMAZON EC2, 2012) sob o domínio **www.gingasocial.tv**. Foi escolhida a Amazon porque possui a melhor relação custo-benefício dentre os centros de processamento de dados conhecido pelos autores.

Para funcionar como servidor *web* e SGBD foram instalados os *softwares* listados em sequência:

- Servidor Web JEE Tomcat, na versão 7.0.32 (APACHE SOFTWARE FOUNDATION, 2012)
- SGBD MySQL, *release* 5.5.28 (MySQL, 2012)
- Java SDK 7u9 (ORACLE, 2012)

Toda a lógica de negócio do sistema de integração está implementada na linguagem de programação Java. Com essa tecnologia o acesso ao banco de dados é efetuado através do JDBC *driver* do MySQL e utilizando o *framework Hibernate* (HIBERNATE, 2012) para mapeamento objeto-relacional (ORM). Com essa técnica adotada no sistema, as tabelas do Banco de Dados são representadas através de classes Java e seus registros como instâncias das classes correspondentes. Utilizou-se essa técnica para não haver preocupação com comandos em linguagem SQL, pois a interface de programação simples do *Hibernate* faz todo o trabalho de gerenciamento dos dados.

Para construção das páginas dinâmicas em *server-side*, a tecnologia JSP foi escolhida por ser de fácil manutenção, implementação e ser um ponto comum de

conhecimento entre os autores. Na construção das páginas *client-side* foi definido - por ser um padrão universal - utilizar HTML, CSS e JavaScript.

Partindo do princípio que o objetivo do trabalho não era construir um *layout* para a aplicação e sim possibilitar o acesso do usuário, foi decidido o uso do *Bootstrap*. De acordo com o próprio *website* de seus desenvolvedores (BOOTSTRAP, 2012), o *Bootstrap* é um poderoso *framework* de *front-end* para tornar o desenvolvimento *web* fácil e rápido.

No que se refere à tecnologia Javascript - responsável pela interação entre a aplicação e o usuário em seu navegador - foi acordado o uso da biblioteca Javascript *cross-browser* (que funciona na grande maioria dos navegadores atuais) jQuery atuando de forma rápida e concisa ao simplificar a manipulação de eventos, animação e interações AJAX no documento HTML.

3.2.2 Implementação

A estratégia de implementação do servidor Ginga Social Web foi inicialmente concebida ao gerar um projeto *web* dinâmico na IDE Eclipse para facilitar a criação da estrutura de pastas e sua programação. A figura 42 do Anexo B demonstra a árvore de diretórios contendo todos os arquivos necessários ao funcionamento do GSW e nas figuras 32, 33 e 34 seguem os diagramas de classes implementados.

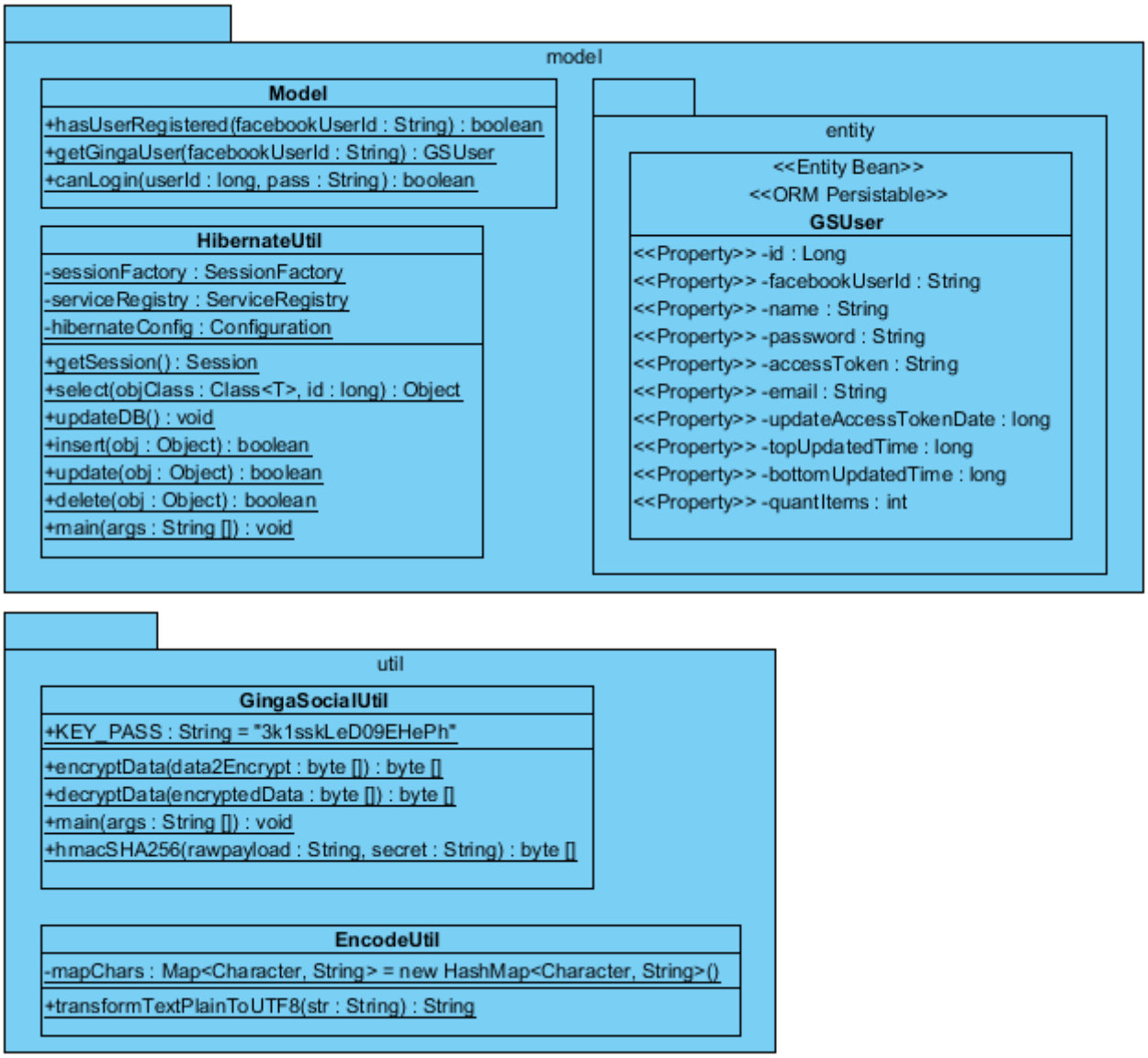


Figura 32 - Diagrama de classe dos pacotes tv.gingasocial.model e tv.gingasocial.util

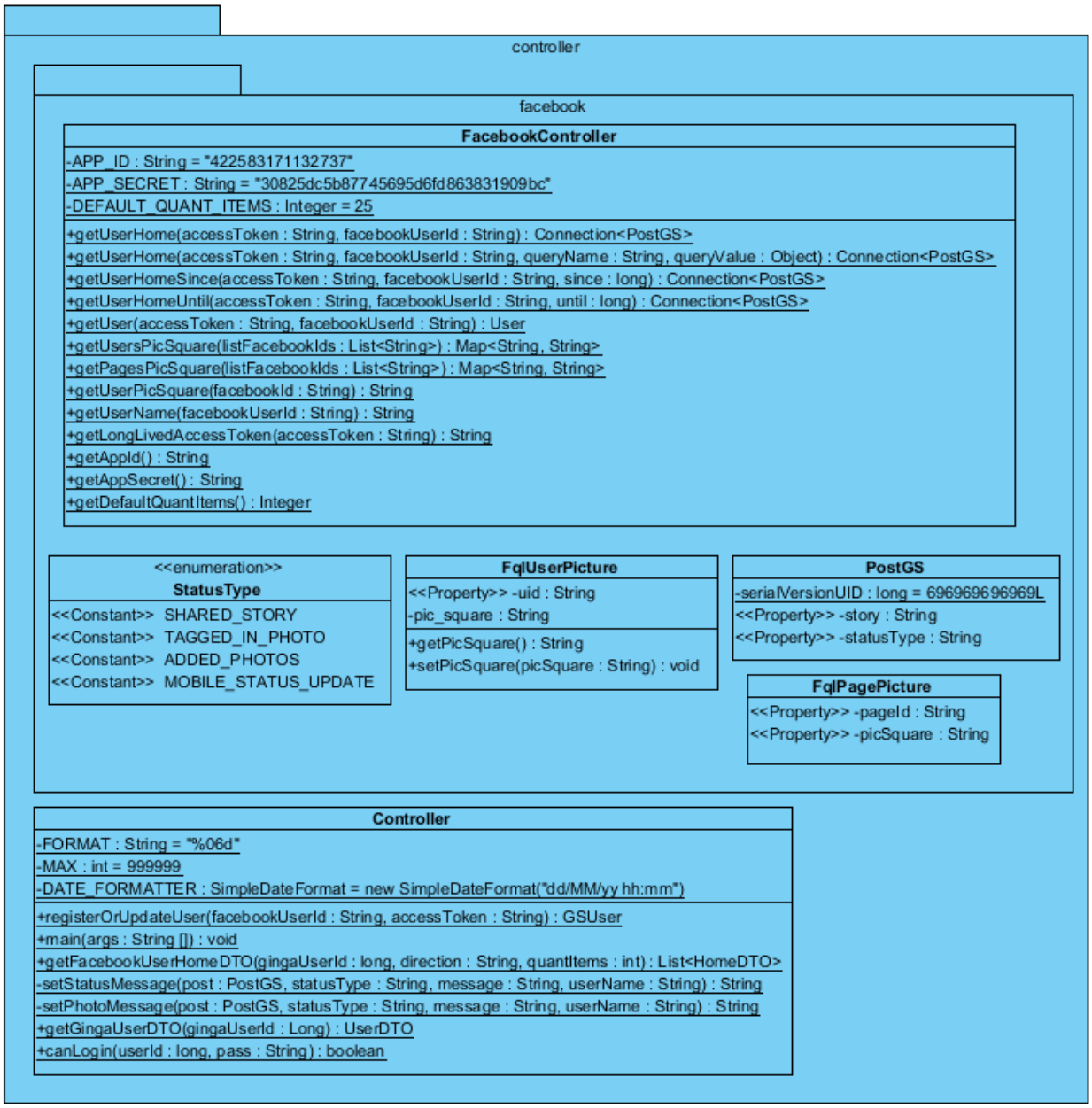


Figura 33 - Diagrama de classes referente ao pacote tv.gingasocial.controller

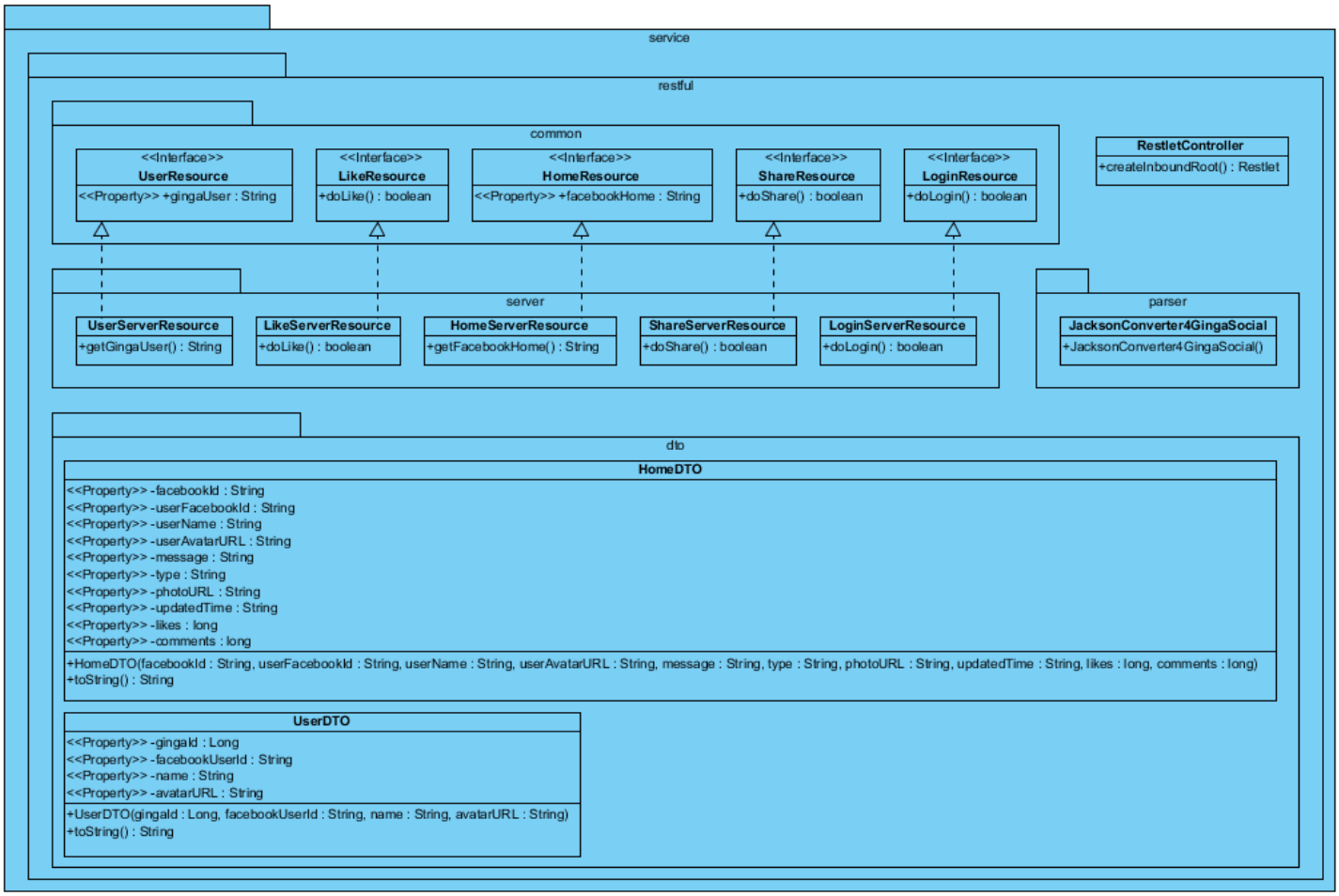


Figura 34 - Diagrama de classes relativo ao pacote tv.gingasocial.service

Em sua arquitetura inicial optou-se por adaptar o padrão de projeto de *software* conhecido como modelo-visão-controlador[14] juntamente com a arquitetura orientada a serviços[15]. Isso foi definido levando em consideração que o servidor GSW possui a camada de interface com o usuário (visão) representado pelas páginas HTML geradas

[14] *Model-View-Controller* - MVC: Padrão arquitetônico de software em que a implementação separa os dados do aplicativo (*model*), componentes gráficos (*view*) e controle de processamento de entrada (*Controller*).
[15] *Service-Oriented Architecture* - SOA: Padrão arquitetônico de software em que as funcionalidades de uma aplicação devem ser providas através de serviços.

dinamicamente (arquivos JSP), portanto, não há a necessidade dos elementos que compõem a visão estarem situados nos pacotes Java.

A partir da definição sobre qual conceito utilizar no desenvolvimento do sistema, o servidor foi inicialmente separado em 4 pacotes Java para contemplar o objetivo principal nesse elo de ligação entre TVDI e Rede Social:

- **tv.gingasocial.controller -** correspondente a toda implementação responsável pelo registro e atualização dos dados do usuário GS, bem como interface entre a arquitetura de serviços e o modelo da aplicação. Além dos controles supracitados, ele atua como ponto de acesso aos dados do usuário no servidor do Facebook.
- **tv.gingasocial.model -** corresponde ao modelo de dados implementado e classes utilitárias para acesso ao BD.
- **tv.gingasocial.service -** pacotes com implementação dos objetos de transferência de dados (*Data transfer objects* - DTO) e com classes que controlam o serviço aberto para o GSTV.
- **tv.gingasocial.util -** classes utilitárias para auxiliar e facilitar o desenvolvimento do restante do sistema.

Na continuação da arquitetura existe o hibernate.cfg que é, obviamente, o arquivo de configuração do ORM Hibernate e está descrito no anexo B. Em suma, ele contém informações sobre o driver do JDBC utilizado, o endereço (URL), usuário e senha de acesso ao SGDB e demais informações complementares necessárias para o funcionamento do framework.

Sendo este um sistema com fins acadêmicos, a senha de acesso ao MySQL não foi alterada, mantendo-a como o padrão “root”. Caso fosse uma aplicação voltada para uso comercial seria extremamente importante para a segurança dos dados dos usuários que fosse modificada para uma senha mais segura. Outra particularidade não realizada nesse trabalho é o uso de melhores práticas de programação no acesso ao BD, poderiam ser utilizadas bibliotecas adicionais em Java para o agrupamento (*pool*) de conexões com a intenção de aprimorar significativamente o desempenho e escalabidade da aplicação.

O modelo de dados concebido pelos autores é muito simples e prático. Existe apenas uma tabela (ou classe) para representar o usuário do sistema, intitulado aqui de GSUser. Nessa tabela estão todos os dados necessários para a autenticação e validação desse usuário, tanto para o Facebook como para o aplicativo instalado na televisão.

Dando continuidade no detalhamento estrutural do GSW, é apresentada a camada de visão contendo os elementos de interação direta com o usuário, descrito na árvore de diretórios como *WebContent*.

Dentro do diretório ilustrado na Figura 43 do anexo B estão os arquivos correspondentes ao *Bootstrap*, jQuery, imagens, páginas JSP e o arquivo de configuração do servidor Tomcat.

Além dos arquivos supracitados, existe um conjunto de bibliotecas Java auxiliando em importantes segmentos do contexto da aplicação. Esses componentes foram sendo definidos ao longo da etapa de convergência entre o Facebook e o aplicativo do Ginga Social TV. A Figura 38 mostra a listagem dos arquivos .jar.

Os 10 primeiros arquivos .jar são responsáveis pelo mapeamento objeto relacional proporcionado pelo Hibernate. Na sequência, o arquivo mysql-connector-

java-5.1.22-bin.jar refere-se à implementação da API JDBC para conexão entre a aplicação e o banco de dados.

Os arquivos localizados na parte final da Figura 44 do Anexo B, iniciados em "org.codehaus", são relacionados ao mapeamento das respostas advindas do Facebook em formato JSON (padrão utilizado pela rede social).

Na necessidade de expor serviços a diferentes aplicações e considerando que a comunidade de desenvolvedores *web* e grandes empresas ligadas a TI estão migrando seus serviços para a arquitetura REST, tais como Google e Facebook, optou-se pela utilização dessa arquitetura em nosso trabalho. Para otimizar essa arquitetura e pela facilidade de implementação, o sistema conta com a Restlet API. Este encontra-se representado na figura 38 pelos arquivos iniciados com "org.restlet".

A configuração de todos os serviços disponíveis no GSW e gerenciadas pelo Restlet está ilustrada na Figura 45.

Para finalizar optou-se pelo uso da biblioteca Java RestFB, mais precisamente empacotada em restfb-1.6.11.jar que faz toda a intercomunicação com a API do Facebook. Ela é responsável por realizar o mapeamento das respostas provenientes do Facebook em objetos Java reduzindo consideravelmente o esforço de implementação.

3.2.3 Autenticação/Autorização Facebook - Obtenção do *Access Token*

Podemos considerar esse caso de uso como sendo o ponto de entrada do usuário que deseja utilizar o Ginga Social em sua TV. É através dele que o Ginga Social Web consegue autorização para acessar dados do usuário no Facebook e por sua vez criar automaticamente um cadastro para que posteriormente possa acessar a aplicação na TV.

Para tal propósito, de acordo com os termos e políticas do Facebook, foi criado um aplicativo no Facebook Developer (FACEBOOK, 2012f), Figura 35, para acessar informações de seus usuários.

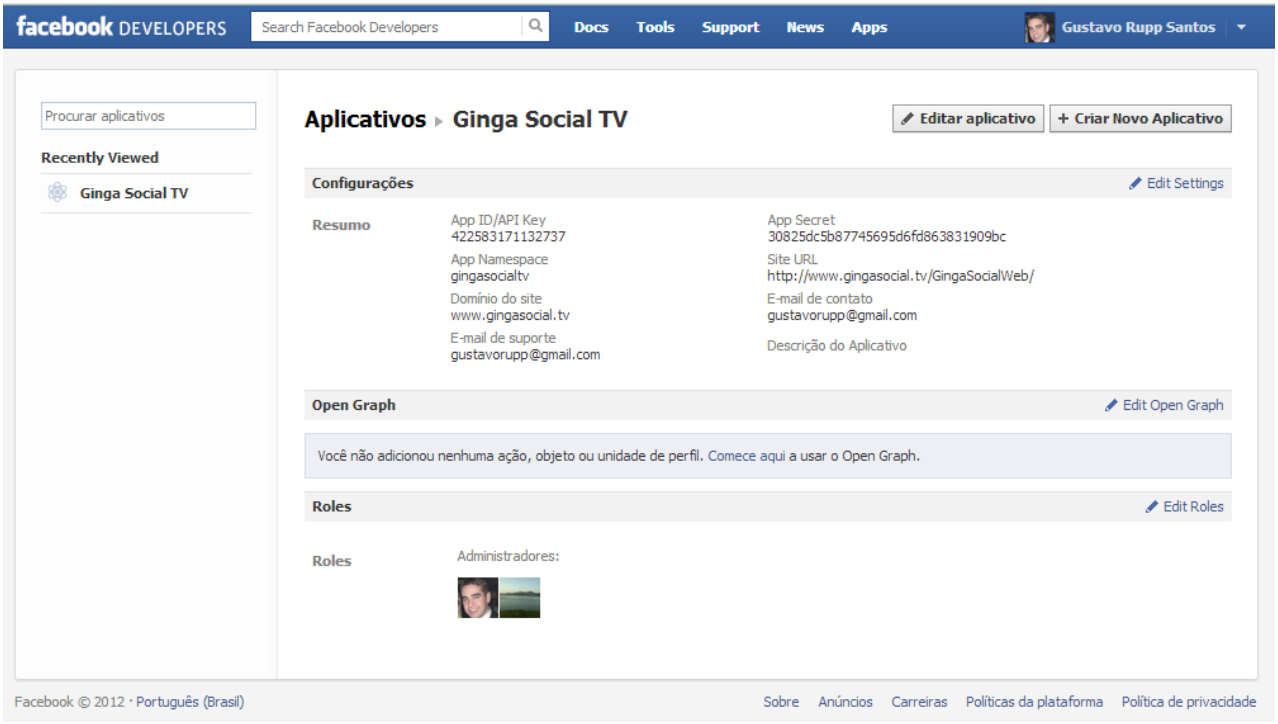


Figura 35 - Facebook developer com a aplicação Ginga Social registrada

Todo esse processo tem início quando um usuário acessa pela primeira vez o endereço www.gingasocial.tv/GingaSocialWeb/index.jsp. A tela exibida em seu navegador (figura 36) solicita ao usuário para usar sua conta do Facebook para conectar-se (cadastrar-se automaticamente) no Ginga Social.

Figura 36 - Página para criação de conta no Ginga Social a partir da conta existente no Facebook.

Novamente aqui apresenta-se mais uma justificativa do motivo pelo qual optou-se incluir um servidor intermediando a comunicação Facebook e TVDI. Até o presente momento, de acordo com a documentação do Facebook, o usuário apenas pode liberar acesso de seus dados contidos na rede social a outro aplicativo através de um navegador *web* ou de forma nativa em celulares que possuam sistema operacional Android ou iOS.

Após o clique realizado pelo indivíduo no botão azul, aparecerá a tela padrão do Facebook solicitando permissão de acesso aos seus dados em nome do aplicativo previamente registrado conforme Figura 37.

Figura 37 - Tela de solicitação dos dados do usuário (fazer *login* com o Facebook)

Considerando o clique em fazer *login*, a página index.jsp recebe um *callback* da API em Javascript do FB com o *access token* (nada mais que um simples texto plano criptografado). Nesse *token* estão encapsuladas as informações da aplicação, quais permissões o usuário concedeu e o tempo de expiração dessa chave de acesso. Logo em seguida, essa mesma página chama - usando a tecnologia AJAX - o login.jsp passando o código recebido e outras informações como nome, email e seu identificador no Facebook (*facebook user id*).

De acordo com a documentação, esse código inicial contém de uma a duas horas de validade. No entanto, por tratar-se de um processo tedioso de acessar o navegador, atualizar código de acesso aos dados e posterior *login* ao aplicativo na TV, foi utilizada mais uma chamada à API para obter nova chave. Esse código é conhecido por *long-lived access token* e expira em 60 dias.

Ao receber os dados necessários, a página login.jsp chama a classe tv.gingasocial.controller.Controller para que realize o cadastro ou atualize os dados do atual usuário. A atualização dos dados ocorre quando o sistema identifica que essa pessoa já permitiu acesso e possui conta criada no Ginga Social.

Na classe Controller existe a verificação se o usuário já está cadastro no sistema chamando métodos da classe tv.gingasocial.model.Model para validar tal informação. Caso não exista, um usuário é gerado a partir dos dados oriundos da rede social e é gerada automaticamente uma senha aleatória contendo 6 números. Novamente, sendo este um trabalho com propósito acadêmico, optamos por utilizar apenas números na senha do usuário e não lhe possibilitando alterá-la. Caso fosse para fins comercias, obviamente que isso deveria ser modificado para fornecer uma melhor experiência a quem acessa o sistema.

Seguindo com o processo, o servidor (login.jsp) responde ao chamador (index.jsp) passando seus dados. Os dados são exibidos na tela para que o usuário possa posteriormente utilizá-los na aplicação da TV finalizando assim esse caso de uso.

3.2.4 Autenticação do usuário na TV (*server-side)*

Como toda a lógica de negócio da sistema de integração está localizado no GSW, este acaba gerenciando todos os *logins* e senhas dos usuários. Sendo assim, o servidor do Ginga Social incorpora mais uma importante função, atuando como o *web service* responsável pela integração e comunicação das diferentes aplicações.

Então, com a necessidade de expor um serviço voltado à autenticação do usuário, conforme citado na etapa de implementação, optou-se pelo uso da API *Restlet* a fim de facilitar o desenvolvimento dos serviços.

Um padrão é seguido na nomenclatura dos pontos de acesso do *web service*, iniciando com o domínio www.gingasocial.tv/, concatenado com o termo "*facebook/*" e por fim a ação desejada, no caso de uso atual, *login*. Essa URL é submetida ao GSW sem passar os dados por nenhum algoritmo de criptografia, tratando-se este de um trabalho acadêmico, utilizando o método GET e passando os parâmetros *login* e *pass*, representando o identificador do usuário no sistema e sua senha respectivamentes:

*http://www.gingasocial.tv/GingaSocialTV/facebook/login?login=**ID**&pass=**SENHA***

Dessa forma, alinhando-se à proposta desse trabalho, o serviço de autenticação sendo totalmente modular abre possibilidade de criação de novos aplicativos que possam estender a plataforma de convergência Ginga Social.

Seguindo com a implementação e o fluxo de execução, assim que o serviço de autenticação for chamado pelo Ginga Social TV, o gerenciador dos serviços irá passar o tratamento para a classe LoginServerResource. Essa está encarregada de fazer o tratamento dos parâmetros e retornar um erro caso estejam nulos ou vazios. Quando forem válidos, eles são repassados à classe Controller que, por sua vez, solicita à Model.java que efetue a verificação no banco de dados. O modelo (classe Model) verifica se existe usuário registrado com aquele identificador (*login*) na base de dados e se a senha é válida retornando ao controlador o resultado.

Na continuidade desse processo, a classe Controller.java envia um valor booleano para LoginServerResource e este o propagará para a TV.

3.2.5 Obtenção dos Dados do Usuário

Este é um caso de uso que contempla dois serviços abertos pelo GSW para obter diferentes dados do usuário a serem consumidos pelo GSTV. Sem sombra de dúvidas este caso de uso, juntamente com a exibição do Facebook *home*, são os elementos centrais de toda a sistema de convergência, já que concentram todos os grandes esforços dos autores tanto no Ginga Social Web quanto no Ginga Social TV.

O primeiro serviço é acessado através de uma requisição *HTTP* no endereço http://www.gingasocial.tv/GingaSocialWeb/facebook/user?ginga_user_id=ID, sendo que o único parâmetro, ginga_user_id, deve representar um usuário válido no Ginga Social.

Assim que uma requisição chegar nessa URL, o gerenciador dos serviços Rest - a classe RestletController - repassará o fluxo de execução para UserServerResource, que fará uma checagem inicial dos parâmetros. Caso o ginga_user_id seja nulo ou diferente de um número inteiro, o *web service* retornará um erro 400 para indicar uma *bad request*. Estando correto, o UserServerResource chamará o controlador solicitando os dados do usuário de acordo com seu identificador.

A classe Controller solicita ao Model a entidade GSUser relacionada a esse identificador. A partir dela, é pego o FacebookUserId e assim o GSW comunica-se com o Facebook solicitando dados mais recentes e, assim que os obtém, os registra na base de dados. Isso é realizado para que os dados do usuário estejam sempre atualizados e não haja inconsistências.

O segundo serviço está disponível na URL abaixo:

http://www.gingasocial.tv/facebook/home?ginga_user_id=**ID**&direction=**DIRECTION**&quant_items=**ITEMS**

A respeito dos parâmetros citados, temos:

- **ginga_user_id** - parâmetro obrigatório, deverá ser um número inteiro e corresponde ao usuário cujo *home* do Facebook deverá ser encaminhado de volta
- **direction** - parâmetro opcional, sendo que no momento em que é realizado o *login*, o Ginga Social TV não enviará esse parâmetro preenchido. Ele somente será usado quando o usuário efetuar uma ação na televisão que necessite da atualização de seu *home*. Dessa forma ele possui dois valores válidos, *top* e *bottom*, este representando os *posts* com data de modificação mais antiga e aquele com data de modificação mais recente.
- **quant_items** - parâmetro opcional, devendo ser um número inteiro entre 1 e 25. Corresponde à quantidade de *posts* solicitada pelo GSTV. Por padrão é adotada a quantidade de 25 itens.

Quando uma requisição *http* chega nessa URL, o fluxo de execução segue de maneira semelhante ao serviço anterior, exceto que a classe RestletController repassará o fluxo de execução para HomeServerResource verificar os parâmetros recebidos. Caso estes estejam corretos, a classe chama o controlador solicitando o *home*.

A classe Controller analisa os parâmetros e, de acordo com seus valores, chamará diferentes métodos da FacebookController, que por sua vez solicita o *home* à API do Facebook passando o *access token* que está associado ao usuário, localizado no banco de dados. Para esses dois serviços descritos, os autores definiram um formato padrão de resposta (protocolo) para o requisitante. O protocolo possui a estrutura sintática demonstrada na Figura 38. A mensagem de resposta inicia e termina com o

símbolo “|” e a informação relevante encontra-se entre apóstrofos e são separados por vírgula.

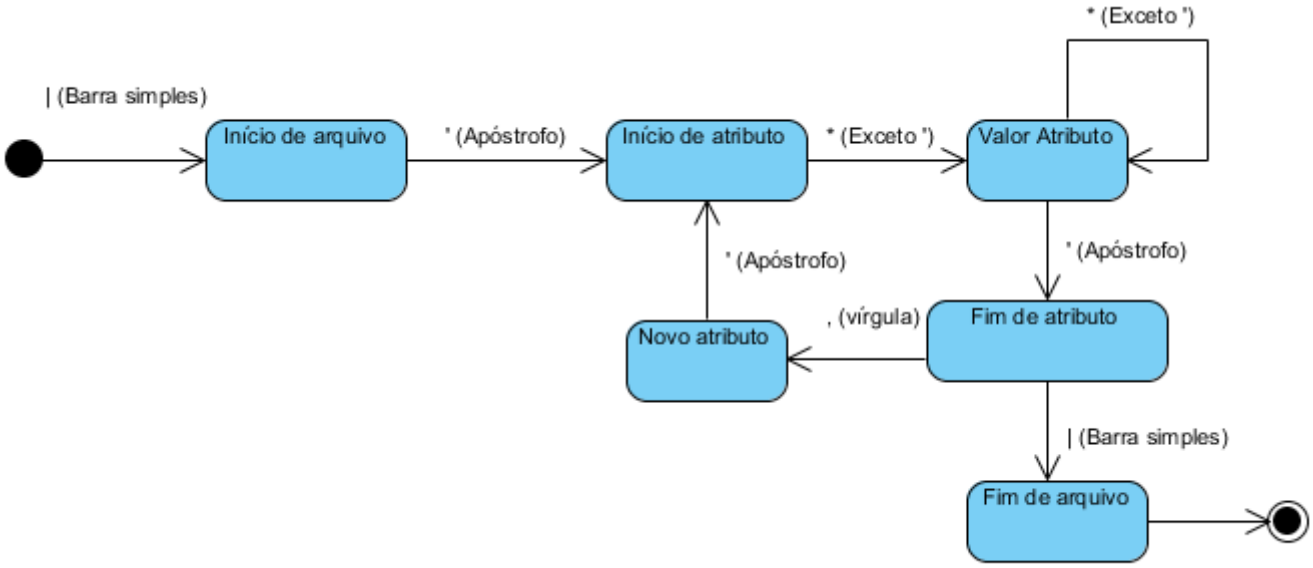


Figura 38 - Sintaxe do protocolo padrão de comunicação

Optou-se por desenvolver este padrão porque os recursos no ambiente de TV são limitados. Sendo assim, um protocolo simples e enxuto é o mais recomendado por não demandar bibliotecas sofisticadas e complexas para fazer sua análise e transformação dos dados.

3.2.6 Ações do Usuário

Para esse trabalho foram implementadas as ações do usuário no Facebook que correspondem à “curtir um *post*” ou “compartilhar um *post*”, as quais possuem diferentes serviços Rest acessíveis através de uma requisição *HTTP*.

A primeira ação, curtir um *post*, pode ser efetuada no endereço virtual http://www.gingasocial.tv/GingaSocialWeb/facebook/like?ginga_user_id=ID&facebook

_id=FBID e a segunda ação, compartilhar um *post* obtém-se substituindo-se o termo *like* por *share*:

http://www.gingasocial.tv/GingaSocialWeb/facebook/share?ginga_user_id=ID&facebook_id=FBID

Como anteriormente citado, o ginga_user_id está relacionado com o usuário cadastrado no banco de dados do GSW. O parâmetro facebook_id é o identificador do evento ocorrido[16] no Facebook o qual receberá a ação desejada (curtir ou compartilhar).

Assim como todos os outros serviços descritos, tudo passa inicialmente pela RestletController que, de acordo com a URL acessada, repassará o controle para LikeServerResource ou ShareServerResource. Essas classes analisam os dados recebidos e, se corretos, chamam o controlador do sistema para seguir o fluxo de execução. Este, por sua vez, chama o FacebookController para que ele registre a referida ação através da API do Facebook.

Ao final desse processo, o fluxo de execução retorna para o cliente um valor *boolean* correspondente ao desfecho da operação.

3.3 Ginga Social TV

Da parte da televisão foi desenvolvida uma aplicação em consonância com o padrão brasileiro de TV digital, mais especificamente com a máquina de execução Ginga-J. Como já mencionado, os autores possuem um conhecimento sólido da linguagem Java, sendo este o fator preponderante que motivou o uso da máquina de execução para o trabalho. Ademais, contribuiu para a escolha deste ambiente o fato de a linguagem Java possuir uma grande comunidade e estar consolidada no mercado. Esta

[16] Neste trabalho entende-se por evento uma mensagem, foto, vídeo, *link*, *check-in* e pergunta adicionados por um usuário

aplicação poderia ser difundida juntamente com o áudio e vídeo (um programa televisivo) da emissora e, opcionalmente, ser executada pelo telespectador. Um ambiente de TV Digital é caracterizado pela limitação dos recursos, como por exemplo, a capacidade de processamento, sendo estes muitas vezes disputados por outras aplicações. Tendo esta premissa em mente, optou-se por desenvolver uma aplicação com componentes visuais leves com o intuito de não consumir processamento em demasia.

3.3.1 Tecnologias

Para a implementação do Ginga Social TV foi utilizado como ambiente de desenvolvimento uma máquina virtual denominada Astro Box (TOTVS, 2012a). Esta plataforma está disponível a todos os membros da rede Astro DevNet (TOTVS, 2012b). A rede Astro DevNet foi planejada pela TOTVS e, de acordo com a mesma, visa "*oferecer a empresas pioneiras a oportunidade de se capacitarem rapidamente no desenvolvimento de aplicações interativas para a TV Digital*". Existem duas modalidades de participação na rede: Modalidade Desenvolvedores de Stickers/Ginga e Modalidade Corporativa. A modalidade de desenvolvedores é gratuita bastando apenas o usuário preencher um cadastro e aceitar os termos de uso.

O AstroBox é um emulador Ginga DTV em plataforma Linux para ser executado através de um PC. No AstroBox é possível executar conteúdos interativos voltados para TV Digital compatível com a especificação do *middleware* Ginga DTV. Isso significa que ele pode executar:

- Aplicações Ginga-J (incluindo suporte completo ao Java TV e especificações Java DTV)

- Aplicações Ginga-NCL (incluindo suporte a scripts Lua e conteúdo XHTML)

Além disso, AstroBox pode ser usado para desenvolver Stickers, que são aplicações Ginga, e podem ser incorporados em qualquer receptor que dê suporte ao StickerCenter, a solução de Broadband e Broadcast da TOTVS. AstroBox é distribuído com Linux Ubuntu 10.04 e é executado através de uma máquina virtual, Virtual Box, a fim de facilitar a sua instalação em qualquer ambiente de desenvolvimento. Dentro do AstroBox existe o AstroTV, que é a implementação TOTVS do middleware brasileiro Ginga e, de acordo com a mesma, é 100% compatível com a especificação brasileira, em um formato comercial e otimizado.

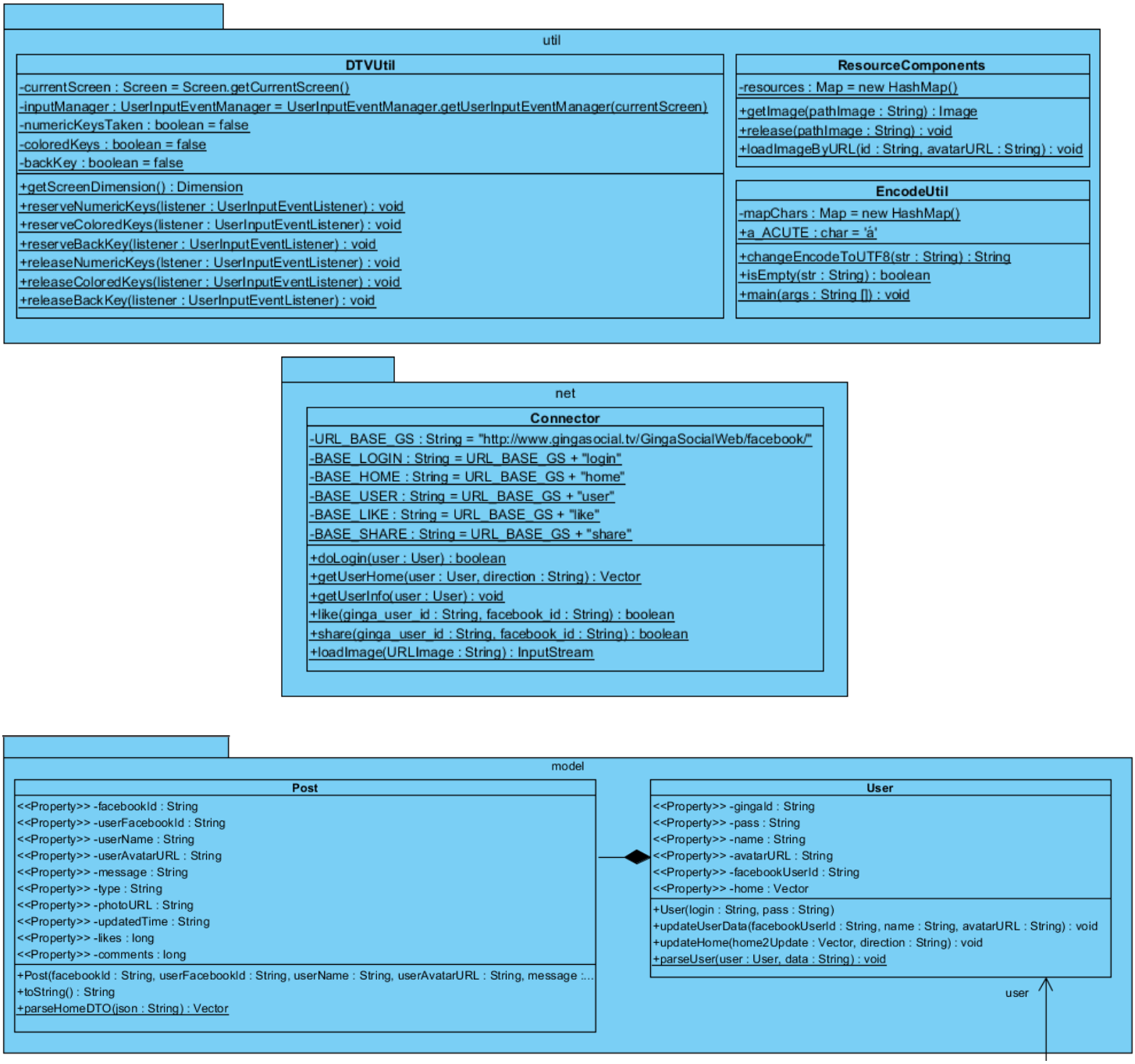
Durante o ciclo de vida de desenvolvimento de aplicações, o AstroBox facilita as tarefas de desenvolvimento, permitindo a execução de aplicações de TV digital na própria estação de trabalho, evitando a necessidade de implantá-lo em um laboratório completo (sistema de *playout* e *set-top box*).

AstroBox fornece uma maneira simples e direta para desenvolver e construir aplicações de TV digital, permitindo a execução da maioria de suas tarefas em um ambiente de emulação Ginga na estação de trabalho do desenvolvedor.

3.3.2 Implementação

Como dito anteriormente, o Ginga Social TV é uma aplicação Ginga-J, sendo assim implementada na linguagem Java. Toda a aplicação foi desenvolvida utilizando os pacotes disponíveis no próprio Ginga-J, ou seja, nenhuma biblioteca externa foi incluída visando sempre a eficiência na execução da aplicação.

Basicamente, numa visão de alto nível, a aplicação está organizada da conforme ilustrado na Figura 39:



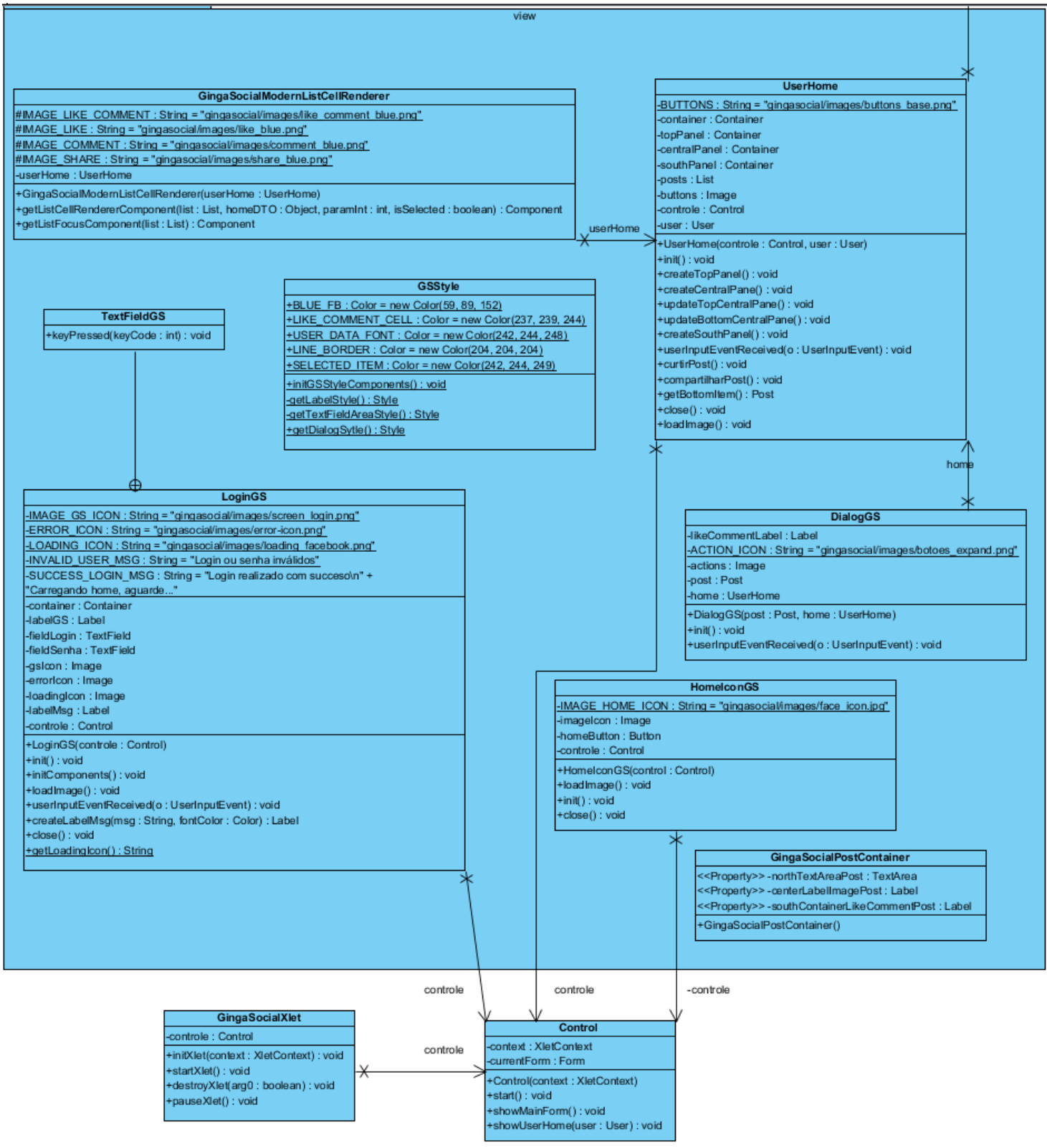


Figura 39 - Diagrama de classes do GSTV

A aplicação como um todo encontra-se dentro do pacote **gingasocial** que não é possível visualizar por questões de espaço e, este por sua vez, contém os seguintes componentes:

- **GingaSocialXlet:** Classe que implementa a interface Xlet. Como vimos anteriormente, toda aplicação Ginga-J deve implementar a interface Xlet, pois através de seus métodos o ciclo de vida da aplicação pode ser gerenciado. A aplicação em questão simplesmente obtém uma referência do contexto da aplicação no método *initXlet()* e instancia um objeto da classe Control.
- **Control:** Classe responsável por gerenciar as interfaces gráficas da aplicação de acordo com a interação do usuário.
- **gingasocial.images:** Pacote contendo as imagens estáticas da aplicação, como por exemplo, a imagem correspondente ao ícone inicial da aplicação. Através de um ícone inicial os usuários tomam conhecimento da existência de uma aplicação disponível para ser executada.
- **gingasocial.model:** Pacote contendo as entidades *User* e *Post*, que representam respectivamente o usuário e os *posts* que compõem o Facebook Home do mesmo.
- **gingasocial.net:** Pacote responsável pela conexão da aplicação com o Ginga Social Web. Através da classe *Connector* todos os dados do Facebook necessários à aplicação são obtidos, bem como dados da aplicação também são enviados.
- **gingasocial.util:** Pacote com classes utilitárias que proveem informações do ambiente, adiciona tratadores de eventos para teclas do controle remoto e gerência de imagens.
- **gingasocial.view:** Pacote contendo as interfaces gráficas da aplicação.

3.3.3 Autenticação do Usuário na TV

Ao iniciar a aplicação é solicitado ao usuário que este informe seu *login* e senha gerados pelo Ginga Social Web no momento da autorização da aplicação. Estes dados são submetidos ao servidor através de uma requisição *HTTP* correspondente ao serviço nele presente. A chamada é feita através das classes presentes em um dos pacotes obrigatórios na plataforma Java do Ginga-J, o pacote *java.net* (ABTN 15606-4). A classe gingasocial.net.Connector encapsula um objeto da classe java.net.URL e o instancia com o endereço correspondente ao serviço de autenticação do usuário. Em seguida, uma conexão é aberta com o GSW através da classe java.net.URLConnection. Com a requisição efetuada, o servidor tem a posse dos dados e verifica se estes correspondem a um usuário e senha válidos e como resposta à conexão aberta pelo Ginga Social TV retorna *true* ou *false.* Cabe ressaltar, novamente, de que sendo esta uma aplicação com fins acadêmicos, não houve a preocupação em se realizar um transporte dos dados de forma segura, como por exemplo, aplicando algoritmos de criptografia nos dados.

A inserção dos dados de *login* e senha foi projetada para ser efetuada através das teclas númericas do controle remoto. Nesta etapa também são utilizadas as teclas coloridas do controle remoto, cada uma desencadeando uma ação de acordo com a Figura 40.

Figura 40 - Autenticação no Ginga Social TV

No ambiente simulado pelo AstroBox as teclas de controle remoto verde, amarela e vermelha são representadas respectivamente pelas teclas F2, F3 e F1.

A interface gráfica desta etapa da aplicação basicamente foi desenvolvida com um Label e dois TextFieldGS, componentes que estendem TextField e sua implementação é feita em LoginGS.java. O TextFieldGS foi criado para que a tecla *BACK* do controle remoto possuísse a mesma funcionalidade da tecla *backspace* dos teclados convencionais. A classe LoginGS é uma manipuladora de eventos do usuário, para tanto, implementa a interface com.sun.dtv.ui.event.UserInputEventListener. Logo após ser instanciada por um objeto da classe Controle, esta classe providencia a reserva das teclas coloridas e numéricas bem como sinaliza ao gerenciador de eventos que ela passará a manipular os eventos das teclas reservadas. Depois da autenticação do usuário, a aplicação procede de forma transparente para a obtenção dos demais dados necessários para a realização da próxima etapa, a exibição do Facebook Home. Estes

dados correspondem aos atributos da classe User e são obtidos em mais duas etapas, lembrando que login e senha do Ginga Social TV já estão disponíveis.

A primeira etapa ocorre através de uma outra requisição *HTTP* para o serviço de obtenção dos dados do usuário disponível no Ginga Social Web e que faz uso do padrão de comunicação desenvolvido pelos autores e já descrito no caso de uso Obtenção Dados Usuário. A comunicação entre o Ginga Social TV e o Ginga Social *Web*, ocorre de forma semelhante àquela descrita anteriormente, com a diferença de que este serviço está disponível em outro endereço. A execução deste serviço retorna os demais atributos do usuário com exceção do *home*, que será obtido em seguida. Nesta etapa, então, tem-se disponíveis as seguintes informações do usuário (objeto User):

- gingaId: identificação do usuário dentro do contexto Ginga Social;
- pass: senha do usuário dentro do contexto Ginga Social;
- name: nome do usuário dentro do contexto Facebook;
- avatarURL: endereço correspondente ao avatar do usuário no Facebook;
- facebookUserId: identificação do usuário dentro do contexto do Facebook.

O atributo *home*, na verdade, trata-se de um vetor de objetos Post que, por sua vez, representam *posts* presentes no Facebook Home do usuário. Por ser um atributo mais complexo, optou-se por obtê-lo separadamente através de um novo serviço, como descrito anteriormente. Um objeto Post possui os seguintes atributos:

- facebookId: identificação do evento no contexto do Facebook.
- userFacebookId: identificação do usuário que gerou o evento no Facebook;
- userName: nome do usuário, dentro do contexto do Facebook, que gerou o evento;

- userAvatarURL: endereço do avatar do usuário que gerou o evento;
- message: o texto associado ao evento (este atributo pode ser nulo);
- type: o tipo do evento gerado, podendo ser uma plublicação de foto, comentário, compartilhamento, etc.
- photoURL: foto associada ao evento (este atributo pode ser nulo)
- updatedTime: o momento em que o evento foi gerado;
- likes: a quantidade de *likes* atribuídos ao evento;
- comments: a quantidade de comentários relativos ao evento.

Como, para este projeto, os desenvolvedores optaram por definir um protocolo próprio de comunicação entre o servidor e o cliente, diferente das tradicionais, como XML ou JSON, foi necessário desenvolver um *parser* para que os dados fossem extraídos da resposta gerada a cada solicitação. Por se tratar de um formato simples, o *parser* criado não demandou muita complexidade, iterando sobre a sequência de caracteres e extraindo o necessário.

Após estas etapas a aplicação libera os recursos por ela consumidos, imagens e teclas coloridas por exemplo, para que a próxima etapa possa ser iniciada.

3.3.4 Exibição do Facebook Home

De posse de todos os dados necessários para a exibição do *Facebook Home*, a interface gráfica responsável, UserHome, é instanciada pelo objeto da classe Control. Basicamente esta interface é composta por *avatares*, fotos e textos. O Ginga Social TV, na verdade, possui nesta etapa somente a URL correspondente a cada imagem, que são

descarregadas através do método estático *loadImageByURL()* da classe ResourceComponents. Este método delega parte da tarefa à classe Connector e esta, através do uso de um objeto da classe java.net.URL, abre uma conexão com o servidor indicado na URL da imagem e recebe um *inputstream* que será utilizado para criar a imagem através do método estático *createImage()* da classe *com.sun.dtv.lwuit.Image*. Uma vez criada a imagem, esta é armazenada em um objeto java.util.HashMap, que é um atributo estático da classe ResourceComponents. As imagens são recuperadas através do método estático *getImage()*, também de ResourceComponents. As mensagens são armazenadas diretamente no atributo message da classe Post e já estão disponíveis.

O Facebook Home será apresentado ao usuário de acordo com a Figura 41:

Figura 41 - Facebook Home no GSTV

No topo da interface são apresentados o avatar do usuário autenticado e o seu respectivo nome. Utilizou-se para tanto objetos da classe Label contendo a informação, o que tornou a implementação simples.

O componente central, em termos de disposição e funcionalidade, desta interface gráfica é um objeto da classe com.sun.dtv.lwuit.List. Assume um papel importante, também, o atributo User pois, através dele, serão obtidos todos os dados necessários. Cada item de uma List corresponde a um objeto incluído explicitamente. Por padrão, o renderizador de uma List é o objeto da classe com.sun.dtv.lwuit.list.DefaultListCellRenderer. Para a exibição do Facebook Home do usuário este renderizador não seria adequado, pois ele simplesmente chama o método toString() do objeto sendo adicionado para desenhar cada um dos itens da lista.

A alternativa adotada foi, através da classe GingaSocialModernListCellRenderer, implementar a interface com.sun.dtv.lwuit.list.ListCellRenderer, que define os métodos utilizados pela List para que a mesma possa ser apresentada. Este novo renderizador desenha cada item da List separadamente através do método getListCellRendererComponent() que, dentre os seus parâmetros possui um Object que irá corresponder ao elemento chave para tanto. No escopo do Ginga Social TV é fácil perceber que é imprescindível para a exibição do home o conjunto de posts, sendo assim, o método getListCellRendererComponent() na verdade recebe um Post, e a conversão (*casting*) é feita no corpo do método.

Na imagem podemos observar que, a princípio, apenas o emissor do post, o texto e a quantidade de *likes* e *comments* são apresentados. A List sempre possuirá um item selecionado (com foco), que pode ser expandido através de um objeto DialogGS que

estende com.sun.dtv.lwuit.Dialog, onde eventuais imagens e textos vinculados ao post serão apresentados de forma mais legível.

Para a instanciação e construção da lista não há a necessidade de o Ginga Social TV se comunicar com o Ginga Social Web, visto que todos os dados necessários já estão disponíveis, seja através do objeto User ou através do ResourceComponents (imagens).

Na parte inferior da imagem podemos notar o tipo de ação que os eventos provenientes das teclas coloridas podem gerar. Todos estes eventos inicialmente são tratados no método userInputEventReceived(), que é o único método da interface com.sun.dtv.ui.event.UserInputEventListener. No corpo do método é verificada a origem do evento e as providências necessárias para a realização da ação solicitada são efetuadas. Esta abordagem ocorre também na interface gráfica LoginGS.

3.3.5 Curtir, Compartilhar e Atualizar Facebook Home

Todas estas opções apresentadas ao usuário juntamente com o seu Facebook Home são acionadas através das teclas coloridas do controle remoto. Necessariamente, a execução de uma delas implica numa interação com o GSW (caso de uso Ações do Usuário ou Obtenção dados Usuário) por parte do servidor.

Toda a comunicação entre GSTV e GSW apresentada até agora ocorre através de requisições *HTTP* para a URL onde o serviço está sendo disponibilizado. Nestes casos de uso ocorre situação semelhante, ou seja, faz-se uso dos objetos java.net.URL e java.net.URLConnection, atributos da classe gingasocial.net.Connector, para obter acesso aos serviços correspondentes.

Para curtir um post basta o usuário selecionar o mesmo e acionar a tecla colorida. Em seguida, através do método *getSelectedItem()* da List o post a ser curtido é recuperado. Como vimos anteriormente, este objeto encapsula uma série de informações, porém, para o GSW é necessário informar apenas o gingaId e o facebookId para repassá-los ao serviço adequado. Os atributos são usados como argumentos na chamada ao método *like()* da classe Connector que acionará o GSW. Com a confirmação do sucesso da operação advinda do servidor, o Facebook Home é atualizado no GSTV.

De maneira semelhante ao processo de curtir um post, o compartilhamento necessita saber qual post o usuário deseja compartilhar. Os atributos selecionados são os mesmos do *like*, entretanto, estes são submetidos ao método *share* da classe Connector que delega a ação para um outro serviço do GSW.

Por último, a opção de atualizar o Facebook *Home* faz uso de um serviço já descrito, no caso de uso Obter Dados Usuário, porém, desta vez, é utilizado o parâmetro opcional *direction* do serviço. Esta funcionalidade comporta-se de maneira similar ao Facebook. Quando o usuário chega no último *post* presente na lista, a aplicação automaticamente busca por *posts* mais antigos e os inclui na lista para que o usuário possa visualizá-los. Para obter *posts* mais recentes aciona-se a tecla colorida correspondente. Estas duas situações invocam o método *update()* da classe Connector passando o valor de parâmetro *top* ou *bottom*, conforme o caso. No corpo do método uma *String* representando o endereço será manipulada com a concatenação deste parâmetro a fim de executar o serviço corretamente. A resposta do servidor será no formato já descrito, passando pelo *parser*, e os novos objetos Post obtidos serão incluídos no início ou final da lista.

4 CONCLUSÃO E TRABALHOS FUTUROS

4.1 Conclusão

O primeiro grande desafio deste trabalho foi encontrar um ambiente apropriado para o desenvolvimento das aplicações Ginga-J. Para tanto, seria necessário dispor de uma plataforma que contivesse a implementação, ao menos de grande parte, do *middleware* Ginga pois, como vimos, a máquina de execução depende das camadas inferiores do *middleware*. Dentre as alternativas encontradas existiam desde emuladores de outros padrões (XleTView, do padrão europeu), que contém o Java TV e algumas bibliotecas equivalentes ao Ginga, até implementações parciais do próprio Ginga, como o Open Ginga. A isso somam-se ainda os pouquíssimos exemplos de aplicações ou exemplos muito simples que não se aprofundavam muito nas APIs do Ginga-J. Inúmeras vezes recorremos às normas ABNT que regem o nosso sistema de TV Digital, em especial à ABNT 15606 que trata do *middleware*, porém, a leitura de normas é um tanto quanto complexa e suas definições não raro são vagas ou fora de contexto, dificultando o entendimento. Esta situação seria amenizada com a existência de aplicações de exemplo, podendo os interessados lançar mão destas fontes de "baixo" e "alto" nível que, ao nosso ver, se complementam.

Assim como as demais áreas de interesse comum, não só aquelas ligadas a tecnologia, a TV Digital possui uma comunidade que se comunica através de fóruns. Notamos que esta comunidade ultimamente está meio inerte e a maioria de suas questões dizem respeito ao Ginga-NCL. A própria TOTVS, uma das maiores empresas atuantes neste setor, mantém um fórum voltado a discussões sobre TV Digital em que registramos algumas dúvidas e até o momento não obtivemos respostas.

Apesar destas dificuldades devemos lembrar que praticamente todos os lares brasileiros possuem um televisor, fazendo deste um dos principais meios de comunicação do país. Com este cenário em mente, somos levados a pensar que, sem dúvidas, a TV Digital no Brasil possui um grande potencial considerando o perfil sócio-econômico de sua população. Embora o mercado da televisão, com emissoras e fabricantes de TV, esteja inserido dentro deste contexto favorável, o fato é que observa-se muito pouco avanço quando o assunto é interatividade. Por outro lado, não se pode olvidar de que este padrão, o SBTVD, é um padrão **brasileiro**, ou seja, não se deve atribuir toda a responsabilidade ao mercado para que este padrão seja consolidado de uma vez por todas. Sendo fruto de esforços do governo brasileiro, entendemos que os órgãos governamentais devem contribuir para este desenvolvimento financiando projetos de pesquisa, sejam de instituições privadas ou públicas.

Este trabalho pode ser encarado, também, como uma forma de disseminação da TV Digital e suas capacidades, já que, aliando-a às redes sociais, mais especificamente ao Facebook, há muito mais chances das pessoas tomarem conhecimento das potencialidades dessa tecnologia. O Brasil é o segundo país em número de usuários nesta rede, estima-se que algo em torno de 61 milhões (SOCIALBAKERS, 2012), fazendo da mesma um meio muito ativo no que tange a propagação de notícias e novidades. Por ser uma rede muito ativa e concentrar muitas pessoas, inúmeras oportunidades são identificadas e novos tipos de serviços e programas voltados ao Facebook são desenvolvidos a cada dia, gerando uma demanda por documentação e informações acerca de como a rede trabalha. Necessidade esta atendida, principalmente, através da página voltada a desenvolvedores (FACEBOOK, 2012f), mantida pelo próprio Facebook, que contém informações relativas às suas API's, facilitando o

desenvolvimento de aplicações que de certa forma fazem uso de seus serviços, como é o caso deste trabalho.

Por fim, acreditamos que o objetivo principal desta empreitada foi alcançado, qual seja, a integração entre redes sociais e TV Digital. Através do desenvolvimento do sistema Ginga Social, proporcionamos o contato entre estas duas diferentes mídias, considerando as limitações de ambas, principalmente no lado da TV. Outrossim, os estudos referentes aos dois temas também foram satisfatórios à medida que deram suporte para que o trabalho fosse concluído.

4.2 Trabalhos Futuros

Ao final dessa jornada, identificamos pontos importantes a serem continuados na integração entre TV digital interativa e redes sociais. Alguns pontos aqui citados foram idealizados pelos autores, porém, devido a impasses tecnológicos e também escassez de material para consulta, são indicados como trabalhos futuros:

- Expansão no GSTV, adicionando um teclado virtual para possibilitar ao usuário escrever uma mensagem e postá-la em sua linha do tempo no Facebook, ou ainda, utilizar o teclado para comentar em postagens de seus amigos.
- Adicionar serviços no GSW para obtenção da sua própria linha do tempo, bem como seu álbum e criar um *layout* para ser mostrado na TV.
- Compartilhamento da tela (*screenshot*) da televisão diretamente no álbum do usuário no Facebook e compartilhamento automático da programação a que está sendo assistida.

- Retenção dos dados dos usuários sobre programação assistida para posterior mineração dos dados e obtenção de informação a respeito dos telespectadores de cada rede de televisão.
- Testes reais da aplicação desenvolvida em um *set-top-box* compatível com o padrão brasileiro.

5 BIBLIOGRAFIA

ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15606-1. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital. Parte 1: Codificação de dados. 2011.

ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15606-2. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações. 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15606-3. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 3: Especificação de transmissão de dados. 2010.

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

ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15606-6. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 6: Java
DTV 1.3. 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15606-8. Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital
Parte 8: Ginga-J - Diretrizes operacionais para a ABNT NBR 15606-4. 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15607-1. Televisão digital terrestre - Canal de interatividade. Parte 1: Protocolos, interfaces físicas e interfaces de software. 2011

Amazon Elastic Compute Cloud (Amazon EC2). Disponível em: <http://aws.amazon.com/ec2/>. Acesso em: Agosto, 2012.

APACHE SOFTWARE FOUNDATION. Apache Tomcat. Disponível em: <http://tomcat.apache.org/>. Acesso em: Agosto, 2012.

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

ARIB. ARIB STD-B23 Version 1.1: Application Execution Engine Platform for Digital Broadcasting (English Translation). ARIB Standard, 2004.

ATSC. Advanced Television Systems Committee, www.atsc.org. Acesso em: Dezembro, 2011.

ATSC A-52. Advanced Television Systems Committee: Standard A/52A - Digital Audio Compression (AC-3) Standard, Revision A, 2001.

ATSC A-53. Advanced Television Systems Committee A/53: ATSC Digital Television Standard, Parts 1-6, 2007.

BARBOSA, Simone Diniz Junqueira; SOARES, Luiz Fernando Gomes. TV Digital Interativa no Brasil se Faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade, 2008. Disponível em: <http://www.ncl.org.br/documentos/JAI2008.pdf>. Acesso em: Dezembro, 2011.

BARROS, Juliana R. B. Diniz; SILVA, Adriana R.; BARROS, Roberto S. M.; FERRAZ, Carlos A. G.; ROSA, Nelson S. Projetando um Serviço de Descoberta de Canais para TV Digital. Faculdade Integrada do Recife, 2004.

BOYD, d.; ELLISON, N. Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication, n. 13, v. 1, art. 11. 2007. Disponível em <http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html>. Acesso Novembro, 2011.

BRASIL. Decreto nº 4.901, 26 de novembro de 2003. Institui o Sistema Brasileiro de Televisão Digital - SBTVD, e dá outras providências. Diário Oficial da União, 27 nov. 2003, p7.

BRASIL. Decreto nº 5.820 - 29 de junho de 2006. Dispõe sobre a implantação do SBTVD-T, estabelece diretrizes para a transição do sistema de transmissão analógica para o sistema de transmissão digital do serviço de radiodifusão de sons e imagens e do serviço de retransmissão de televisão, e dá outras providências. Diário Oficial da União, 30 jun. 2006, p51

BRAY, Tim; PAOLI, Jean; SPERBERG-MCQUEEN, C. M.; MALER, Eve; YEARGEAU, François. Extensible Markup Language (XML) 1.0 (Fifth Edition). Disponível em: <http://www.w3.org/TR/REC-xml/>. Acesso em: Agosto, 2012.

CPqD. Especificação Técnica de Referência OS 40.544. Campinas-SP, 2006 *apud* BITTENCOURT, Fábio Antônio, BENNERT, Wagner Alves. TV Digital: uma análise

das modulações e codificações de áudio e vídeo no modelo terrestre. Curitiba: Universidade Tecnologia Federal do Paraná, 2007.

DAMASCENO, Jean Ribeiro. Middleware Ginga, 2008. Universidade Ferderal Fluminense. Disponível em: <http://www.midiacom.uff.br/~debora/fsmm/trab-2008-2/middleware.pdf>. Acesso em: Novembro, 2012.

DASE. DTV application software environment level 1 (DASE-1) Part 1: introduction, architecture, and common facilities. 2003. Disponível em:<http://www.atsc.org/cms/standards/a100/a_100_1.pdf>. Acesso em: Julho, 2012.

DiBEG. Digital Broadcasting Experts Group. Disponível em: www.dibeg.org. Acesso em: Dezembro, 2011.

DONNELLY, David, F. HDTV Standards Setting: Politics, Technology, and Industry. [S.l. s.n], 1995. Disponível em <http://www.tfi.com/pubs/ntq/articles/view/95Q3_A4.pdf>. Acesso em 15/04/2005 *apud* MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: conceitos, desafios e perspectivas para o Brasil. Florianópolis: Ed. da UFSC, 2ª edição, 2005.

DVB. Digital Video Broadcasting Project. Disponível em: www.dvb.org. Acesso em: Dezembro, 2011.

ETSI. ETSI EN 300 744 - Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television, Edition 1.4.1. 2001.

ETSI. ETSI EN 300 421 - Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for 11/12 GHz Satellite Services, Edition 1.1.2. 1997a.

ETSI. ETSI EN 300 748, "Digital Video Broadcasting (DVB): Multipoint Video Distribution Systems (MVDS) at 10 GHZ and above, Edition 1.1.2. 1997b.

ETSI. ETSI EN 300 749 - Digital Video Broadcasting (DVB): Microwave Multipoint Distribution Systems (MMDS) below 10 GHz, Edition 1.1.2.1997c.

ETSI. ETSI EN 300 429 - Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for Cable Systems, Edition 1.2.1. 1998.

ETSI. ETSI ES 201 812 - Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.0.3. 2006.

ETSI. ETSI TS 102 819 V1.2.1 - GEM – Globally Executable MHP: A Guide to Platform Harmonisation, DVB Project Office White Paper on iTV Platform Harmonisation. 2005.

FACEBOOK. ESTATÍSTICAS FACEBOOK. Disponível em: <http://www.facebook.com/press/info.php?statistics>. Acesso em: Outubro, 2012a.

FACEBOOK. Open Graph Overview. Disponível em: <https://developers.facebook.com/docs/concepts/opengraph/overview/>. Acesso em: Outubro, 2012b

FACEBOOK. Getting Started: The Graph API. Disponível em: <http://developers.facebook.com/docs/getting-started/graphapi/>. Acesso em: Outubro, 2012c.

FACEBOOK. Graph API. Disponível em: http://developers.facebook.com/docs/reference/api/. Acesso em: Outubro,2012d.

FACEBOOK. Access Tokens and types. Disponível em: <https://developers.facebook.com/docs/concepts/login/access-tokens-and-types/>. Acesso em: Abril, 2012e.

FACEBOOK. Developers Facebook. Disponível em: <http://developers.facebook.com/>. Acesso em: Setembro, 2012f.

FACEBOOK. Graph API Explorer. Disponível em: <https://developers.facebook.com/tools/explorer>. Acesso em: Dezembro, 2011.

FERNANDES, Jorge; LEMOS, Guido; SILVEIRA, Gledson. Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas. In: Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação. 2004, Salvador. Disponível em: <http://www.cic.unb.br/~jhcf/MyBooks/itvdi/texto/itvdi.pdf>. Acesso em: Dezembro, 2011.

FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Acesso em: Julho, 2012.

FOROUZAN, Behrouz A. Sinais. Comunicação de Dados e Redes de Computadores / Behrouz A. Forouzan; tradução Glayson Eduardo de Figueiredo. Porto Alegre: Bookman. 2006. Cap nº 3.

GALO, Bruno. Você pode ganhar muito dinheiro no Facebook. Istoé Dinheiro. 2011. Disponível em: http://www.istoedinheiro.com.br/noticias/65157_VOCE+PODE+GANHAR+MUITO+DINHEIRO+NO+FACEBOOK. Acesso em: Setembro, 2012.

GALPERIN, Hernan. Comunicación e integración en la era digital: Un balance de la transición hacia la televisión digital en Brasil y Argentina. RevistanEletrônica Telos, Madrid, Espanha, 2003 *apud* MONTEZ, Carlos; BECKER, Valdecir. TV Digital

Interativa: conceitos, desafios e perspectivas para o Brasil. Florianópolis: Ed. da UFSC, 2ª edição, 2005.

GAWLINSKI, Mark. Interactive television production. Oxford, Focal Press, 2003.

GHISI, Bruno Cavaler; LOPES, Guilherme Figueiredo; SIQUEIRA, Frank. Integração de Aplicações para TV Digital Interativa com Redes Sociais, 2010. Disponível em:<http://www.inf.ufsc.br/~frank/papers/WTVDI2010-Bruno.pdf>. Acesso em: Novembro, 2011.

GLIFFY. Online Diagram Software and Flowchart Software - Gliffy. Disponível em: <http://www.gliffy.com/>. Acesso em: Agosto, 2012.

HIBERNATE - Relational Persistence for Java and .NET. Disponível em: <http://www.hibernate.org/>. Acesso em: Agosto, 2012.

HOUAISS, Antônio; VILLAR, Mauro de Salles. Dicionário Houaiss da Língua Portuguesa. Rio de Janeiro: Objetiva, 2001.

ISO. ISO/IEC 13818-1 - Information Technology – Generic Coding of Moving Pictures and Associated Audio Information – Part 1: Systems (MPEG-2 Systems), 1996a.

ISO. ISO/IEC 13818-2 - Information Technology – Generic Coding of Moving Pictures and Associated Audio Information – Part 2: Video (MPEG-2 Video), 1996b.

JQUERY FOUNDATION. jQuery - JavaScript library. Disponível em: <http://jquery.com/>. Acesso em: Agosto, 2012.

JSON JavaScript Object Notation. Introducing JSON. Disponível em: <http://json.org/>. Acesso em: Novembro, 2011.

JUNIOR SARAIVA, Erisvaldo Gadelha Saraivaa. Ginga-J Emulator: Uma Ferramenta de Execução de Aplicações Imperativas para o Middleware Ginga. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação). Universidade Federal da Paraíba.João Pessoa, 2010.

KULESZA, Raoni; et al.; Ginga-J: Implementação de Referência do Ambiente Imperativo do Middleware Ginga, 2010. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/webmedia/2010/05_webmi_c.pdf>. Acesso em: Novembro, 2012.

Laboratorio TeleMídia da PUC-Rio. Ambiente para Desenvolvimento de Aplicações Declarativas para a TV Digital Brasileira. Depto. Informática PUC-Rio, 2007. Disponível em: <http://gredes.ifto.edu.br/wp-content/uploads/MDIC2007.pdf>. Acesso em: Novembro, 2011.

LEITE, Luiz Eduardo Cunha et al. FlexTV – Uma Proposta de Arquitetura de *Middleware* para o Sistema Brasileiro de TV Digital. Revista de Engenharia de Computação e Sistemas Digitais, 2005, Vol. 2, pp. 29-50.

LEMOS, André L.M. Anjos interativos e retribalização do mundo: sobre interatividade e interfaces digitais, 1997.

LOPES, Denise Maria Moura da Silva. Sistema Brasileiro de Tv Digital: Caminhos percorridos e implantação. Disponível em: <http://rp-bahia.com.br/biblioteca/hist-midia2005/resumos/R0097-1.pdf>. Acesso em: Dezembro, 2011.

LUA. A linguagem de programação Lua. Disponível em: <http://www.lua.org/portugues.html>. Acesso em: Novembro, 2011.

MACHADO, Daniel. Configurando Hibernate com MySQL. Disponível em: <http://www.k19.com.br/artigos/configurando-hibernate-com-mysql/>. Acesso em: Agosto, 2012.

MACLIN, Bem. What Every Marketer Needs to Know about iTV. Nova Iorque, eMarketer Analyst Brief, 2001 *apud* MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: conceitos, desafios e perspectivas para o Brasil. Florianópolis: Ed. da UFSC, 2ª edição, 2005.

MATTOS, Sérgio. História da Televisão Brasileira: Uma visão econômica, social e política, 2ª Edição, Petrópolis, Vozes, 2002.

MCLUHAN, Herbert Marshall. Os Meios de Comunicação como Extensões do Homem. São Paulo: Cultrix, 1964 *apud* MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: conceitos, desafios e perspectivas para o Brasil. Florianópolis: Ed. da UFSC, 2ª edição, 2005.

MELLO, Ronaldo dos Santos. Sistema Gerenciador de Banco de Dados. Disponível em: <http://www.inf.ufsc.br/~ronaldo/ine5613/2-sgbd.pdf>. Acesso em: 17 de fevereiro de 2012

MENDES, Luciano Leonel. Artigo: SBTVD – Uma Visão sobre a TV Digital No Brasil. T&C Amazônia Ano V, Numero 12, Outubro de 2007.

MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: conceitos, desafios e perspectivas para o Brasil. Florianópolis: Ed. da UFSC, 2ª edição, 2005.

MONTEZ, Carlos; BECKER, Valdecir; FILHO, Günter H. Herweg. Datacasting e Desenvolvimento de Serviços e Aplicações para TV Digital Interativa. 2005. Disponível em: <http://www.tvdi.inf.br/site/artigos/Assuntos%20Diversos/Datacasting%20e%20Desenvolvimento%20de%20Servicos%20e%20Aplicacoes%20para%20TV%20Digital%20Int

erativa%20-%20BECKER,%20PICCIONI,%20MONTEZ,%20HERWEG.pdf>. Acesso em: Julho, 2012.

MORRIS, S. Smith-Chaigneau, 2005. A. Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV. Focal Press *apud* KULESZA, Raoni; *et al*.; Ginga-J: Implementação de Referência do Ambiente Imperativo do *Middleware* Ginga. 2010. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/webmedia/2010/05_webmi_c.pdf>. Acesso em 13 de novembro de 2012.

MySQL. Disponível em: <http://www.mysql.com/>. Acesso em: Agosto, 2012.

NETO, Moacyr Franco de Moraes. Um modelo para obtenção de previsibilidade temporal em aplicações Java para TV Digital. Dissertação (mestrado). Universidade Federal de Santa Catarina. Florianópolis, 2011

NIELSENWIRE. Americans Using TV and Internet Together 35% More Than A Year Ago. 2010 Disponível em: <http://blog.nielsen.com/nielsenwire/online_mobile/three-screen-report-q409/>. Acesso em: Novembro, 2011.

OAuth Community Site. Disponível em: <http://oauth.net/>. Acesso em: Setembro, 2012.

OLIVEIRA, Antônio Carlos Albuquerque de; LACERDA, João Paulo Lopes de. A TV digital no Brasil e o desenvolvimento de aplicações interativas para o middleware Ginga. 2008. Trabalho de Conlusão de Curso (Graduação em Ciência da Computação). Universidade Federal de Sergipe. Aracaju. 2008.

ORACLE. Java SE Overview. Disponível em: <http://www.oracle.com/technetwork/java/javase/index.html>. Acesso em: Agosto, 2012.

ORACLE. JSP - Java Server Pages Technology. Disponível em: <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>. Acesso em: Agosto, 2012.

O'REILLY, Tim. What Is Web 2.0, 2005 Disponível em: <http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1>. Acesso em: Novembro, 2011.

PAES, Alexsandro; ANTONIAZZI, Renato. Padrões de Middleware para TV Digital. Universidade Federal Fluminense, 2005 Disponível em: <http://www.tvdi.inf.br/site/artigos/Assuntos%20Diversos/Padroes%20de%20Middleware%20para%20TV%20Digital%20-%20PAES,%20ANTONIAZZI.pdf>. Acesso em: Julho, 2012.

PAULINELLI, Fernanda; KULESZA, Raoni. Histórico Ginga-J. GingaCDN. Disponível em:

http://gingacdn.lavid.ufpb.br/projects/openginga/wiki/Hist%C3%B3rico_Ginga-J.
Acesso em: Julho, 2012.

RECUERO, Raquel da Cunha. Comunidades em Redes Sociais na Internet: Proposta de Tipologia baseada no Fotolog.com. Porto Alegre: UFRGS, 2006. Tese (Doutorado em Comunicação e Informação), Universidade Federal do Rio Grande do Sul. 2006.

RECUERO, Raquel da Cunha. Considerações sobre a Difusão de Informações em Redes Sociais na Internet. In: Intercom Sul, 2007, Passo Fundo-RS. Anais do VIII Congresso de Ciências da Comunicação da Região Sul, 2007.

REISMAN, Richard R. Rethinking Interactive TV – I want my Coactive TV. [S.l.] Teleshuttle Corporation, 2002. Disponível em <http://www.teleshuttle.com/cotv/CoTVIntroWtPaper.htm>. Acesso em 15/04/2005 *apud* MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: conceitos, desafios e perspectivas para o Brasil. Florianópolis: Ed. da UFSC, 2ª edição, 2005.

RIAN NOVOSTI. The world map of social networks. Rian Novosti. 2011. Disponível em:
<http://en.rian.ru/infographics/20110228/162792394.html>. Acesso em: Dezembro, 2011.

RODRIGUES, Ana; GOMES, Regina. Modulação COFDM – Uma proposta atrativa para os padrões de TV Digital. Revista Digital Online. Vol3. Agosto de 2004 *apud* OLIVEIRA, Antônio Carlos Albuquerque de; LACERDA, João Paulo Lopes de. A TV digital no Brasil e o desenvolvimento de aplicações interativas para o middleware Ginga. 2008. Trabalho de Conlusão de Curso (Graduação em Ciência da Computação). Universidade Federal de Sergipe. Aracaju, 2008.

RNP. GT Middleware. Disponível em <http://www.rnp.br/pd/gts2004-2005/middleware.html>.
Acesso em: Dezembro,2011.

SASAKE, Carina Satiko. Estudo Comparativo entre middlewares para TV digital. Universidade Federal de Mato Grosso. Cuiabá, 2007.

SERASA EXPERIAN. Facebook dobra participação no ranking de redes sociais em dezembro, de acordo com Hitwise. 2013. Disponível em: <http://www.serasaexperian.com.br/release/noticias/2013/noticia_01072.htm>. Acesso em: Janeiro, 2013.

SILVA, Rodrigo. Redes Sociais, Televisão e a Relação que está Fortalecendo Ambas, 2011. Disponível: <http://www.bravaitv.com.br/blog/index.php/category/pesquisas/>. Acesso em: Novembro, 2011.

SILVA, Lincoln David Nery, et al. Suporte para desenvolvimento de aplicações multiusuário e multidispositivo para TV Digital com Ginga. T&C Amazônia Magazine. N. 12, 2007, pp. 75-84.

SILVA, Lincoln David Nery. 2008. Uma Proposta de API para Desenvolvimento de Aplicações Multiusuário e Multidispositivo para TV Digital Utilizando o Middleware Ginga. Departamento de Informática, Universidade Federal da Paraíba. 2008. p. 77, Dissertação (mestrado).

SILVA, Jones Q. TV Digital Interativa. Universidade do Vale do Rio dos Sinos. São Leopoldo, 2003.

SOARES, Luiz Fernando Gomes. 2006. MAESTRO: The Declarative Middleware Proposal for the SBTVD. Proceedings of the 4th European Interactive TV Conference. 2006.

SOARES, Luiz Fernando Gomes. TV Interativa se Faz com Ginga. Disponível em: <http://www.telemidia.puc-rio.br/sites/telemidia.puc-rio.br/files/2009_06_soares.pdf>. 2009. Acesso em: Novembro, 2011.

SOCIALBAKERS. Facebook Statistics by Country. Disponível em: <http://www.socialbakers.com/facebook-statistics/>. Acesso em: Outubro, 2012.

TOBALDINI; Ricardo Ghisi. Arquitetura REST: um estudo de sua implementação em linguagens de programação. Trabalho de Conclusção de Curso (Graduação Ciências da Computação). Universidade Federal de Santa Catarina, Florianópolis, 2007.

TOTVS. Astro Box > Sobre. Disponível em: <https://www.astrodevnet.com.br/AstroDevNet/restrict/astrobox_sobre.html>. Acesso em: Março, 2012a.

TOTVS. O que é a Astro DevNet? Disponível em: <https://www.astrodevnet.com.br/AstroDevNet/sobre_oquee.html>. Acesso em: Março, 2012b.

WIKIPEDIA. Centro de Processamento de Dados. Disponível em: <http://pt.wikipedia.org/wiki/Centro_de_processamento_de_dados>. Acesso em: Fevereiro, 2012.

WIKIPEDIA. Mapeamento Objeto-relacional. Disponível em: <http://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional>. Acesso em: Agosto, 2012.

WIKIPEDIA. SOA - Service-Oriented Architecture. Disponível em: <http://pt.wikipedia.org/wiki/Service-oriented_architecture>. Acesso em: Setembro, 2012.

WIKIPEDIA. Data transfer object. Disponível em: <http://en.wikipedia.org/wiki/Data_transfer_object>. Acesso em: Setembro, 2012.

WIKIPEDIA. Uniform Resource Locator. Disponível em: <http://pt.wikipedia.org/wiki/URL>. Acesso em: Setembro, 2012.

W3C. Cascading Style Sheets. Disponível em: <http://www.w3.org/Style/CSS/>. Acesso em: Agosto, 2012.

W3C. JavaScript Web APIs. Disponível em: <http://www.w3.org/standards/webdesign/script.html>. Acesso em: Agosto, 2012.

ZIMERMANN, Filipe. Canal de Retorno em TV Digital: Tecnologias e abordagens para efetivação da interatividade televisiva. Florianópolis: Universidade Federal de Santa Catarina, 2007.

ANEXO A - *Interface* Xlet

```
public interface Xlet {
        public void initXlet( XletContext ctx );
                throws XletstateChangeException;
        public void startXlet();
                throws XletstateChangeException;
        public void pauseXlet();
        public void destroyXlet( Boolean uncondicional);
                throws XletstateChangeException;

            }
```

ANEXO B - Detalhes de implementação do Ginga Social Web (servidor)

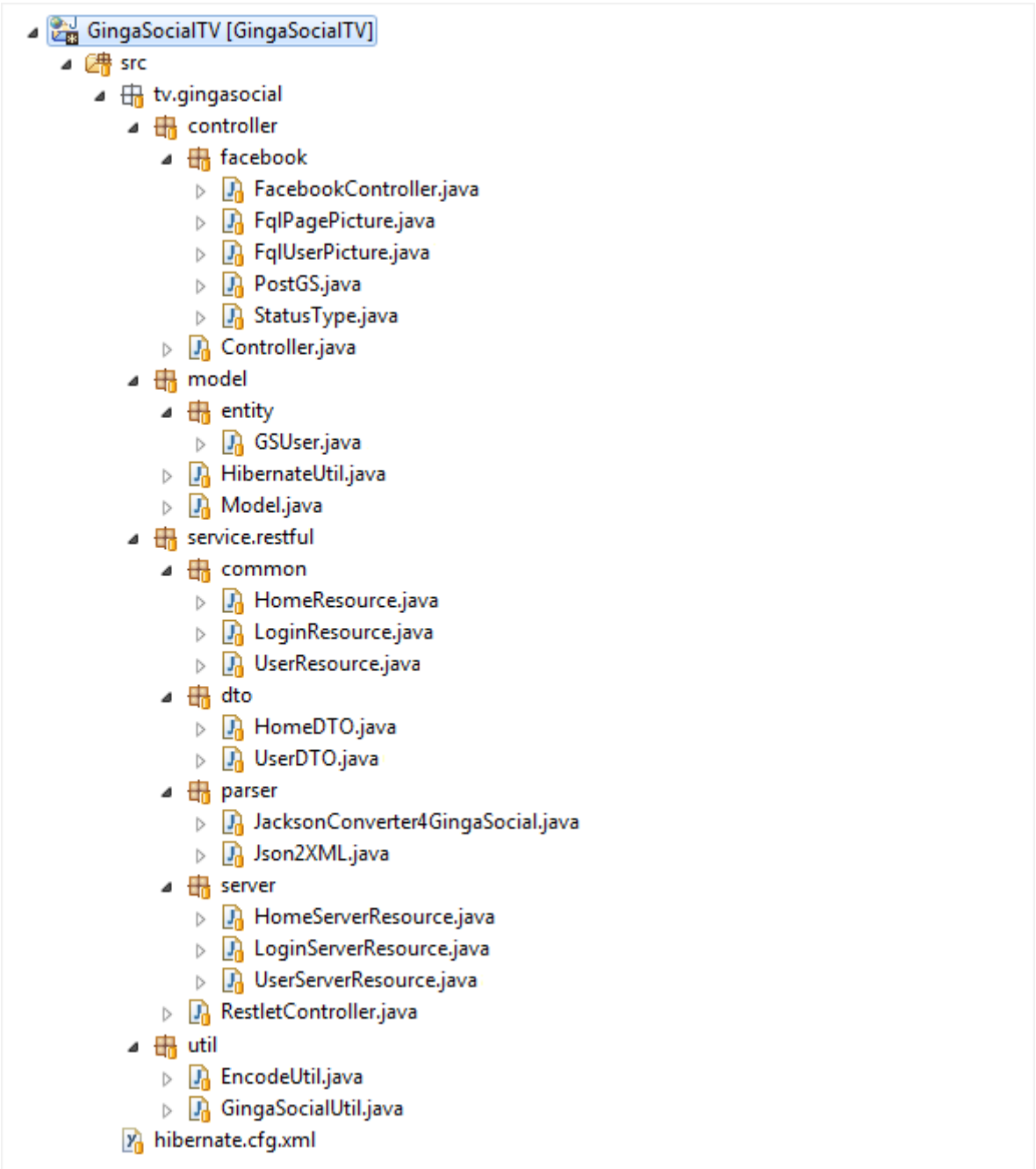


Figura 42 - Estrutura de diretórios do servidor GS no Eclipse

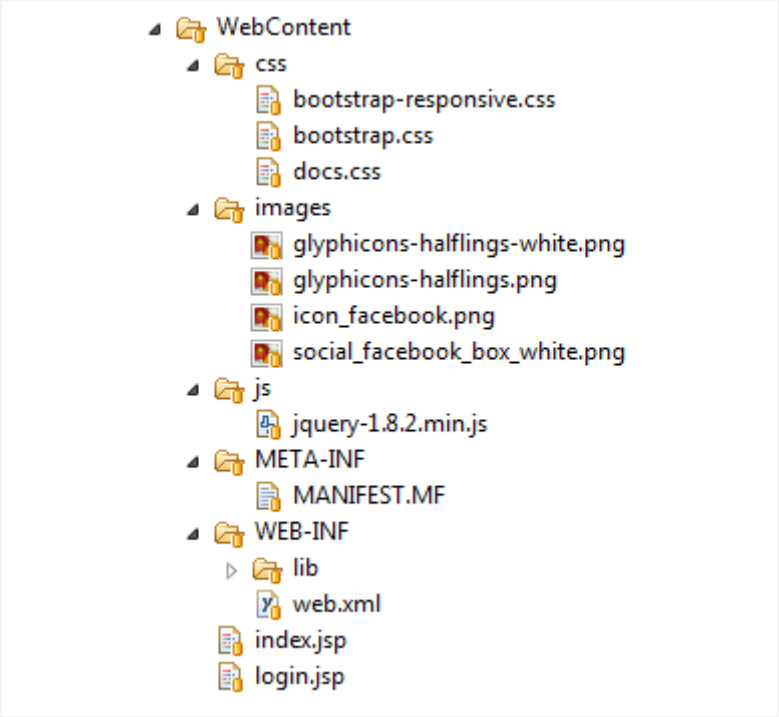


Figura 43 - Estrutura de diretórios com conteúdo para Web (visão)

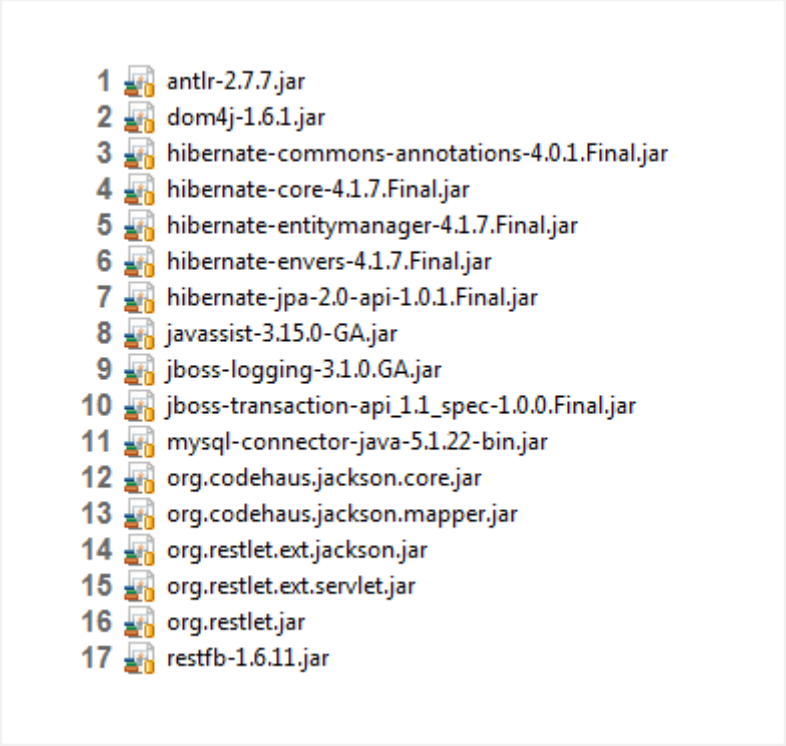


Figura 44 - Bibliotecas Java auxiliares no desenvolvimento do GSW

```java
/**
 * Creates a root Restlet that will receive all incoming calls.
 */
@Override
public synchronized Restlet createInboundRoot() {
    Engine engine = Engine.getInstance();
    engine.getRegisteredConverters().add(new JacksonConverter4GingaSocial());

    // Create a router Restlet that routes each call to a new instance of HelloWorldResource.
    Router router = new Router(getContext());

    // Defines only one routSe
    router.attach("/user", UserServerResource.class);
    router.attach("/login", LoginServerResource.class);
    router.attach("/home", HomeServerResource.class);
    router.attach("/like", LikeServerResource.class);
    router.attach("/share", ShareServerResource.class);

    return router;
}
```

Figura 45 - Classe RestletController.java para controle e gerenciamento dos serviços

FacebookController.java

```java
package tv.gingasocial.controller.facebook;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.hibernate.envers.tools.StringTools;

import com.restfb.Connection;
import com.restfb.DefaultFacebookClient;
import com.restfb.FacebookClient;
import com.restfb.Parameter;
import com.restfb.types.FacebookType;
import com.restfb.types.Post;
import com.restfb.types.User;
import com.restfb.util.StringUtils;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class FacebookController {

      private static final String APP_ID = "422583171132737";
```

```
	private static final String APP_SECRET =
"30825dc5b87745695d6fd863831909bc";
	private static final Integer DEFAULT_QUANT_ITEMS = 25;

	public static Connection<PostGS> getUserHome(String accessToken,
String facebookUserId) {
		return getUserHome(accessToken, facebookUserId, null,
null);
	}

	public static Connection<PostGS> getUserHome(String accessToken,
String facebookUserId, String queryName, Object queryValue) {
		FacebookClient facebookClient = new
DefaultFacebookClient(accessToken);

		Connection<PostGS> targetedSearch = null;
		if(StringUtils.isBlank(queryName) || queryValue == null) {
			targetedSearch =
facebookClient.fetchConnection("me/home", PostGS.class,
Parameter.with("locale", "pt_BR"));
		} else {
			targetedSearch =
facebookClient.fetchConnection("me/home", PostGS.class,
Parameter.with("locale", "pt_BR"), Parameter.with(queryName,
queryValue));
		}

		return targetedSearch != null ? targetedSearch : null;
	}

	public static Connection<PostGS> getUserHomeSince(String
accessToken, String facebookUserId, long since) {
		return getUserHome(accessToken, facebookUserId, "since",
(since / 1000));
	}

	public static Connection<PostGS> getUserHomeUntil(String
accessToken, String facebookUserId, long until) {
		return getUserHome(accessToken, facebookUserId, "until",
(until / 1000));
	}

	public static User getUser(String accessToken, String
facebookUserId) {
		FacebookClient facebookClient = new
DefaultFacebookClient(accessToken);
		User user = facebookClient.fetchObject(facebookUserId,
User.class);
		return user;
	}
```

```
	public static Map<String, String> getUsersPicSquare(List<String>
listFacebookIds) {
		String listString =
listFacebookIds.toString().replaceAll("\\[|\\]", "");
		String query = "SELECT uid, pic_square FROM user WHERE uid
IN (" + listString + ")";

		FacebookClient facebookClient = new
DefaultFacebookClient();
		List<FqlUserPicture> result =
facebookClient.executeQuery(query, FqlUserPicture.class);

		Map<String, String> map = new HashMap<String, String>();
		for (FqlUserPicture fqlUserPicture : result) {
			map.put(fqlUserPicture.getUid(),
fqlUserPicture.getPicSquare());
		}

		return map;
	}

	public static Map<String, String> getPagesPicSquare(List<String>
listFacebookIds) {
		String listString =
listFacebookIds.toString().replaceAll("\\[|\\]", "");
		String query = "SELECT page_id, pic_square FROM page WHERE
page_id IN (" + listString + ")";

		FacebookClient facebookClient = new
DefaultFacebookClient();
		List<FqlPagePicture> result =
facebookClient.executeQuery(query, FqlPagePicture.class);

		Map<String, String> map = new HashMap<String, String>();
		for (FqlPagePicture fqlPagePicture : result) {
			map.put(fqlPagePicture.getPageId(),
fqlPagePicture.getPicSquare());
		}

		return map;
	}

	public static String getUserPicSquare(String facebookId) {
		String query = "SELECT uid, pic_square FROM user WHERE uid
= " + facebookId;

		FacebookClient facebookClient = new
DefaultFacebookClient();
		List<FqlUserPicture> result =
facebookClient.executeQuery(query, FqlUserPicture.class);
```

```
            return result.size() > 0 ? result.get(0).getPicSquare() :
"";
      }

      public static String getUserName(String facebookUserId) {
            User user = getUser(null, facebookUserId);

            return user != null ? user.getName() : null;
      }

      public static String getLongLivedAccessToken(String accessToken)
{
            StringBuilder sb = new StringBuilder();

      sb.append("https://graph.facebook.com/oauth/access_token");
            sb.append("?client_id=" + APP_ID);
            sb.append("&client_secret=" + APP_SECRET);
            sb.append("&grant_type=fb_exchange_token");
            sb.append("&fb_exchange_token=" + accessToken);

            String longLivedAT = null;
            try {
                  URL url = new URL(sb.toString());
                  URLConnection connection = url.openConnection();
                  BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));

                  String inputLine;
                  while ((inputLine = in.readLine()) != null) {
                        longLivedAT = inputLine;
                  }

                  longLivedAT =
longLivedAT.split("&")[0].replaceAll("access_token=", "");
                  in.close();
            } catch (MalformedURLException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }

            return longLivedAT;
      }

      public static String getAppId() {
            return APP_ID;
      }

      public static String getAppSecret() {
            return APP_SECRET;
      }
```

```
	public static Integer getDefaultQuantItems() {
		return DEFAULT_QUANT_ITEMS;
	}
	public static boolean doLikePost(String accessToken, String
facebookId) {
		FacebookClient facebookClient = new
DefaultFacebookClient(accessToken);

		Post postData = getPostData(accessToken, facebookId);
		String objectId = postData.getObjectId();
		if(StringTools.isEmpty(objectId)) {
			String link =
postData.getActions().get(0).getLink();
			int lastIndex = link.lastIndexOf("/");
			objectId = link.substring(lastIndex, link.length());
		}

		String publish = facebookClient.publish(facebookId +
"/likes", String.class);
		boolean result = publish != null ?
publish.contains("true") : false;
		return result;
	}

	public static boolean doSharePost(String facebookUserId, String
accessToken, String facebookId) {
		FacebookClient facebookClient = new
DefaultFacebookClient(accessToken);

		Post postData = getPostData(accessToken, facebookId);
		String link = postData.getLink() == null ?
postData.getActions().get(0).getLink() : postData.getLink();
		String message = ""; //this is not necessary
		String picture = postData.getPicture() == null ? "" :
postData.getPicture();
		String name = postData.getName() == null ? "" :
postData.getName();
		String caption = postData.getCaption() == null ? "" :
postData.getCaption();
		String desc = postData.getDescription() == null ? "" :
postData.getDescription();


		Parameter[] listParams = {
			Parameter.with("link", link),
Parameter.with("message", message),
			Parameter.with("picture", picture),
Parameter.with("name", name),
			Parameter.with("caption", caption),
Parameter.with("description", desc)
```

```
            };

            FacebookType publish =
facebookClient.publish(facebookUserId + "/links", FacebookType.class,
listParams);
            return !StringTools.isEmpty(publish.getId());
      }

      public static boolean doPost(String facebookUserId, String
accessToken, String link, String message,
                  String picture, String name, String caption, String
desc) {
            FacebookClient facebookClient = new
DefaultFacebookClient(accessToken);

            Parameter[] listParams = {
                        Parameter.with("link", link),
Parameter.with("message", message),
                        Parameter.with("picture", picture),
Parameter.with("name", name),
                        Parameter.with("caption", caption),
Parameter.with("description", desc)
            };

            FacebookType publish =
facebookClient.publish(facebookUserId + "/links", FacebookType.class,
listParams);
            return !StringTools.isEmpty(publish.getId());
      }

      public static Post getPostData(String accessToken, String
facebookId) {
            FacebookClient facebookClient = new
DefaultFacebookClient(accessToken);
            PostGS post = facebookClient.fetchObject("/" + facebookId,
PostGS.class, Parameter.with("locale", "pt_BR"));

            return post;
      }
}
```

FqlPagePicture.java

```
package tv.gingasocial.controller.facebook;

import com.restfb.Facebook;

/**
* @author Gustavo Rupp Santos
```

```
* @author Fiodor Castro
*/
public class FqlPagePicture {

	@Facebook("page_id")
	private String pageId;

	@Facebook("pic_square")
	private String picSquare;

	public String getPageId() {
		return pageId;
	}

	public void setPageId(String pageId) {
		this.pageId = pageId;
	}

	public String getPicSquare() {
		return picSquare;
	}

	public void setPicSquare(String pic) {
		this.picSquare = pic;
	}
}
```

FqlUserPicture.java

```
package tv.gingasocial.controller.facebook;

import com.restfb.Facebook;

/**
* @author Gustavo Rupp Santos and Fiodor Castro
* @author Fiodor Castro
*/
public class FqlUserPicture {

	@Facebook
	private String uid;

	@Facebook
	private String pic_square;

	public String getUid() {
		return uid;
	}

	public void setUid(String uid) {
		this.uid = uid;
	}
```

```
	public String getPicSquare() {
		return pic_square;
	}

	public void setPicSquare(String picSquare) {
		this.pic_square = picSquare;
	}

}
```

PostGS.java

```
package tv.gingasocial.controller.facebook;

import com.restfb.Facebook;
import com.restfb.types.Post;

/**
* @author Gustavo Rupp Santos and Fiodor Castro
* @author Fiodor Castro
*/
public class PostGS extends Post {


	private static final long serialVersionUID = 696969696969L;

	@Facebook
	private String story;

	@Facebook("status_type")
	private String statusType;

	public String getStory() {
		return story;
	}

	public void setStory(String story) {
		this.story = story;
	}

	public String getStatusType() {
		return statusType;
	}

	public void setStatusType(String statusType) {
		this.statusType = statusType;
	}

}
```

StatusType.java

```
package tv.gingasocial.controller.facebook;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public enum StatusType {
      SHARED_STORY,
      TAGGED_IN_PHOTO,
      ADDED_PHOTOS,
      MOBILE_STATUS_UPDATE;
}
```

Controller.java

```
package tv.gingasocial.controller;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Random;

import org.hibernate.envers.tools.StringTools;

import tv.gingasocial.controller.facebook.FacebookController;
import tv.gingasocial.controller.facebook.PostGS;
import tv.gingasocial.model.HibernateUtil;
import tv.gingasocial.model.Model;
import tv.gingasocial.model.entity.GSUser;
import tv.gingasocial.service.restful.dto.HomeDTO;
import tv.gingasocial.service.restful.dto.UserDTO;
import tv.gingasocial.util.EncodeUtil;

import com.restfb.Connection;
import com.restfb.types.Post;
import com.restfb.types.User;
import com.restfb.util.StringUtils;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class Controller {

      private static final String FORMAT = "%06d"; //if necessary,
complete left side with zeros until 6 numbers
      private static final int MAX = 999999; //maximum range of
password
```

```
	private static final SimpleDateFormat DATE_FORMATTER = new
SimpleDateFormat("dd/MM/yy hh:mm");

	public static GSUser registerOrUpdateUser(String facebookUserId,
String accessToken) {
		GSUser gingaUser = null;
		User user = FacebookController.getUser(accessToken,
facebookUserId);
		String longLived =
FacebookController.getLongLivedAccessToken(accessToken);
		if(longLived != null) {
			accessToken = longLived;
		}

		//check if user already exists
		if(Model.hasUserRegistered(facebookUserId)) {
			//update the important fields in DB
			gingaUser = Model.getGingaUser(facebookUserId);
			gingaUser.setName(user.getName());
			gingaUser.setEmail(user.getEmail());
			gingaUser.setAccessToken(accessToken);

	gingaUser.setUpdateAccessTokenDate(System.currentTimeMillis());

			HibernateUtil.update(gingaUser);
		} else {
			//generate a random password
			Random random = new Random();
			String password = String.format(FORMAT,
random.nextInt(MAX));

			gingaUser = new GSUser();
			gingaUser.setFacebookUserId(facebookUserId);
			gingaUser.setName(user.getName());
			gingaUser.setPassword(password);
			gingaUser.setEmail(user.getEmail());
			gingaUser.setAccessToken(accessToken);

	gingaUser.setUpdateAccessTokenDate(System.currentTimeMillis());

			HibernateUtil.insert(gingaUser);
		}

		return gingaUser;
	}


	public static List<HomeDTO> getFacebookUserHomeDTO(long
gingaUserId, String direction, int quantItems) {
		GSUser gingaUser = (GSUser)
HibernateUtil.select(GSUser.class, gingaUserId);
```

```
            String accessToken = gingaUser.getAccessToken();
            String facebookUserId = gingaUser.getFacebookUserId();
            quantItems = gingaUser.getQuantItems() + quantItems;

            Connection<PostGS> homeConnection = null;

            if(StringTools.isEmpty(direction) || quantItems == 0) {
                  homeConnection =
FacebookController.getUserHome(accessToken, facebookUserId);
            } else if(direction.equals("top")) {
                  homeConnection =
FacebookController.getUserHomeSince(accessToken, facebookUserId,
gingaUser.getTopUpdatedTime());
            } else if(direction.equals("bottom")) {
                  homeConnection =
FacebookController.getUserHomeUntil(accessToken, facebookUserId,
gingaUser.getBottomUpdatedTime());
            }

            List<HomeDTO> postsList = new ArrayList<HomeDTO>();

            List<String> listFacebookUserIds = new
ArrayList<String>();
            List<String> listFacebookPagesIds = new
ArrayList<String>();
            List<PostGS> dataPosts = homeConnection.getData();
            for(Post post : dataPosts) {
                  String fbId = post.getFrom().getId();

      if(StringUtils.isBlank(post.getFrom().getCategory())) {
                        listFacebookUserIds.add(fbId);
                  } else {
                        listFacebookPagesIds.add(fbId);
                  }
            }

            Map<String, String> mapPictures =
FacebookController.getUsersPicSquare(listFacebookUserIds);
mapPictures.putAll(FacebookController.getPagesPicSquare(listFacebookPa
gesIds));

            for(PostGS post : dataPosts) {
                  String facebookId = post.getId();
                  if(facebookId.startsWith("0_")) {
                        continue;
                  }

                  String userFacebookId = post.getFrom().getId();
                  String userName = post.getFrom().getName();
                  String userAvatarURL =
mapPictures.get(post.getFrom().getId());
```

```
                String type = post.getType();
                String statusType = post.getStatusType();
                String message = post.getMessage();
                String photoURL = null;
                String updatedTime =
DATE_FORMATTER.format(post.getUpdatedTime());
                long likes = post.getLikes() != null ?
post.getLikes().getCount() : 0;
                long comments = post.getComments() != null ?
post.getComments().getCount() : 0;

                if(type.equals("photo")) {
                      message = setPhotoMessage(post, statusType,
message, userName);
                      photoURL = post.getPicture();
                } else if(type.equalsIgnoreCase("video")) {
                      message = setVideoMessage(post, message,
userName);
                } else if(type.equalsIgnoreCase("status")) {
                      message = setStatusMessage(post, statusType,
message, userName);
                } else if(type.equalsIgnoreCase("link")) {
                      continue;
                } else if(type.equalsIgnoreCase("checkin")) {
                      message = "[CHECKIN] " + post.getCaption();
                }

                //change the encode to works on TV
                userName =
EncodeUtil.transformTextPlainToUTF8(userName);
                message =
EncodeUtil.transformTextPlainToUTF8(message);

                HomeDTO homeDTO = new HomeDTO(facebookId,
userFacebookId, userName, userAvatarURL, message, type, photoURL,
updatedTime, likes, comments);
                postsList.add(homeDTO);
          }

          if(dataPosts != null) {
                long topUpdatedTime =
dataPosts.get(0).getUpdatedTime().getTime();
                long bottomUpdatedTime =
dataPosts.get(dataPosts.size()-1).getUpdatedTime().getTime();
                gingaUser.setTopUpdatedTime(topUpdatedTime);
                gingaUser.setBottomUpdatedTime(bottomUpdatedTime);
                gingaUser.setQuantItems(quantItems);
                HibernateUtil.update(gingaUser);
          }

          return postsList;
```

```
	}

	private static String setStatusMessage(PostGS post, String
statusType, String message, String userName) {
		String story = post.getStory() != null ? post.getStory() :
"";

		if(message == null) {
			message = story;
		} else {
			message = story + "\n\n" + message;
		}

		return message;
	}


	private static String setPhotoMessage(PostGS post, String
statusType, String message, String userName) {
		String story = post.getStory();
		String caption = post.getCaption() != null ? "\n" +
post.getCaption() : "";
		String description = post.getDescription() != null ? "\n"
+ post.getDescription() : "";
		String msgAux = message != null ? "\n" + message : "";

		if(statusType == null) {
			message = story;
		} else if(statusType.equals("shared_story")) {
			message = story + caption;
		} else if(statusType.equals("tagged_in_photo")) {
			message = story + description;
		} else if(statusType.equals("added_photos")) {
			message = userName + msgAux;
		}

		return message;
	}

	public static String setVideoMessage(PostGS post, String
message, String userName) {
		if(StringTools.isEmpty(message)) {
			message = userName + "\n[VIDEO] " + post.getName();
		} else {
			message = userName + "\n" + message + "\n[VIDEO] " +
post.getName();
		}

		return message;
	}
```

```
	public static UserDTO getGingaUserDTO(Long gingaUserId) {
		GSUser gingaUser =
(GSUser)HibernateUtil.select(GSUser.class, gingaUserId);
		String facebookUserId = gingaUser.getFacebookUserId();
		String picSquareURL =
FacebookController.getUserPicSquare(facebookUserId);

		registerOrUpdateUser(facebookUserId,
gingaUser.getAccessToken());

		return new UserDTO(gingaUserId, facebookUserId,
gingaUser.getName(), picSquareURL);
	}

	public static boolean canLogin(long userId, String pass) {
		return Model.canLogin(userId, pass);
	}

	public static boolean doLikePost(String gingaUserId, String
facebookId) {
		Long gUserId = Long.parseLong(gingaUserId);
		GSUser gingaUser =
(GSUser)HibernateUtil.select(GSUser.class, gUserId);

		return
FacebookController.doLikePost(gingaUser.getAccessToken(), facebookId);
	}

	public static boolean doSharePost(String gingaUserId, String
facebookId) {
		Long gUserId = Long.parseLong(gingaUserId);
		GSUser gingaUser =
(GSUser)HibernateUtil.select(GSUser.class, gUserId);

		return
FacebookController.doSharePost(gingaUser.getFacebookUserId(),
gingaUser.getAccessToken(), facebookId);
	}

}
```

GSUser.java

```
package tv.gingasocial.model.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
/**
* @author Gustavo Rupp Santos and Fiodor Castro
* @author Fiodor Castro
*/
@Entity
@Table(name="GSUSER")
public class GSUser {

	@Id
	@Column
	@GeneratedValue
	private Long id;
	private String facebookUserId;
	private String name;
	private String password;
	private String accessToken;
	private String email;
	private long updateAccessTokenDate;
	private long topUpdatedTime;
	private long bottomUpdatedTime;
	private int quantItems;

	public Long getId() {
		return id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getAccessToken() {
		return accessToken;
	}

	public void setAccessToken(String accessToken) {
		this.accessToken = accessToken;
	}

	public String getFacebookUserId() {
```

```
		return facebookUserId;
	}

	public void setFacebookUserId(String facebookUserId) {
		this.facebookUserId = facebookUserId;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public long getUpdateAccessTokenDate() {
		return updateAccessTokenDate;
	}

	public void setUpdateAccessTokenDate(long updateAccessTokenDate)
{
		this.updateAccessTokenDate = updateAccessTokenDate;
	}

	public long getTopUpdatedTime() {
		return topUpdatedTime;
	}

	public void setTopUpdatedTime(long topUpdatedTime) {
		this.topUpdatedTime = topUpdatedTime;
	}

	public long getBottomUpdatedTime() {
		return bottomUpdatedTime;
	}

	public void setBottomUpdatedTime(long bottomUpdatedTime) {
		this.bottomUpdatedTime = bottomUpdatedTime;
	}

	public int getQuantItems() {
		return quantItems;
	}

	public void setQuantItems(int quantItems) {
		this.quantItems = quantItems;
	}

}
```

HibernateUtil.java

```
package tv.gingasocial.model;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;
import org.hibernate.service.ServiceRegistryBuilder;
import org.hibernate.tool.hbm2ddl.SchemaUpdate;

import tv.gingasocial.model.entity.GSUser;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class HibernateUtil {

      private static SessionFactory sessionFactory;
      private static ServiceRegistry serviceRegistry;
      private static Configuration hibernateConfig;

      static {
            try {
            hibernateConfig = new Configuration();
            hibernateConfig.configure();

            serviceRegistry = new
ServiceRegistryBuilder().applySettings(hibernateConfig.getProperties()
).buildServiceRegistry();
            sessionFactory =
hibernateConfig.buildSessionFactory(serviceRegistry);

            } catch (Throwable ex) {
                  System.err.println("Failed to create sessionFactory
object." + ex);
                  throw new ExceptionInInitializerError(ex);
            }
      }

      public static Session getSession() {
            return sessionFactory.openSession();
      }

   public static <T> Object select(Class<T> objClass, long id) {
       Object object = null;
       Session session = null;
       Transaction transaction = null;
```

```
    try {
         session = getSession();
         transaction = session.beginTransaction();
         object = session.get(objClass, id);
    } catch (HibernateException e) {
        transaction.rollback();
        System.err.println(e.fillInStackTrace());
    } finally {
        session.close();
    }

    return object;
}
public static void updateDB(){
    try {
        SchemaUpdate su = new SchemaUpdate(hibernateConfig);
        su.execute(true, true);
    } catch (Exception e) { }
}

public static boolean insert(Object obj) {
   Session session = null;
   Transaction transaction = null;
    try {
         session = getSession();
         transaction = session.beginTransaction();
         session.save(obj);
        transaction.commit();
    } catch (HibernateException e) {
         transaction.rollback();
        System.err.println(e.fillInStackTrace());
        return false;
    } finally {
        session.close();
    }

    return true;
}

public static boolean update(Object obj) {
   Session session = null;
   Transaction transaction = null;
    try {
         session = getSession();
         transaction = session.beginTransaction();
         session.update(obj);
        transaction.commit();
    } catch (HibernateException e) {
         transaction.rollback();
        System.err.println(e.fillInStackTrace());
        return false;
```

```
        } finally {
            session.close();
        }

        return true;
    }

    public static boolean delete(Object obj){
        Session session = null;
        Transaction transaction = null;
        try {
             session = getSession();
             transaction = session.beginTransaction();
             session.delete(obj);
            transaction.commit();
        } catch (HibernateException e) {
             transaction.rollback();
            System.err.println(e.fillInStackTrace());
            return false;
        } finally {
            session.close();
        }

        return true;
    }

       public static void main(String[] args) {
             Session openSession = sessionFactory.openSession();
             Query createQuery = openSession.createQuery("FROM " +
GSUser.class.getSimpleName() + " WHERE facebookUserId = :fbUserId");
             createQuery.setString("fbUserId", "100001231232131");
       }

}
```

Model.java

```
package tv.gingasocial.model;

import java.util.List;

import org.hibernate.Query;

import tv.gingasocial.model.entity.GSUser;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class Model {
       public static boolean hasUserRegistered(String facebookUserId) {
```

```
            GSUser user = getGingaUser(facebookUserId);

            return user != null ? true : false;
      }

      @SuppressWarnings("unchecked")
      public static GSUser getGingaUser(String facebookUserId) {
            String hql = "FROM " + GSUser.class.getSimpleName() + "
WHERE facebookUserId = :fbUserId";
            Query query = HibernateUtil.getSession().createQuery(hql);
            query.setParameter("fbUserId", facebookUserId);

            List<GSUser> list = query.list();
            GSUser user = null;
            if(list.size() > 0) {
                  user = list.get(0);
            }

            return user;
      }

      public static boolean canLogin(long userId, String pass) {
            boolean result = false;
            GSUser gingaUser = (GSUser)
HibernateUtil.select(GSUser.class, userId);
            if(gingaUser.getPassword().equals(pass)) {
                  result = true;

                  gingaUser.setTopUpdatedTime(0);
                  gingaUser.setBottomUpdatedTime(0);
                  gingaUser.setQuantItems(0);
                  HibernateUtil.update(gingaUser);
            }
            return result;
      }

}
```

HomeResource.java

```
package tv.gingasocial.service.restful.common;

import org.restlet.resource.Get;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public interface HomeResource {

      @Get("txt")
```

```
	public String getFacebookHome();

}
```

LikeResource.java

```
package tv.gingasocial.service.restful.common;

import org.restlet.resource.Get;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public interface LikeResource {

	@Get
	public boolean doLike();

}
```

LoginResource.java

```
package tv.gingasocial.service.restful.common;

import org.restlet.resource.Get;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public interface LoginResource {

	@Get
	public boolean doLogin();

}
```

ShareResource.java

```
package tv.gingasocial.service.restful.common;

import org.restlet.resource.Get;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public interface ShareResource {

	@Get
```

```
	public boolean doShare();

}
```

UserResource.java

```
package tv.gingasocial.service.restful.common;

import org.restlet.resource.Get;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public interface UserResource {

	@Get("txt")
	public String getGingaUser();

}
```

HomeDTO.java

```
package tv.gingasocial.service.restful.dto;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class HomeDTO {
	private String facebookId;
	private String userFacebookId;
	private String userName;
	private String userAvatarURL;
	private String message;
	private String type;
	private String photoURL;
	private String updatedTime;
	private long likes;
	private long comments;

	public HomeDTO(String facebookId, String userFacebookId, String
userName,
			String userAvatarURL, String message, String type,
String photoURL,
			String updatedTime, long likes, long comments) {
		setFacebookId(facebookId);
		setUserFacebookId(userFacebookId);
		setUserName(userName);
		setUserAvatarURL(userAvatarURL);
```

```
        setMessage(message);
        setType(type);
        setPhotoURL(photoURL);
        setUpdatedTime(updatedTime);
        setLikes(likes);
        setComments(comments);
}

public String getFacebookId() {
        return facebookId;
}

public void setFacebookId(String facebookId) {
        this.facebookId = facebookId;
}

public String getUserFacebookId() {
        return userFacebookId;
}

public void setUserFacebookId(String userFacebookId) {
        this.userFacebookId = userFacebookId;
}

public String getUserName() {
        return userName;
}

public void setUserName(String userName) {
        this.userName = userName;
}

public String getUserAvatarURL() {
        return userAvatarURL;
}

public void setUserAvatarURL(String userAvatarURL) {
        this.userAvatarURL = userAvatarURL;
}

public String getMessage() {
        return message;
}

public void setMessage(String message) {
        this.message = message;
}

public String getType() {
        return type;
}
```

```
public void setType(String type) {
	this.type = type;
}

public String getUpdatedTime() {
	return updatedTime;
}

public void setUpdatedTime(String updatedTime) {
	this.updatedTime = updatedTime;
}

public long getLikes() {
	return likes;
}

public void setLikes(long likes) {
	this.likes = likes;
}

public long getComments() {
	return comments;
}

public void setComments(long comments) {
	this.comments = comments;
}

public String getPhotoURL() {
	return photoURL;
}

public void setPhotoURL(String photoURL) {
	this.photoURL = photoURL;
}

@Override
public String toString() {
	StringBuilder sb = new StringBuilder();
	sb.append("|");
	sb.append("'"+facebookId+"',");
	sb.append("'"+userFacebookId+"',");
	sb.append("'"+userName+"',");
	sb.append("'"+userAvatarURL+"',");
	sb.append("'"+message+"',");
	sb.append("'"+type+"',");
	sb.append("'"+photoURL+"',");
	sb.append("'"+updatedTime+"',");
	sb.append("'"+likes+"',");
	sb.append("'"+comments+"'");
```

```
            sb.append("|");

            return sb.toString();
      }

}
```

UserDTO.java

```
package tv.gingasocial.service.restful.dto;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class UserDTO {

      private Long gingaId;
      private String facebookUserId;
      private String name;
      private String avatarURL;

      public UserDTO(Long gingaId, String facebookUserId, String name,
String avatarURL) {
            this.gingaId = gingaId;
            this.facebookUserId = facebookUserId;
            this.name = name;
            this.avatarURL = avatarURL;
      }

      public Long getGingaId() {
            return gingaId;
      }

      public void setGingaId(Long gingaId) {
            this.gingaId = gingaId;
      }

      public String getFacebookUserId() {
            return facebookUserId;
      }

      public void setFacebookUserId(String facebookId) {
            this.facebookUserId = facebookId;
      }

      public String getName() {
            return name;
      }

      public void setName(String name) {
```

```
            this.name = name;
      }

      public String getAvatarURL() {
            return avatarURL;
      }

      public void setAvatarURL(String userAvatarURL) {
            this.avatarURL = userAvatarURL;
      }

      @Override
      public String toString() {
            StringBuilder sb = new StringBuilder();
            sb.append("|");
            sb.append("'"+gingaId+"',");
            sb.append("'"+facebookUserId+"',");
            sb.append("'"+name+"',");
            sb.append("'"+avatarURL+"'");
            sb.append("|");

            return sb.toString();
      }

}
```

JacksonConverter4GingaSocial.java

```
package tv.gingasocial.service.restful.parser;

import org.codehaus.jackson.map.annotate.JsonSerialize.Inclusion;
import org.restlet.ext.jackson.JacksonConverter;

public class JacksonConverter4GingaSocial extends JacksonConverter {

      public JacksonConverter4GingaSocial() {
            super();

      getObjectMapper().getSerializationConfig().setSerializationInclu
sion(Inclusion.NON_DEFAULT);
      }

}
```

HomeServerResource.java

```
package tv.gingasocial.service.restful.server;

import java.util.List;

import org.hibernate.envers.tools.StringTools;
```

```
import org.restlet.data.Form;
import org.restlet.resource.ResourceException;
import org.restlet.resource.ServerResource;

import tv.gingasocial.controller.Controller;
import tv.gingasocial.service.restful.common.HomeResource;
import tv.gingasocial.service.restful.dto.HomeDTO;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class HomeServerResource extends ServerResource implements
HomeResource {

	@Override
	public String getFacebookHome() {
		Form query = getReference().getQueryAsForm();

		String gingaUserStr =
query.getFirstValue("ginga_user_id");
		String direction = query.getFirstValue("direction");
		String quantItemsStr = query.getFirstValue("quant_items");
//quant items that TV wants

		Long gingaUserId = null;
		if(gingaUserStr != null) {
			gingaUserId = Long.parseLong(gingaUserStr);
		} else {
			throw new ResourceException(400); //absence of
user_id
		}

		int quantItems = 0;
		if(!StringTools.isEmpty(quantItemsStr)) {
			try {
				quantItems = Integer.parseInt(quantItemsStr);
			} catch (Exception e) { }
		}

		List<HomeDTO> facebookUserHome =
Controller.getFacebookUserHomeDTO(gingaUserId, direction, quantItems);

		StringBuilder sb = new StringBuilder();
		for (HomeDTO homeDTO : facebookUserHome) {
			sb.append(homeDTO);
		}

		return sb.toString();
	}
```

```
}
```

LikeServerResource.java

```
package tv.gingasocial.service.restful.server;

import java.net.URLDecoder;

import org.restlet.data.Form;
import org.restlet.resource.ResourceException;
import org.restlet.resource.ServerResource;

import tv.gingasocial.controller.Controller;
import tv.gingasocial.service.restful.common.LikeResource;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class LikeServerResource extends ServerResource implements
LikeResource {

      public boolean doLike() {
            Form query = getReference().getQueryAsForm();

            String gingaUserId;
            String facebookId;

            try {
                  gingaUserId =
URLDecoder.decode(query.getFirstValue("ginga_user_id"), "UTF-8");
                  facebookId =
URLDecoder.decode(query.getFirstValue("facebook_id"), "UTF-8");
            } catch (Exception e) {
                  throw new ResourceException(400);
            }

            return Controller.doLikePost(gingaUserId, facebookId);
      }

}
```

LoginServerResource.java

```
package tv.gingasocial.service.restful.server;

import java.net.URLDecoder;

import org.restlet.data.Form;
import org.restlet.resource.ResourceException;
import org.restlet.resource.ServerResource;
```

```
import tv.gingasocial.controller.Controller;
import tv.gingasocial.service.restful.common.LoginResource;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class LoginServerResource extends ServerResource implements
LoginResource {

      @Override
      public boolean doLogin() {
            Form query = getReference().getQueryAsForm();

            String login;
            String pass;

            try {
                  login =
URLDecoder.decode(query.getFirstValue("login"), "UTF-8");
                  pass =
URLDecoder.decode(query.getFirstValue("pass"), "UTF-8");
            } catch (Exception e) {
                  throw new ResourceException(404);
            }

            long userId = Long.parseLong(login);

            return Controller.canLogin(userId, pass);
      }

}
```

ShareServerResource.java

```
package tv.gingasocial.service.restful.server;

import java.net.URLDecoder;

import org.restlet.data.Form;
import org.restlet.resource.ResourceException;
import org.restlet.resource.ServerResource;

import tv.gingasocial.controller.Controller;
import tv.gingasocial.service.restful.common.ShareResource;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
```

```
*/
public class ShareServerResource extends ServerResource implements
ShareResource {

	@Override
	public boolean doShare() {
		Form query = getReference().getQueryAsForm();

		String gingaUserId;
		String facebookId; //represent the event on Facebook

		try {
			gingaUserId =
URLDecoder.decode(query.getFirstValue("ginga_user_id"), "UTF-8");
			facebookId =
URLDecoder.decode(query.getFirstValue("facebook_id"), "UTF-8");
		} catch (Exception e) {
			throw new ResourceException(400);
		}

		return Controller.doSharePost(gingaUserId, facebookId);
	}

}
```

UserServerResource.java

```
package tv.gingasocial.service.restful.server;

import org.restlet.data.Form;
import org.restlet.resource.ResourceException;
import org.restlet.resource.ServerResource;

import tv.gingasocial.controller.Controller;
import tv.gingasocial.service.restful.common.UserResource;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class UserServerResource extends ServerResource implements
UserResource {

	@Override
	public String getGingaUser() {
		Form query = getReference().getQueryAsForm();

		String gingaUserStr =
query.getFirstValue("ginga_user_id");
```

```
            Long gingaUserId = null;
            if(gingaUserStr != null) {
                  gingaUserId = Long.parseLong(gingaUserStr);
            } else {
                  throw new ResourceException(400); //absence of
user_id
            }

            return Controller.getGingaUserDTO(gingaUserId).toString();
      }

}
```

RestletController.java

```
package tv.gingasocial.service.restful;

import org.restlet.Application;
import org.restlet.Restlet;
import org.restlet.engine.Engine;
import org.restlet.routing.Router;

import
tv.gingasocial.service.restful.parser.JacksonConverter4GingaSocial;
import tv.gingasocial.service.restful.server.HomeServerResource;
import tv.gingasocial.service.restful.server.LikeServerResource;
import tv.gingasocial.service.restful.server.LoginServerResource;
import tv.gingasocial.service.restful.server.ShareServerResource;
import tv.gingasocial.service.restful.server.UserServerResource;

public class RestletController extends Application {

   @Override
   public synchronized Restlet createInboundRoot() {
      Engine engine = Engine.getInstance();
      engine.getRegisteredConverters().add(new
JacksonConverter4GingaSocial());

       Router router = new Router(getContext());

       router.attach("/user", UserServerResource.class);
       router.attach("/login", LoginServerResource.class);
       router.attach("/home", HomeServerResource.class);
       router.attach("/like", LikeServerResource.class);
       router.attach("/share", ShareServerResource.class);

       return router;
   }

}
```

EncodeUtil.java

```
package tv.gingasocial.util;

import java.util.HashMap;
import java.util.Map;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class EncodeUtil {

	private static Map<String, String> mapChars = new
HashMap<String, String>();

	static {
		mapChars.put("À", "u00C0");
		mapChars.put("Á", "u00C1");
		mapChars.put("Â", "u00C2");
		mapChars.put("Ã", "u00C3");
		mapChars.put("Ä", "u00C4");
		mapChars.put("Å", "u00C5");
		mapChars.put("Æ", "u00C6");
		mapChars.put("Ç", "u00C7");
		mapChars.put("È", "u00C8");
		mapChars.put("É", "u00C9");
		mapChars.put("Ê", "u00CA");
		mapChars.put("Ë", "u00CB");
		mapChars.put("Ì", "u00CC");
		mapChars.put("Í", "u00CD");
		mapChars.put("Î", "u00CE");
		mapChars.put("Ï", "u00CF");
		mapChars.put("Ð", "u00D0");
		mapChars.put("Ñ", "u00D1");
		mapChars.put("Ò", "u00D2");
		mapChars.put("Ó", "u00D3");
		mapChars.put("Ô", "u00D4");
		mapChars.put("Õ", "u00D5");
		mapChars.put("Ö", "u00D6");
		mapChars.put("×", "u00D7");
		mapChars.put("Ø", "u00D8");
		mapChars.put("Ù", "u00D9");
		mapChars.put("Ú", "u00DA");
		mapChars.put("Û", "u00DB");
		mapChars.put("Ü", "u00DC");
		mapChars.put("Ý", "u00DD");
		mapChars.put("Þ", "u00DE");
		mapChars.put("ß", "u00DF");
		mapChars.put("à", "u00E0");
```

```
        mapChars.put("á", "u00E1");
        mapChars.put("â", "u00E2");
        mapChars.put("ã", "u00E3");
        mapChars.put("ä", "u00E4");
        mapChars.put("å", "u00E5");
        mapChars.put("æ", "u00E6");
        mapChars.put("ç", "u00E7");
        mapChars.put("è", "u00E8");
        mapChars.put("é", "u00E9");
        mapChars.put("ê", "u00EA");
        mapChars.put("ë", "u00EB");
        mapChars.put("ì", "u00EC");
        mapChars.put("í", "u00ED");
        mapChars.put("î", "u00EE");
        mapChars.put("ï", "u00EF");
        mapChars.put("ð", "u00F0");
        mapChars.put("ñ", "u00F1");
        mapChars.put("ò", "u00F2");
        mapChars.put("ó", "u00F3");
        mapChars.put("ô", "u00F4");
        mapChars.put("õ", "u00F5");
        mapChars.put("ö", "u00F6");
        mapChars.put("÷", "u00F7");
        mapChars.put("ø", "u00F8");
        mapChars.put("ù", "u00F9");
        mapChars.put("ú", "u00FA");
        mapChars.put("û", "u00FB");
        mapChars.put("ü", "u00FC");
        mapChars.put("ý", "u00FD");
        mapChars.put("þ", "u00FE");
        mapChars.put("ÿ", "u00FF");
        mapChars.put("\n", "u00GS");
        mapChars.put("\r", "u00GS");
        mapChars.put("\t", "u00GF");
    }

    public static String transformTextPlainToUTF8(String str) {
        char[] charArray = str.toCharArray();

        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < charArray.length; i++) {
            String s = String.valueOf(charArray[i]);
            if(s.equals("\\") && i < charArray.length) {
                s += String.valueOf(charArray[i+1]);
            }

            if(mapChars.containsKey(s)) {
                s = mapChars.get(s);
            }
            sb.append(s);
        }
```

```
            return sb.toString();
      }

}
```

hibernate.cfg

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD//EN" "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">

<hibernate-configuration>
      <session-factory>
            <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</proper
ty>
            <property
name="hibernate.connection.url">jdbc:mysql://localhost/gingasocial?aut
oReconnect=true</property>
            <property
name="hibernate.connection.username">root</property>
            <property
name="hibernate.connection.password">root</property>
            <property
name="hibernate.connection.pool_size">10</property>
            <property name="show_sql">true</property>
            <property name="format_sql">true</property>
            <property name="hbm2ddl.auto">update</property>
            <property
name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
            <property
name="hibernate.current_session_context_class">thread</property>

            <mapping class="tv.gingasocial.model.entity.GSUser" />

      </session-factory>
</hibernate-configuration>
```

index.jsp

```
<%@page
import="tv.gingasocial.controller.facebook.FacebookController"%>
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html lang="pt-br">
 <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
```

```
      <title>Ginga Social</title>
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <meta name="description" content="">
   <meta name="author" content="">

   <!-- Le styles -->
   <link href="css/bootstrap.css" rel="stylesheet">
   <style type="text/css">
            body {
            padding-top: 60px;
            padding-bottom: 40px;
      }

            .img-facebook {
            width: 22px;
            margin: -3px 8px 0 -5px;
      }

      #auth-username {
            font-weight: bold;
            }

            .info-label {
                  font-weight: bold;
            }
   </style>
   <link href="css/bootstrap-responsive.css" rel="stylesheet">

   <!-- Le HTML5 shim, for IE6-8 support of HTML5 elements -->
   <!--[if lt IE 9]>
     <script
src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
   <![endif]-->
      <script src="js/jquery-1.8.2.min.js"></script>
 </head>

 <body>
      <div id="fb-root"></div>
   <script>
            // Load the SDK Asynchronously
      (function(d){
               var js, id = 'facebook-jssdk', ref =
d.getElementsByTagName('script')[0];
               if (d.getElementById(id)) {return;}
               js = d.createElement('script'); js.id = id; js.async =
true;
               js.src = "//connect.facebook.net/en_US/all.js";
               ref.parentNode.insertBefore(js, ref);
            }(document));
```

```
            // Init the SDK upon load
            window.fbAsyncInit = function() {
                  FB.init({
                    appId      : '<%=FacebookController.getAppId()%>',
// App ID
                    channelUrl :
'//'+window.location.hostname+'/channel', // Path to your Channel File
                    status     : true, // check login status
                    cookie     : true, // enable cookies to allow the
server to access the session
                    xfbml      : true  // parse XFBML
                });

                  // check the login status of the user
            FB.getLoginStatus(function(response) {
                        if(response.status === 'connected') {
                              doLogin();
                        } else {
                              $("#auth-button").html("Conectar
Facebook - GingaSocial");
                        $("#auth-loginlink").removeClass("disabled");
                        }
                  });

            // respond to clicks on the login button
            $(function() {
                  $('#auth-loginlink').click(function() {
                        doLogin();
                  });
            });
      }; // end Facebook SDK init

      function doLogin() {
            FB.login(function(response) {
                        if(response.authResponse) {
                              //getting user's info
                              var uid = response.authResponse.userID;
                        var accessToken =
response.authResponse.accessToken;

                        //registering new user or updating its
access_token
                        $.get("login.jsp", { access_token :
accessToken, fb_user_id : uid }, function(response) {
                              $("#auth-
loginlink").addClass("disabled");
                              var parsed_JSON =
jQuery.parseJSON(response);
                              $("#info-login").html(parsed_JSON.id);
                              $("#info-
pass").html(parsed_JSON.password);
```

```
                                //display the user name
                                FB.api('/me', function(me) {
                                            if(me.name) {
                                            $("#auth-
username").html(me.name);
                                            $(".button-
loggedout").fadeOut(function() {
                                                  $(".button-
loggedin").fadeIn();
                                            });
                                            }
                                      });
                                console.log(">>>" + response);
                          });
                          }
                  }, {scope:
'user_about_me,user_birthday,user_interests,user_likes,email,publish_s
tream,publish_actions,read_stream'});
      }
   </script>
   <div class="navbar navbar-inverse navbar-fixed-top">
     <div class="navbar-inner">
       <div class="container">
         <a class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
           <span class="icon-bar"></span>
           <span class="icon-bar"></span>
           <span class="icon-bar"></span>
         </a>
         <a class="brand" href="#">Ginga Social TV</a>
         <div class="nav-collapse collapse">
           <ul class="nav">
             <li class="active"><a href="#">Home</a></li>
             <li><a href="#about">About</a></li>
             <li><a href="#contact">Contact</a></li>
           </ul>
         </div><!--/.nav-collapse -->
       </div>
     </div>
   </div>

   <div class="container">

     <!-- Main hero unit for a primary marketing message or call to
action -->
     <div class="hero-unit">
       <h1>Ginga Social TV!</h1>
       <p>Plataforma interativa para TVs Digitais com a tecnologia
Ginga&reg;</p>
```

```
       <p>Use sua conta do Facebook para conectar-se ao Ginga
Social.</p>
       <p class="button-loggedout">
            <a id="auth-loginlink" class="btn btn-primary btn-large
disabled">
                  <img src="images/social_facebook_box_white.png"
class="img-facebook" /><span id="auth-button">Conectando ao
Facebook...</span>
            </a>
       </p>
       <div class="well button-loggedin" style="display: none; width:
480px;">
            Olá, <span id="auth-username"></span>
            <br/><br/>
            <div>
                  Acesse o Ginga Social em sua TV com os dados abaixo:
                  <br/>
                  <i class="icon-user"></i>  <span class="info-
label">Login:</span> <span id="info-login"></span><br/>
                  <i class="icon-asterisk"></i> <span class="info-
label">Senha:</span> <span id="info-pass"></span><br/>
            </div>
       </div>
     </div>

     <hr>

     <footer>
       <p>&amp;copy; Rupp &amp;amp; Castro Corporation 2012</p>
     </footer>

   </div>

 </body>
</html>
```

login.jsp

```
<%@page import="org.codehaus.jackson.map.ObjectMapper"%>
<%@page import="tv.gingasocial.model.entity.GSUser"%>
<%@page import="tv.gingasocial.controller.Controller"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%
      String accessToken = request.getParameter("access_token");
      String facebookUserId = request.getParameter("fb_user_id");

      GSUser gingaUser =
Controller.registerOrUpdateUser(facebookUserId, accessToken);
      ObjectMapper mapper = new ObjectMapper(); // can reuse, share
globally
```

```
      String json = mapper.writeValueAsString(gingaUser);

%>
<%=json%>
```

ANEXO C - Detalhes de implementação do Ginga Social TV (cliente)

Control.java

```
package gingasocial;

import gingasocial.model.User;
import gingasocial.view.GSStyle;
import gingasocial.view.HomeIconGS;
import gingasocial.view.LoginGS;
import gingasocial.view.UserHome;

import javax.tv.xlet.XletContext;

import com.sun.dtv.lwuit.Form;

public class Control {

  private XletContext context;
  private Form currentForm;

  public Control(XletContext context) {
       this.context = context;
       GSStyle.initGSStyleComponents();
  }

  public void start() {
       currentForm = new HomeIconGS(this);
       ((HomeIconGS) currentForm).init();
  }

  public void showMainForm() {
       currentForm = new LoginGS(this);

       ((LoginGS) currentForm).loadImage();
       ((LoginGS) currentForm).init();
  }

  public void showUserHome(User user) {
       currentForm = new UserHome(this, user);
       ((UserHome) currentForm).loadImage();
       ((UserHome) currentForm).init();
  }

}
```

GingaSocialXlet.java

```
package gingasocial;

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

```
import javax.tv.xlet.XletStateChangeException;

import com.sun.dtv.lwuit.plaf.UIManager;

public class GingaSocialXlet implements Xlet {

  private Control controle;

  public void initXlet(XletContext context) throws
XletStateChangeException {
       controle = new Control(context);
  }

  public void startXlet() throws XletStateChangeException {
       controle.start();
  }

  public void destroyXlet(boolean arg0) throws
XletStateChangeException { }
  public void pauseXlet() { }

}
```

Post.java

```
package gingasocial.model;

import gingasocial.util.EncodeUtil;
import gingasocial.util.ResourceComponents;

import java.util.Vector;

/**
* @author Gustavo Rupp Santos
* @author Fiodor Castro
*/
public class Post {

  private String facebookId;
  private String userFacebookId;
  private String userName;
  private String userAvatarURL;
  private String message;
  private String type;
  private String photoURL;
  private String updatedTime;
  private long likes;
  private long comments;

  public Post(String facebookId, String userFacebookId, String
userName,
```

```
            String userAvatarURL, String message, String type, String
photoURL,
            String updatedTime, long likes, long comments) {
        setFacebookId(facebookId);
        setUserFacebookId(userFacebookId);
        setUserName(EncodeUtil.changeEncodeToUTF8(userName));
        setUserAvatarURL(userAvatarURL);
        ResourceComponents.loadImageByURL(userFacebookId,
userAvatarURL);
        setMessage(EncodeUtil.changeEncodeToUTF8(message));
        setType(type);
        setPhotoURL(photoURL);
        setUpdatedTime(updatedTime);
        setLikes(likes);
        setComments(comments);
    }

    public String getFacebookId() {
        return facebookId;
    }

    public void setFacebookId(String facebookId) {
        this.facebookId = facebookId;
    }

    public String getUserFacebookId() {
        return userFacebookId;
    }

    public void setUserFacebookId(String userFacebookId) {
        this.userFacebookId = userFacebookId;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserAvatarURL() {
        return userAvatarURL;
    }

    public void setUserAvatarURL(String userAvatarURL) {
        this.userAvatarURL = userAvatarURL;
    }

    public String getMessage() {
        return message;
```

```
  }

  public void setMessage(String message) {
       this.message = message;
  }

  public String getType() {
       return type;
  }

  public void setType(String type) {
       this.type = type;
  }

  public String getUpdatedTime() {
       return updatedTime;
  }

  public void setUpdatedTime(String updatedTime) {
       this.updatedTime = updatedTime;
  }

  public long getLikes() {
       return likes;
  }

  public void setLikes(long likes) {
       this.likes = likes;
  }

  public long getComments() {
       return comments;
  }

  public void setComments(long comments) {
       this.comments = comments;
  }

  public String getPhotoURL() {
       return photoURL;
  }

  public void setPhotoURL(String photoURL) {

       if (!photoURL.equals("null")) {
             String temp = photoURL.substring(photoURL.length() - 5);
             temp = temp.replace('s', 'q');
             photoURL = photoURL.substring(0, photoURL.length() - 5) +
temp;
       }
       this.photoURL = photoURL;
```

```
  }

  public String toString() {
       return "HomeDTO [facebookId=" + facebookId + ",
userFacebookId="
                    + userFacebookId + ", userName=" + userName
                    + ", userAvatarURL=" + userAvatarURL + ", message="
+ message
                    + ", type=" + type + ", photoURL=" + photoURL
                    + ", updatedTime=" + updatedTime + ", likes=" +
likes
                    + ", comments=" + comments + "]";
  }

  public static Vector parseHomeDTO(String json) {
       Vector vector = new Vector();
       int pointer = 0;
       do {
             int indexStart = json.indexOf("|'", pointer) + 2;
             int indexEnd = json.indexOf("','", pointer);
             String[] arr = new String[10];

             for (int i = 0; i <= 9; i++) {
                   String substring = json.substring(indexStart,
indexEnd);
                   arr[i] = substring;

                   indexStart = indexEnd + 3;
                   if (i == 8) {
                         indexEnd = json.indexOf("'|", indexStart);
                   } else if (i != 9) {
                         indexEnd = json.indexOf("','", indexStart);
                   }
             }

             Post home = new Post(arr[0], arr[1], arr[2], arr[3],
arr[4],
                         arr[5], arr[6], arr[7],
Long.parseLong(arr[8]),
                         Long.parseLong(arr[9]));
             vector.add(home);
             int as = json.indexOf("|'", indexEnd);
             if (as < 0) {
                   break;
             }
             pointer = indexEnd - 1;
       } while (true);

       return vector;
  }
```

```
}
```

User.java

```
package gingasocial.model;

import gingasocial.util.ResourceComponents;

import java.io.UnsupportedEncodingException;
import java.util.Vector;

public class User {

  private String gingaId;
  private String pass;
  private String name;
  private String avatarURL;
  private String facebookUserId;
  private Vector home;

  public User(String login, String pass) {
       this.gingaId = login;
       this.pass = pass;
  }

  public void updateUserData(String facebookUserId, String name,
String avatarURL) {
       setName(name);
       setAvatarURL(avatarURL);
       setFacebookUserId(facebookUserId);
       ResourceComponents.loadImageByURL(facebookUserId, avatarURL);
  }

  public String getGingaId() {
       return gingaId;
  }

  public void setGingaId(String gingaId) {
       this.gingaId = gingaId;
  }

  public String getPass() {
       return pass;
  }

  public void setPass(String pass) {
       this.pass = pass;
  }

  public String getName() {
       return name;
```

```
  }

  public void setName(String name) {
       this.name = name;
  }

  public String getAvatarURL() {
       return avatarURL;
  }

  public void setAvatarURL(String avatarURL) {
       this.avatarURL = avatarURL;
  }

  public String getFacebookUserId() {
       return facebookUserId;
  }

  public void setFacebookUserId(String facebookUserId) {
       this.facebookUserId = facebookUserId;
  }

  public Vector getHome() {
       return home;
  }

  public void setHome(Vector home) {
       this.home = home;
  }

  public synchronized void updateHome(Vector home2Update, String
direction) {
       Vector vec = null;
       if(direction.equals("bottom")) {
             vec = new Vector(home);
             vec.addAll(home2Update);
       } else {
             vec = new Vector(home2Update);
             vec.addAll(home);
       }
       setHome(vec);
  }

      public static void parseUser(User user, String data) {
      int indexStart = data.indexOf("|'") + 2;
      int indexEnd = data.indexOf("','");
      String[] arr = new String[4];

      for (int i = 0; i <= 3; i++) {
            String substring = data.substring(indexStart, indexEnd);
            try {
```

```
                    arr[i] = new String(substring.getBytes(), "UTF-8");
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }

                indexStart = indexEnd + 3;
                if(i == 2) {
                indexEnd = data.indexOf("'|", indexStart);
                } else if(i != 3) {
                indexEnd = data.indexOf("','", indexStart);
                }
        }

        user.updateUserData(arr[1], arr[2], arr[3]);
        }

}
```

Connector.java

```
package gingasocial.net;

import gingasocial.model.Post;
import gingasocial.model.User;
import gingasocial.util.EncodeUtil;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.Vector;

public class Connector {

  private static final String URL_BASE_GS =
"http://www.gingasocial.tv/GingaSocialWeb/facebook/";
  private static final String BASE_LOGIN = URL_BASE_GS + "login";
  private static final String BASE_HOME = URL_BASE_GS + "home";
  private static final String BASE_USER = URL_BASE_GS + "user";
  private static final String BASE_LIKE = URL_BASE_GS + "like";
  private static final String BASE_SHARE = URL_BASE_GS + "share";

  public static boolean doLogin(User user) throws IOException{
       URL loginURL = new URL(BASE_LOGIN +
"?login="+user.getGingaId()+"&pass="+user.getPass());
       URLConnection connection = loginURL.openConnection();
       connection.connect();

       BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
```

```
        String result = in.readLine();
        System.out.println("[DEBUG] Return doLogin >" + result);

        return result.equals("true");
  }

  public static Vector getUserHome(User user, String direction) throws
IOException {
        String userIdParam = "?ginga_user_id="+user.getGingaId();
        String directionParam = EncodeUtil.isEmpty(direction) ? "" :
"&direction=" + direction;
        String quantItemsParam = "&quant_items=25";

        URL homeURL = new URL (BASE_HOME + userIdParam + directionParam
+ quantItemsParam);
        URLConnection connection = homeURL.openConnection();
        BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        String result="";
        String line = "";

       while ((line = in.readLine()) != null) {
              result=result+line;
       }

        return Post.parseHomeDTO(result);
  }

  public static void getUserInfo(User user) throws IOException {
        URL loginURL = new URL(BASE_USER +
"?ginga_user_id="+user.getGingaId());
        URLConnection connection = loginURL.openConnection();
        connection.connect();

        BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        String result = in.readLine();

        User.parseUser(user, result);
  }

  public static boolean like(String ginga_user_id, String facebook_id)
throws IOException{
        URL loginURL = new
URL(BASE_LIKE+"?ginga_user_id="+ginga_user_id+"&facebook_id="+facebook
_id);
        URLConnection connection = loginURL.openConnection();
        connection.connect();

        BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
```

```
        String result = in.readLine();

        return result.equals("true");
  }

  public static boolean share(String ginga_user_id, String
facebook_id)throws IOException{

        URL loginURL = new
URL(BASE_SHARE+"?ginga_user_id="+ginga_user_id+"&facebook_id="+faceboo
k_id);
        URLConnection connection = loginURL.openConnection();
        connection.connect();
        BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        String result = in.readLine();

        return result.equals("true");
  }

  public static InputStream loadImage(String URLImage) throws
IOException
  {
        URL loginURL = new URL(URLImage);
        InputStream inputStream = loginURL.openStream();
        return inputStream;
  }
}
```

DTVUtil.java

```
package gingasocial.util;

import com.sun.dtv.lwuit.geom.Dimension;
import com.sun.dtv.ui.DTVContainer;
import com.sun.dtv.ui.Screen;
import com.sun.dtv.ui.event.KeyEvent;
import com.sun.dtv.ui.event.RemoteControlEvent;
import com.sun.dtv.ui.event.UserInputEventListener;
import com.sun.dtv.ui.event.UserInputEventManager;

public class DTVUtil {

  private static Screen currentScreen = Screen.getCurrentScreen();
  private static UserInputEventManager inputManager =
UserInputEventManager.getUserInputEventManager(currentScreen);

  public static Dimension getScreenDimension() {
        DTVContainer container = DTVContainer.getCurrentDTVContainer();
        return container.getBounds().getSize();
  }
```

```
  private static boolean numericKeysTaken = false;
  private static boolean coloredKeys = false;
  private static boolean backKey = false;

  public static void reserveNumericKeys(UserInputEventListener
listener) {
       if (numericKeysTaken) {
             System.out.println("Teclas numericas sendo utilizadas");
             return;
       }

       KeyEvent event_0 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_0, '0');
       KeyEvent event_1 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_1, '0');
       KeyEvent event_2 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_2, '0');
       KeyEvent event_3 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_3, '0');
       KeyEvent event_4 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_4, '0');
       KeyEvent event_5 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_5, '0');
       KeyEvent event_6 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_6, '0');
       KeyEvent event_7 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_7, '0');
       KeyEvent event_8 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_8, '0');
       KeyEvent event_9 = new KeyEvent(null, KeyEvent.KEY_PRESSED, 0,
0, KeyEvent.VK_9, '0');

       inputManager.addUserInputEventListener(listener, event_0);
       inputManager.addUserInputEventListener(listener, event_1);
       inputManager.addUserInputEventListener(listener, event_2);
       inputManager.addUserInputEventListener(listener, event_3);
       inputManager.addUserInputEventListener(listener, event_4);
       inputManager.addUserInputEventListener(listener, event_5);
       inputManager.addUserInputEventListener(listener, event_6);
       inputManager.addUserInputEventListener(listener, event_7);
       inputManager.addUserInputEventListener(listener, event_8);
       inputManager.addUserInputEventListener(listener, event_9);

       numericKeysTaken = true;

  }

  public static void reserveColoredKeys(UserInputEventListener
listener) {
       if (coloredKeys) {
```

```
            System.out.println("Teclas sendo utilizadas" +
listener.getClass());
            return;
      }
      RemoteControlEvent redKey = new RemoteControlEvent(null,
KeyEvent.KEY_PRESSED, 0, 0, RemoteControlEvent.VK_COLORED_KEY_0,
RemoteControlEvent.CHAR_UNDEFINED);
      RemoteControlEvent greenKey = new RemoteControlEvent(null,
KeyEvent.KEY_PRESSED, 0, 0, RemoteControlEvent.VK_COLORED_KEY_1,
RemoteControlEvent.CHAR_UNDEFINED);
      RemoteControlEvent yellowKey = new RemoteControlEvent(null,
KeyEvent.KEY_PRESSED, 0, 0, RemoteControlEvent.VK_COLORED_KEY_2,
RemoteControlEvent.CHAR_UNDEFINED);
      RemoteControlEvent blueKey = new RemoteControlEvent(null,
KeyEvent.KEY_PRESSED, 0, 0, RemoteControlEvent.VK_COLORED_KEY_3,
RemoteControlEvent.CHAR_UNDEFINED);

      inputManager.addUserInputEventListener(listener, redKey);
      inputManager.addUserInputEventListener(listener, greenKey);
      inputManager.addUserInputEventListener(listener, yellowKey);
      inputManager.addUserInputEventListener(listener, blueKey);

      coloredKeys = true;
  }

  public static void reserveBackKey(UserInputEventListener listener) {
      if (backKey) {
            System.out.println("Tecla BACK sendo utilizada");
            return;
      }
      RemoteControlEvent back = new RemoteControlEvent(null,
KeyEvent.KEY_PRESSED, 0, 0, RemoteControlEvent.VK_BACK,
RemoteControlEvent.CHAR_UNDEFINED);
      inputManager.addUserInputEventListener(listener, back);
      backKey = true;
  }

  public static void releaseNumericKeys(UserInputEventListener
lstener) {
      if (!numericKeysTaken) {
            System.out.println("Nao possui as teclas");
            return;
      }
      inputManager.removeUserInputEventListener(lstener);
      numericKeysTaken = false;
  }

  public static void releaseColoredKeys(UserInputEventListener
listener) {
      if (!coloredKeys) {
            System.out.println("Nao possui as teclas");
```

```
            return;
      }
      inputManager.removeUserInputEventListener(listener);
      coloredKeys = false;
  }

  public static void releaseBackKey(UserInputEventListener listener) {
      if (!backKey) {
            System.out.println("Nao possui a tecla back");
            return;
      }
      inputManager.removeUserInputEventListener(listener);
      backKey = false;
  }

}
```

EncodeUtil.java

```
package gingasocial.util;

import java.util.HashMap;
import java.util.Map;

public class EncodeUtil {

  private static Map mapChars = new HashMap();
  static {
      mapChars.put("u00C0", "\u00C0");
      mapChars.put("u00C1", "\u00C1");
      mapChars.put("u00C2", "\u00C2");
      mapChars.put("u00C3", "\u00C3");
      mapChars.put("u00C4", "\u00C4");
      mapChars.put("u00C5", "\u00C5");
      mapChars.put("u00C6", "\u00C6");
      mapChars.put("u00C7", "\u00C7");
      mapChars.put("u00C8", "\u00C8");
      mapChars.put("u00C9", "\u00C9");
      mapChars.put("u00CA", "\u00CA");
      mapChars.put("u00CB", "\u00CB");
      mapChars.put("u00CC", "\u00CC");
      mapChars.put("u00CD", "\u00CD");
      mapChars.put("u00CE", "\u00CE");
      mapChars.put("u00CF", "\u00CF");
      mapChars.put("u00D0", "\u00D0");
      mapChars.put("u00D1", "\u00D1");
      mapChars.put("u00D2", "\u00D2");
      mapChars.put("u00D3", "\u00D3");
      mapChars.put("u00D4", "\u00D4");
      mapChars.put("u00D5", "\u00D5");
      mapChars.put("u00D6", "\u00D6");
```

```
        mapChars.put("u00D7", "\u00D7");
        mapChars.put("u00D8", "\u00D8");
        mapChars.put("u00D9", "\u00D9");
        mapChars.put("u00DA", "\u00DA");
        mapChars.put("u00DB", "\u00DB");
        mapChars.put("u00DC", "\u00DC");
        mapChars.put("u00DD", "\u00DD");
        mapChars.put("u00DE", "\u00DE");
        mapChars.put("u00DF", "\u00DF");
        mapChars.put("u00E0", "\u00E0");
        mapChars.put("u00E1", "\u00E1");
        mapChars.put("u00E2", "\u00E2");
        mapChars.put("u00E3", "\u00E3");
        mapChars.put("u00E4", "\u00E4");
        mapChars.put("u00E5", "\u00E5");
        mapChars.put("u00E6", "\u00E6");
        mapChars.put("u00E7", "\u00E7");
        mapChars.put("u00E8", "\u00E8");
        mapChars.put("u00E9", "\u00E9");
        mapChars.put("u00EA", "\u00EA");
        mapChars.put("u00EB", "\u00EB");
        mapChars.put("u00EC", "\u00EC");
        mapChars.put("u00ED", "\u00ED");
        mapChars.put("u00EE", "\u00EE");
        mapChars.put("u00EF", "\u00EF");
        mapChars.put("u00F0", "\u00F0");
        mapChars.put("u00F1", "\u00F1");
        mapChars.put("u00F2", "\u00F2");
        mapChars.put("u00F3", "\u00F3");
        mapChars.put("u00F4", "\u00F4");
        mapChars.put("u00F5", "\u00F5");
        mapChars.put("u00F6", "\u00F6");
        mapChars.put("u00F7", "\u00F7");
        mapChars.put("u00F8", "\u00F8");
        mapChars.put("u00F9", "\u00F9");
        mapChars.put("u00FA", "\u00FA");
        mapChars.put("u00FB", "\u00FB");
        mapChars.put("u00FC", "\u00FC");
        mapChars.put("u00FD", "\u00FD");
        mapChars.put("u00FE", "\u00FE");
        mapChars.put("u00FF", "\u00FF");
        mapChars.put("u00GS", "\n");
        mapChars.put("u00GF", "\t");
    }

    public static final char a_ACUTE = '\u00E1';

    public static String changeEncodeToUTF8(String str) {
        int indexOf = str.indexOf("u00");

        while(indexOf != -1) {
```

```
            String firstPart = str.substring(0, indexOf);
            String unicodeChar = str.substring(indexOf, indexOf + 5);
            String lastPart = str.substring(indexOf + 5,
str.length());

            str = firstPart + mapChars.get(unicodeChar) + lastPart;

            indexOf = str.indexOf("u00");
      }

      return str;
  }

  public static boolean isEmpty(String str) {
      return str == null || str.equals("");
  }
}
```

ResourceComponentes.java

```
package gingasocial.util;

import gingasocial.net.Connector;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import com.sun.dtv.lwuit.Image;

public class ResourceComponents {

  private static Map resources = new HashMap();

  public static Image getImage(String pathImage) {
      Image image = null;
      Object o = resources.get(pathImage);
      if (o instanceof Image) {
            image = (Image) o;
      } else if (o == null) {
            try {
                  image = Image.createImage(pathImage);
                  if (image != null) {
                        resources.put(pathImage, image);
                  }
            } catch (IOException ioe) {
                  ioe.printStackTrace();
            }
      }
```

```
        return image;
    }

    public static void release(String pathImage) {
        Object o = resources.get(pathImage);
        if(o instanceof Image) {
            Image img = (Image) o;
            img.getAWTImage().flush();
        }
        resources.remove(pathImage);
    }

    public static void loadImageByURL(String id, String avatarURL){
        if(!avatarURL.equals("null")) {
            Image image;
            try {
                image =
Image.createImage(Connector.loadImage(avatarURL));

                if (image != null) {
                    resources.put(id, image);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}
```

DialogGS.java

```
package gingasocial.view;

import gingasocial.model.Post;
import gingasocial.util.DTVUtil;
import gingasocial.util.EncodeUtil;
import gingasocial.util.ResourceComponents;

import com.sun.dtv.lwuit.Dialog;
import com.sun.dtv.lwuit.Image;
import com.sun.dtv.lwuit.Label;
import com.sun.dtv.lwuit.layouts.BorderLayout;
import com.sun.dtv.ui.event.KeyEvent;
import com.sun.dtv.ui.event.RemoteControlEvent;
import com.sun.dtv.ui.event.UserInputEvent;
import com.sun.dtv.ui.event.UserInputEventListener;

public class DialogGS extends Dialog implements UserInputEventListener
{
```

```
  private Post post;
  private UserHome home;
  private Label likeCommentLabel;
  private static final String ACTION_ICON =
"gingasocial/images/botoes_expand.png";
  private Image actions;

  public DialogGS(Post post, UserHome home) {
       this.post = post;
       this.home = home;
       actions = ResourceComponents.getImage(ACTION_ICON);
       DTVUtil.reserveBackKey(this);
  }

  public void init() {

       setLayout(new BorderLayout());
       Label avatarLabel = new Label();
       likeCommentLabel = new Label();
       GingaSocialPostContainer rightContainer = new
GingaSocialPostContainer();

       avatarLabel.setWidth(40);
       rightContainer.setWidth(50);
       addComponent(BorderLayout.WEST, avatarLabel);
       addComponent(BorderLayout.CENTER, rightContainer);
       ResourceComponents.loadImageByURL(post.getFacebookId(),
post.getPhotoURL());

avatarLabel.setIcon(ResourceComponents.getImage(post.getUserFacebookId
()));
       avatarLabel.getStyle().setMargin(1, 1, 1, 0);
       String msg = post.getMessage() != null ? post.getMessage() :
"";
       rightContainer.getNorthTextAreaPost().setFocusable(false);
       rightContainer.getNorthTextAreaPost().setText(msg);
       rightContainer.getNorthTextAreaPost().setFocusable(true);
       rightContainer.getNorthTextAreaPost().setWidth(100);

       if (post.getType().equals("photo")) {
            Label centerImage = new
Label(ResourceComponents.getImage(post.getFacebookId()));
            rightContainer.setCenterLabelImagePost(centerImage);
            rightContainer.setWidth(200);
            rightContainer.setHeight(200);
       }

likeCommentLabel.setIcon(ResourceComponents.getImage(GingaSocialModern
ListCellRenderer.IMAGE_LIKE_COMMENT));
```

```
       likeCommentLabel.setText(post.getLikes() + " pessoas curtiram
isso\n" + post.getComments() + " coment" + EncodeUtil.a_ACUTE +
"rios");
       likeCommentLabel.getStyle().setMargin(1, 0, 2, 1);

likeCommentLabel.getStyle().setBgColor(GSStyle.LIKE_COMMENT_CELL);

rightContainer.setSouthContainerLikeCommentPost(likeCommentLabel);

       Label acoesUsuario = new Label(actions);
       getStyle().setBgTransparency(0);
       getStyle().setBgColor(GSStyle.BLUE_FB);
       getStyle().setBgSelectionColor(GSStyle.BLUE_FB);
       getStyle().setMargin(2, 2, 2, 2);
       getStyle().setPadding(3, 3, 3, 3);

       addComponent(BorderLayout.SOUTH, acoesUsuario);

  }

  public void userInputEventReceived(UserInputEvent o) {
       KeyEvent event = (KeyEvent) o;

       int code = event.getKeyCode();
       if (code == RemoteControlEvent.VK_COLORED_KEY_1) {// verde -
curtir
             home.curtirPost();
             likeCommentLabel.setText((post.getLikes()) + " pessoas
curtiram isso\n" + post.getComments() + " coment" + EncodeUtil.a_ACUTE
+ "rios");
       }
       if (code == RemoteControlEvent.VK_COLORED_KEY_2) {// amarelo -
compartilhar
             home.compartilharPost();
       }

       if (code == RemoteControlEvent.VK_BACK) {
             DTVUtil.releaseColoredKeys(DialogGS.this);
             DTVUtil.releaseNumericKeys(DialogGS.this);
             DTVUtil.releaseBackKey(this);
             DTVUtil.reserveColoredKeys(home);
             DTVUtil.reserveNumericKeys(home);
             dispose();
       }
  }

}
```

GingaSocialModernListCellRenderer.java

```
package gingasocial.view;
```

```
import gingasocial.model.Post;
import gingasocial.util.EncodeUtil;
import gingasocial.util.ResourceComponents;

import java.awt.Color;

import com.sun.dtv.lwuit.Component;
import com.sun.dtv.lwuit.Container;
import com.sun.dtv.lwuit.Label;
import com.sun.dtv.lwuit.List;
import com.sun.dtv.lwuit.geom.Dimension;
import com.sun.dtv.lwuit.layouts.BorderLayout;
import com.sun.dtv.lwuit.list.ListCellRenderer;
import com.sun.dtv.lwuit.plaf.Style;

public class GingaSocialModernListCellRenderer extends Container
implements ListCellRenderer {

  private UserHome userHome;

  protected static final String IMAGE_LIKE_COMMENT =
"gingasocial/images/like_comment_blue.png";

  protected static final String IMAGE_LIKE =
"gingasocial/images/like_blue.png";
  protected static final String IMAGE_COMMENT =
"gingasocial/images/comment_blue.png";
  protected static final String IMAGE_SHARE =
"gingasocial/images/share_blue.png";

  public GingaSocialModernListCellRenderer(UserHome userHome) {
        super();
        this.userHome = userHome;
        setUIID("GingaSocialModernListCellRenderer");
        setCellRenderer(true);
        setSize(new Dimension(20, 400));
        setLayout(new BorderLayout());

        getStyle().setBgColor(Color.WHITE);
  }

  public Component getListCellRendererComponent(List list, Object
homeDTO, int paramInt, boolean isSelected) {
       Post home = (Post)homeDTO;
       Style style = this.getStyle();

       Label avatarLabel = new Label();
       Label likeCommentLabel = new Label();
       GingaSocialPostContainer rightContainer = new
GingaSocialPostContainer();
```

```
        avatarLabel.setWidth(40);
        rightContainer.setWidth(200);

        addComponent(BorderLayout.WEST, avatarLabel);
        addComponent(BorderLayout.CENTER, rightContainer);

avatarLabel.setIcon(ResourceComponents.getImage(home.getUserFacebookId
()));
        avatarLabel.getStyle().setMargin(1, 1, 1, 0); //south, north,
west, east
        String msg = home.getMessage() != null ? home.getMessage() :
"";
        rightContainer.getNorthTextAreaPost().setText(msg);

        setFocus(isSelected);
        avatarLabel.setFocus(isSelected);
        rightContainer.setFocus(isSelected);
        rightContainer.getNorthTextAreaPost().setFocus(isSelected);

likeCommentLabel.setIcon(ResourceComponents.getImage(IMAGE_LIKE_COMMEN
T));
        likeCommentLabel.setText(home.getLikes() + " pessoas curtiram
isso\n" + home.getComments() + " coment" + EncodeUtil.a_ACUTE +
"rios");
        likeCommentLabel.getStyle().setMargin(1, 0, 2, 1);

likeCommentLabel.getStyle().setBgColor(GSStyle.LIKE_COMMENT_CELL);

rightContainer.setSouthContainerLikeCommentPost(likeCommentLabel);

        if(isSelected) {
              style.setBgTransparency(255);
              style.setBgSelectionColor(GSStyle.BLUE_FB);
              if(home.getFacebookId() ==
userHome.getBottomItem().getFacebookId()) {
                    userHome.updateBottomCentralPane();
              }
        } else {
              setFocus(false);
              style.setBgTransparency(255);
              style.setBgColor(GSStyle.LINE_BORDER);
        }

        return this;
  }

  public Component getListFocusComponent(List list) {
        Container container = new Container();
        container.getStyle().setBgTransparency(100);
        return container;
```

```
  }

}
```

GingaSocialPostContainer.java

```
package gingasocial.view;

import com.sun.dtv.lwuit.Container;
import com.sun.dtv.lwuit.Label;
import com.sun.dtv.lwuit.TextArea;
import com.sun.dtv.lwuit.layouts.BorderLayout;

public class GingaSocialPostContainer extends Container {

  private TextArea northTextAreaPost;
  private Label centerLabelImagePost;
  private Label southContainerLikeCommentPost;

  public GingaSocialPostContainer() {
       setLayout(new BorderLayout());
       northTextAreaPost = new TextArea(2, 30);
       southContainerLikeCommentPost = new Label();
       northTextAreaPost.getStyle().setMargin(1, 0, 0, 1);
       addComponent(BorderLayout.NORTH, northTextAreaPost);
  }

  public TextArea getNorthTextAreaPost() {
       return northTextAreaPost;
  }

  public void setNorthTextAreaPost(TextArea northTextAreaPost) {
       this.northTextAreaPost = northTextAreaPost;
  }

  public Label getCenterLabelImagePost() {
       return centerLabelImagePost;
  }

  public void setCenterLabelImagePost(Label centerLabelImagePost) {
       this.centerLabelImagePost = centerLabelImagePost;
       this.centerLabelImagePost.getStyle().setMargin(0, 0, 0, 1);
       addComponent(BorderLayout.CENTER, centerLabelImagePost);
  }

  public Label getSouthContainerLikeCommentPost() {
       return southContainerLikeCommentPost;
  }

  public void setSouthContainerLikeCommentPost(Label
southContainerLikeCommentPost) {
```

```
        this.southContainerLikeCommentPost =
southContainerLikeCommentPost;
        addComponent(BorderLayout.SOUTH,
southContainerLikeCommentPost);
  }

}
```

GSStyle.java

```
package gingasocial.view;

import java.awt.Color;

import com.sun.dtv.lwuit.plaf.Style;
import com.sun.dtv.lwuit.plaf.UIManager;

public class GSStyle {

  public static final Color BLUE_FB = new Color(59, 89, 152); // dark
blue
  public static final Color LIKE_COMMENT_CELL = new Color(237, 239,
244); //
  public static final Color USER_DATA_FONT = new Color(242, 244, 248);
  public static final Color LINE_BORDER = new Color(204, 204, 204);
  public static final Color SELECTED_ITEM = new Color(242, 244, 249);

  public static void initGSStyleComponents() {
        UIManager manager = UIManager.getInstance();

        manager.setComponentStyle("Label", getLabelStyle());
        manager.setComponentStyle("TextField",
getTextFieldAreaStyle());
        manager.setComponentStyle("TextArea", getTextFieldAreaStyle());
        manager.setComponentStyle("Dialog", getDialogSytle());
  }

  private static Style getLabelStyle() {
        Style labelStyle = new Style();

        labelStyle.setBgColor(Color.WHITE);
        labelStyle.setBgSelectionColor(SELECTED_ITEM);
        labelStyle.setFgColor(Color.BLACK);
        labelStyle.setBgTransparency(255);

        return labelStyle;
  }

  private static Style getTextFieldAreaStyle() {
        Style textStyle = new Style();
```

```
        textStyle.setBgColor(Color.WHITE);
        textStyle.setBgSelectionColor(SELECTED_ITEM);
        textStyle.setFgColor(Color.BLACK);
        textStyle.setBgTransparency(255);
        return textStyle;
  }

  public static Style getDialogSytle() {
        Style dialogStyle = new Style();
        dialogStyle.setBgColor(Color.WHITE);

        return dialogStyle;
  }

}
```

HomeIconGS.java

```
package gingasocial.view;

import gingasocial.Control;
import gingasocial.util.DTVUtil;
import gingasocial.util.ResourceComponents;

import com.sun.dtv.lwuit.Button;
import com.sun.dtv.lwuit.Form;
import com.sun.dtv.lwuit.Image;
import com.sun.dtv.lwuit.events.ActionEvent;
import com.sun.dtv.lwuit.events.ActionListener;
import com.sun.dtv.lwuit.geom.Dimension;
import com.sun.dtv.lwuit.layouts.CoordinateLayout;
import com.sun.dtv.ui.event.RemoteControlEvent;

public class HomeIconGS extends Form {

  private static final String IMAGE_HOME_ICON =
"gingasocial/images/face_icon.jpg";
  private Control control;
  private Image imageIcon;
  private Button homeButton;

  public HomeIconGS(Control control) {
        super();
        this.control = control;
        loadImage();
  }

  public void loadImage() {
        imageIcon = ResourceComponents.getImage(IMAGE_HOME_ICON);
  }
```

```
  public void init() {
        Dimension dimension = DTVUtil.getScreenDimension();

        setLayout(new CoordinateLayout(dimension.getWidth(),
dimension.getHeight()));
        getStyle().setBgTransparency(0);

        homeButton = new Button(imageIcon);
        homeButton.getStyle().setMargin(0, 0, 0, 0);
        homeButton.getStyle().setPadding(0, 0, 0, 0);
        homeButton.getStyle().setBorder(null);

        homeButton.setX((int) (dimension.getWidth() * 0.10F));
        homeButton.setY((int) (dimension.getHeight() * 0.1F));
        addComponent(homeButton);

        homeButton.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent inputEvent) {
                    control.showMainForm();
              }
        });

        addKeyListener(RemoteControlEvent.VK_ESCAPE, new
ActionListener() {
              public void actionPerformed(ActionEvent ae) {
                    close();
              }
        });

        show();
  }

  public void close() {
        setVisible(false);
        ResourceComponents.release(IMAGE_HOME_ICON);
  }

}
```

LoginGS.java

```
package gingasocial.view;

import gingasocial.Control;
import gingasocial.model.User;
import gingasocial.net.Connector;
import gingasocial.util.DTVUtil;
import gingasocial.util.ResourceComponents;

import java.awt.Color;
import java.io.IOException;
```

```
import com.sun.dtv.lwuit.Container;
import com.sun.dtv.lwuit.Form;
import com.sun.dtv.lwuit.Image;
import com.sun.dtv.lwuit.Label;
import com.sun.dtv.lwuit.TextField;
import com.sun.dtv.lwuit.events.ActionEvent;
import com.sun.dtv.lwuit.events.ActionListener;
import com.sun.dtv.lwuit.geom.Dimension;
import com.sun.dtv.lwuit.layouts.CoordinateLayout;
import com.sun.dtv.ui.event.KeyEvent;
import com.sun.dtv.ui.event.RemoteControlEvent;
import com.sun.dtv.ui.event.UserInputEvent;
import com.sun.dtv.ui.event.UserInputEventListener;

public class LoginGS extends Form implements UserInputEventListener {

  private static final String IMAGE_GS_ICON =
"gingasocial/images/screen_login.png";
  private static final String ERROR_ICON = "gingasocial/images/error-
icon.png";
  private static final String LOADING_ICON =
"gingasocial/images/loading_facebook.png";
  private static final String INVALID_USER_MSG = "Login ou senha
inv\u00E1lidos";
  private static final String EXPIRED_ACCESS_TOKEN_MSG =
"Autoriza\u00e7\u00e3o do Facebook expirada,\n" + "acesse
www.gingasocial.tv/GingaSocialWeb/index.jsp para atualizar seu
c\u00f3digo.";
  private static final String SUCCESS_LOGIN_MSG = "Login realizado com
succeso\n" + "Carregando home, aguarde...";

  private Control controle;
  private Container container;

  private Label labelLogin, labelSenha, labelGS, labelConectar,
labelLimpar, labelVoltar;
  private TextField fieldLogin, fieldSenha;
  private Image gsIcon, errorIcon, loadingIcon;
  private Label labelMsg;

  public LoginGS(Control controle) {
       this.controle = controle;
  }

  public void init() {
       Dimension screenDimension = DTVUtil.getScreenDimension();
       setLayout(new CoordinateLayout(screenDimension.getWidth(),
screenDimension.getHeight()));

       getStyle().setBgTransparency(0);
       getStyle().setMargin(0, 0, 0, 0);
```

```
        getStyle().setPadding(0, 0, 0, 0);

        container = new Container();

        Dimension dmContainer = new Dimension(448, 306); // 458
        container.setSize(dmContainer);
        container.setLayout(new CoordinateLayout(dmContainer));
        addComponent(container);

        container.setX((int) (screenDimension.getWidth() * 0.03F));
        container.setY((int) (screenDimension.getHeight() * 0.01F));
        container.getStyle().setBgTransparency(255);
        container.getStyle().setBgColor(GSStyle.BLUE_FB);
        container.getStyle().setMargin(0, 0, 0, 0);
        container.getStyle().setPadding(0, 0, 0, 0);

        initComponents();
        container.addComponent(labelGS);
        container.addComponent(fieldLogin);
        container.addComponent(fieldSenha);
        DTVUtil.reserveNumericKeys(this);
        DTVUtil.reserveColoredKeys(this);

        addKeyListener(RemoteControlEvent.VK_ESCAPE, new
ActionListener() {
              public void actionPerformed(ActionEvent ae) {
                    close();
              }
        });

        show();
  }

  public void initComponents() {
        labelGS = new Label(gsIcon);
        labelGS.getStyle().setMargin(0, 0, 0, 0);
        labelGS.getStyle().setPadding(0, 0, 0, 0);

        fieldLogin = new TextFieldGS();
        fieldLogin.setWidth(260);
        fieldLogin.setHeight(29);
        fieldLogin.setX(135);
        fieldLogin.setY(154);

        fieldSenha = new TextFieldGS();
        fieldSenha.setConstraint(TextField.PASSWORD);
        fieldSenha.setWidth(260);
        fieldSenha.setHeight(29);
        fieldSenha.setX(135);
        fieldSenha.setY(191);
  }
```

```
  public void loadImage() {
       gsIcon = ResourceComponents.getImage(IMAGE_GS_ICON);
       errorIcon = ResourceComponents.getImage(ERROR_ICON);
       loadingIcon = ResourceComponents.getImage(getLoadingIcon());
  }

  private class TextFieldGS extends TextField {
       public void keyPressed(int keyCode) {
            if (keyCode == RemoteControlEvent.VK_BACK) {
                 String tempTextFieldText = getText();
                 if (tempTextFieldText.length() > 0) {
                      setText(tempTextFieldText.substring(0,
tempTextFieldText.length() - 1));
                 }
            } else {
                 super.keyPressed(keyCode);
            }
       }
  }

  public void userInputEventReceived(UserInputEvent o) {
       KeyEvent event = (KeyEvent) o;
       int code = event.getKeyCode();

       if (code == RemoteControlEvent.VK_COLORED_KEY_1) {
            String login = fieldLogin.getText();
            String pass = fieldSenha.getText();
            if (login.equals("") || pass.equals("")) {
                 labelMsg = createLabelMsg(INVALID_USER_MSG,
Color.RED);
                 container.addComponent(labelMsg);
                 repaint();
                 return;
            }
            User user = new User(login, pass);

            try {
                 boolean canLogin = Connector.doLogin(user);
                 if (canLogin) {
                      labelMsg = createLabelMsg(SUCCESS_LOGIN_MSG,
new Color(11, 114, 7));
                      container.addComponent(labelMsg);
                      repaint();
                      Connector.getUserInfo(user); // all
attributes setted
                      user.setHome(Connector.getUserHome(user,
null));
                      close();
                      controle.showUserHome(user);
                 } else {
```

```
                        labelMsg = createLabelMsg(INVALID_USER_MSG,
Color.RED);
                        container.addComponent(labelMsg);
                        repaint();
                  }

            } catch (IOException e1) {
                  e1.printStackTrace();
            }
      } else {
            if (code == RemoteControlEvent.VK_COLORED_KEY_0) {
                  close();
            } else {
                  if (code == RemoteControlEvent.VK_COLORED_KEY_2) {
                        fieldLogin.setText("");
                        fieldSenha.setText("");
                  }
            }
      } // end else
  }

  public Label createLabelMsg(String msg, Color fontColor) {
      Label label = new Label(msg);
      label.getStyle().setBgColor(Color.WHITE);
      label.getStyle().setFgColor(fontColor);
      label.getStyle().setBgTransparency(255);
      label.setX(135);
      label.setY(260);
      label.setWidth(260);
      if (msg.equals(SUCCESS_LOGIN_MSG)) {
            label.setHeight(40);
            label.setIcon(loadingIcon);
      } else {
            label.setHeight(40);
            label.setIcon(errorIcon);
      }

      return label;
  }

  public void close() {
      setVisible(false);
      DTVUtil.releaseNumericKeys(this);
      DTVUtil.releaseColoredKeys(this);
      ResourceComponents.release(IMAGE_GS_ICON);
  }

  public static String getLoadingIcon() {
      return LOADING_ICON;
  }
```

```
}
```

UserHome.java

```
package gingasocial.view;

import gingasocial.Control;
import gingasocial.model.Post;
import gingasocial.model.User;
import gingasocial.net.Connector;
import gingasocial.util.DTVUtil;
import gingasocial.util.ResourceComponents;

import java.awt.Color;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;

import com.sun.dtv.lwuit.Container;
import com.sun.dtv.lwuit.Form;
import com.sun.dtv.lwuit.Image;
import com.sun.dtv.lwuit.Label;
import com.sun.dtv.lwuit.List;
import com.sun.dtv.lwuit.events.ActionEvent;
import com.sun.dtv.lwuit.events.ActionListener;
import com.sun.dtv.lwuit.geom.Dimension;
import com.sun.dtv.lwuit.layouts.BorderLayout;
import com.sun.dtv.lwuit.layouts.CoordinateLayout;
import com.sun.dtv.lwuit.plaf.Style;
import com.sun.dtv.ui.event.KeyEvent;
import com.sun.dtv.ui.event.RemoteControlEvent;
import com.sun.dtv.ui.event.UserInputEvent;
import com.sun.dtv.ui.event.UserInputEventListener;

public class UserHome extends Form implements UserInputEventListener {

  private static final String BUTTONS =
"gingasocial/images/buttons_base.png";

  private Control controle;
  private Container container, topPanel, centralPanel, southPanel;
  private User user;
  private List posts;
  private Image buttons;

  public UserHome(Control controle, User user) {
       this.controle = controle;
       this.user = user;
  }

  public void init() {
```

```
        Dimension screenDimension = DTVUtil.getScreenDimension();
        setLayout(new CoordinateLayout(screenDimension.getWidth(),
screenDimension.getHeight()));
        getStyle().setBgTransparency(0);

        container = new Container();
        Dimension dmContainer = new Dimension((int)
(screenDimension.getWidth() * 0.9), (int) (screenDimension.getHeight()
* 0.9));
        container.setSize(dmContainer);

        container.setLayout(new BorderLayout());
        container.setX((int) (screenDimension.getWidth() * 0.03F));
        container.setY((int) (screenDimension.getHeight() * 0.01F));
        container.getStyle().setBgTransparency(255);
        container.getStyle().setBgColor(Color.WHITE);
        container.getStyle().setMargin(0, 0, 0, 0);
        container.getStyle().setPadding(0, 0, 0, 0);
        container.getStyle().setPadding(0, 0, 0, 0);
        addComponent(container);

        createTopPanel();
        container.addComponent(BorderLayout.NORTH, topPanel);
        createCentralPane();
        container.addComponent(BorderLayout.CENTER, centralPanel);
        createSouthPanel();
        container.addComponent(BorderLayout.SOUTH, southPanel);

        DTVUtil.reserveColoredKeys(this);
        DTVUtil.reserveNumericKeys(this);
        addKeyListener(RemoteControlEvent.VK_ESCAPE, new
ActionListener() {
              public void actionPerformed(ActionEvent ae) {
                    close();
              }
        });

        show();
  }

  public void createTopPanel() {
        topPanel = new Container();
        topPanel.setLayout(new BorderLayout());

        Label userLabel = new Label("Benvindo,\n" + user.getName());
        Style style = userLabel.getStyle();


userLabel.setIcon(ResourceComponents.getImage(user.getFacebookUserId()
));
        userLabel.setTextPosition(Label.RIGHT);
```

```
        style.setBgColor(GSStyle.BLUE_FB);
        style.setFgColor(GSStyle.USER_DATA_FONT);

        topPanel.addComponent(BorderLayout.CENTER, userLabel);
    }

    public void createCentralPane() {
        centralPanel = new Container();
        centralPanel.setLayout(new BorderLayout());

        posts = new List();
        posts.setListCellRenderer(new
GingaSocialModernListCellRenderer(this));

        Vector dto = user.getHome();
        Iterator i = dto.iterator();
        while (i.hasNext()) {
            Post post = (Post) i.next();
            posts.addItem(post);
        }

        centralPanel.addComponent(BorderLayout.CENTER, posts);
    }

    public void updateTopCentralPane() {
        String dir = "top";
        try {
            Vector newPosts = Connector.getUserHome(user, dir);
            user.updateHome(newPosts, dir); // update user posts
        } catch (IOException e) {
            e.printStackTrace();
        }

        posts = new List();
        posts.setListCellRenderer(new
GingaSocialModernListCellRenderer(this));
        Vector dto = user.getHome();
        Iterator i = dto.iterator();
        while (i.hasNext()) {
            Post post = (Post) i.next();
            posts.addItem(post);
        }
        centralPanel.addComponent(BorderLayout.CENTER, posts);
    }

    public void updateBottomCentralPane() {
        String dir = "bottom";
        try {
            Vector newPosts = Connector.getUserHome(user, dir);
            user.updateHome(newPosts, dir); // update user posts
```

```
            for (Iterator iterator = newPosts.iterator();
iterator.hasNext();) {
                  Post post = (Post) iterator.next();
                  System.out.println("[DEBUG] UserName > " +
post.getUserName());
                  posts.addItem(post);
            }
      } catch (IOException e) {
            e.printStackTrace();
      }

  }

  public void createSouthPanel() {
      southPanel = new Container();
      southPanel.setHeight(35);
      Style styleSouth = southPanel.getStyle();
      styleSouth.setPadding(0, 0, 0, 0);
      styleSouth.setMargin(0, 0, 0, 0);

      Label acoesUsuario = new Label(buttons);
      southPanel.addComponent(acoesUsuario);
  }

  public void userInputEventReceived(UserInputEvent o) {
      KeyEvent event = (KeyEvent) o;
      int code = event.getKeyCode();

      if (code == RemoteControlEvent.VK_COLORED_KEY_1) {// verde -
curtir
            curtirPost();
      } else if (code == RemoteControlEvent.VK_COLORED_KEY_0) {//
vermelho - expandir
            Post post = (Post) posts.getSelectedItem();

            DTVUtil.releaseColoredKeys(this);
            DTVUtil.releaseNumericKeys(this);
            DialogGS dialog = new DialogGS(post, this);
            DTVUtil.reserveColoredKeys(dialog);
            DTVUtil.reserveNumericKeys(dialog);
            dialog.init();
            dialog.show(100, 50, 200, 200, false);

      } else if (code == RemoteControlEvent.VK_COLORED_KEY_2) {//
amarelo -
            compartilharPost();
      } else if (code == RemoteControlEvent.VK_COLORED_KEY_3) {//
azul -
      updateTopCentralPane();
      } else if (code == RemoteControlEvent.VK_1) {
            posts.setSelectedIndex(0);
```

```
        } else if (code == RemoteControlEvent.VK_0) {
              close();
        }
  }

  public void curtirPost() {
        Post post = (Post) posts.getSelectedItem();

        try {
              boolean resposta = Connector.like(user.getGingaId(),
post.getFacebookId());
              if (resposta) {
                    post.setLikes(post.getLikes() + 1);
              }
        } catch (IOException e) {

              e.printStackTrace();
        }
  }

  public void compartilharPost() {
        Post post = (Post) posts.getSelectedItem();
        try {
              Connector.share(user.getGingaId(), post.getFacebookId());
        } catch (IOException e) {
              e.printStackTrace();
        }
  }

  public Post getBottomItem() {
        return (Post) user.getHome().lastElement();
  }

  public void close() {
        setVisible(false);
        DTVUtil.releaseColoredKeys(this);
        DTVUtil.releaseNumericKeys(this);

        Vector home = user.getHome();
        for (Iterator iterator = home.iterator(); iterator.hasNext();)
{
              Post post = (Post) iterator.next();
              ResourceComponents.release(post.getFacebookId());
             ResourceComponents.release(post.getUserFacebookId());
        }
  }

  public void loadImage() {
        buttons = ResourceComponents.getImage(BUTTONS);
  }
}
```

ANEXO D – Artigo Ginga Social

Convergência TV Digital Interativa e Rede Social - Ginga Social

Gustavo Rupp Santos, Fiodor Castro

Departamento de Informática e Estatística (INE) – Centro Tecnológico (CTC) –
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

gustavorupp@inf.ufsc.br, fiodorc@inf.ufsc.br

Resumo. *Este artigo descreve como foi realizada a implementação de um software responsável pela integração entre TV Digital Interativa com a rede social Facebook. O sistema foi desenvolvido utilizando o Ginga-J e a API do Facebook.*

1. Introdução

O processo de definição do padrão para teledifusão digital no Brasil durou cerca de 15 anos, compreendendo os anos de 1991 a 2006. Neste processo, ainda em andamento, muito destaque está sendo dado ao sinal de alta definição em detrimento das demais características da televisão digital, praticamente ainda desconhecidas, como por exemplo, multiprogramação e interatividade.

No Brasil a interatividade será possível graças ao Ginga e a um canal de retorno. O Ginga é uma camada de *software* intermediário (*middleware*) que permite o desenvolvimento de aplicações interativas para a TV Digital de forma independente da plataforma de *hardware* dos fabricantes de terminais de acesso (*set-top boxes*). Aplicações interativas podem integrar serviços *Web*, complementar o conteúdo de programas sendo veiculados, jogos, aplicações para o governo (*T-Gov*), bancárias (*T-Banking*), comércio eletrônico (*T-Commerce*) e aplicações que permitam a integração com outros dispositivos que não o controle remoto. O canal de retorno é um meio que permitirá os televisores enviarem dados às emissoras de TV e, assim, de fato propiciar a interatividade.

Paralelamente deu-se a ascensão do Facebook, a qual trata-se de um serviço que conectam indivíduos e seu conteúdo consiste somente em informações disponibilizados por estes indivíduos. O grande número de pessoas adeptas à essa rede social, bem como o alto nível de interação entre seus usuários, fez com que esta se tornasse uma importante fonte de informação, combinando qualidade de informação com celeridade para disseminá-las.

As redes sociais passaram a influenciar fortemente as emissoras de televisão e os seus programas e, por outro lado, os conteúdos televisivos são os principais e mais frequentes temas repercutidos nas redes. Um caso interessante foi o uso do Twitter durante a transmissão do evento esportivo Super Bowl nos Estados Unidos, onde 50% da atividade nesta rede social foi relacionada ao evento. No Brasil aconteceu algo parecido durante a exibição do programa BBB10.

Esta interação entre redes sociais e TV Digital tende a se intensificar no Brasil a medida que o padrão SBTVD vai se concretizando já que, podemos observar, ambos possuem grande aceitação por parte dos brasileiros. Fala-se em uma convergência entre estas duas plataformas e esta nova realidade torna-se um interessante tema a ser estudado.

2. TV Digital no Brasil

Em 2003, com as discussões sobre TV Digital cada vez mais presentes, o governo, através do Decreto nº 4.901 (BRASIL, 2003), instituiu o Sistema Brasileiro de Televisão Digital - SBTVD. Além de inovação tecnológica o governo pretendia utilizar-se do SBTVD como uma ferramenta de combate à exclusão digital e consequentemente exclusão social, sendo que esta é uma das características que o distingue dos demais sistemas.

Após diversas discussões, em 2006 foi instituído um novo decreto, o de número nº 5.820, onde foi definido que o SBTVD teria como base o ISDB e seria possível incorporar inovações tecnológicas com o intuito de uma melhor adequação à realidade brasileira. O sistema brasileiro passou por alterações e de fato não se pode dizer que SBTVD e ISDB são equivalentes. A figura 1 ilustra a arquitetura do padrão brasileiro.

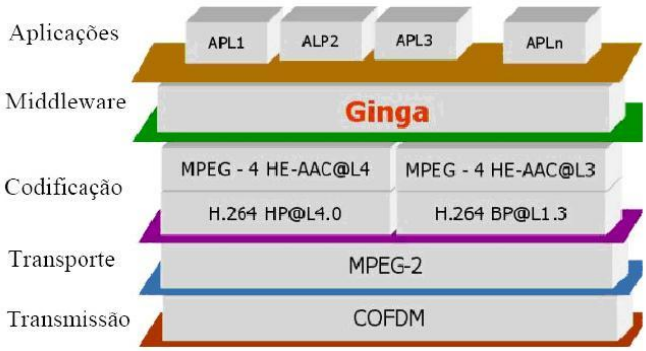


Figura 1. Arquitetura SBTVD. Fonte: (OLIVEIRA; LACERDA, 2008)

Assim como os demais, o padrão brasileiro é composto pelas camadas: transmissão, transporte, codificação, *middleware*, aplicação. Uma das inovações do SBTVD é *middleware* Ginga.

O Ginga possui dois ambientes de desenvolvimento, declarativo e imperativo, representados pelos componentes Ginga-NCL e Ginga-J respectivamente. A linguagem declarativa utilizada é a *Nested Context Lenguage* (NCL) e a linguagem imperativa é o Java bem como a linguagem Lua de script, esta ultima executada no ambiente delcarativo juntamente com o NCL.

Existe também a possibilidade de aplicações híbridas fazendo uso de uma ponte entre o ambiente declarativo e o procedural, no qual uma aplicação pode alterar e/ou executar uma aplicação de outro ambiente. Na arquitetura do Ginga, existem três componentes principais, Ginga-J, Ginga-NCL e Ginga-cc como mostrado na Figura 2.

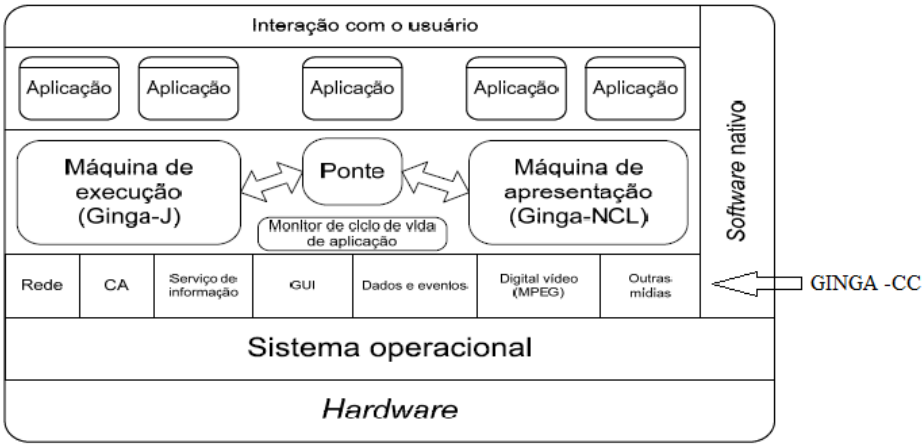


Figura 2. Arquitetura Ginga. Adaptado de (ABNT 15606-1).

O terceiro componente, o Núcleo Comum do Ginga (*Ginga Common Core*) é o subsistema responsável por oferecer funcionalidades específicas de TV Digital comuns para os ambientes imperativo e declarativo, abstraindo as características específicas de plataforma e *hardware* para as outras camadas acima. Como suas principais funções, podemos citar a: exibição e controle de mídias, o controle de recursos do sistema, canal de retorno, dispositivos de armazenamento, acesso as informações de serviço, sintonização de canais, entre outros (KULESZA *et al.* 2010).

4.1. Ginga-J

O Ginga-J é obrigatório em dispositivos fixos e composto por um conjunto de APIs definidas para atender todas as funcionalidades necessárias para a implementação de aplicativos para TV Digital, desde a manipulação de dados multimídia até protocolos de acesso. Sua especificação é formada por uma adaptação da API de acesso a informação de serviço do padrão japonês (ARIB B.23), pela especificação Java DTV (que inclui a API JavaTV), além de um conjunto de APIs adicionais de extensão ou inovação (PAULINELLI; KULESZA, 2012). Atualmente a arquitetura do Ginga-J está consolidada e é mostrada na Figura 3.

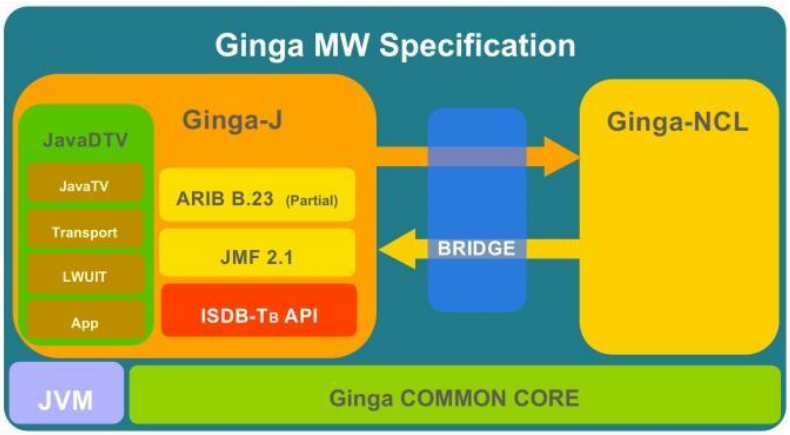


Figura 3. Arquitetura Ginga-J. Fonte: (PAULINELLI; KULESZA)

3. Redes Sociais

Em linhas gerais, o estudo das redes sociais é o estudo dos "agrupamentos sociais estabelecidos através da interação mediada pelo computador" (RECUERO, 2006). Uma rede social é composta por dois elementos: os atores (ou seja, as pessoas, instituições ou grupos, que constituem os nós da rede) e suas conexões (ou seja, as relações que se estabelecem entre os indivíduos) (RECUERO, 2006).

Atualmente, quando falamos de redes sociais no mundo vem a cabeça o mais utilizado, o Facebook. O infográfico da Figura 4 divulgado pelo portal de notícias russo Rian Novosti[17] demonstra o domínio desse *website* principalmente na América, Europa, África e Oceania. Mudando o foco para o Brasil, de acordo com a medição do Instituto de pesquisas IBOPE Nielsen Online[18], o Orkut que esteve na liderança de acessos diários durante 7 anos, perdeu seu posto em meados de agosto de 2011 (GALO, 2011). A partir dessa data, o Facebook ocupa a primeira colocação, assim como o faz em todo o restante do mundo, verificado na Figura 4.

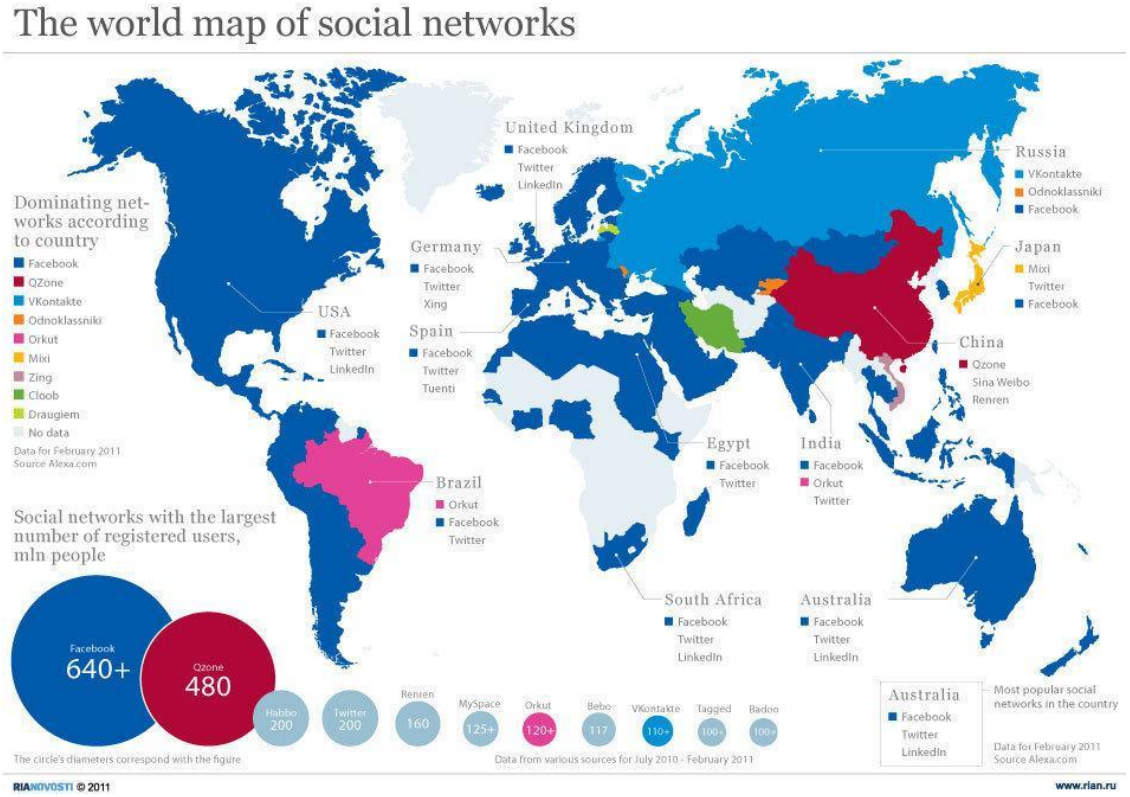


Figura 4. Infográfico representando o domínio das redes sociais no mundo. Fonte: (RIAN NOVOSTI, 2011)

Atualmente, o Facebook possui sede em diversos países. A tabela 1 apresenta dados interessantes dessa plataforma:

[17] Rian Novosti: http://en.rian.ru/

[18] IBOPE Nielsen: http://br.nielsennetpanel.com/pnl/br/home

Pessoas no FB	- Um bilhão de usuários ativos mensais a partir de outubro de 2012 - Mais de 50% dos usuários acessam o *site* por dia
Atividade no FB	- Mais de 900 milhões de objetos utilizados para interagir entre as pessoas (páginas, grupos, eventos e comunidades) - Em média, mais de 250 milhões de fotos são carregadas por dia
Alcance Global	- *Site* disponível em mais de 70 idiomas - Aproximadamente 80% dos usuários estão fora dos EUA
Plataforma	- Mais de 7 milhões de aplicativos e websites estão integrados com FB
Móvel	- Mais de 350 milhões de usuários ativos acessam o Facebook através dos seus dispositivos móveis - Mais de 475 operadoras móveis do mundo trabalham para implantar e promover os produtos Facebook móvel

Tabela 1 - Estatísticas do Facebook (FACEBOOK, 2012a).

No centro do Facebook está o gráfico social (FACEBOOK, 2012b), que representa as pessoas e suas conexões com seus interesses. Quando as pessoas usam a rede social, elas compartilham suas histórias, as quais aproximam outras pessoas e ajudam na construção de relacionamentos online e na vida real. Os aplicativos que conectam ao Facebook Open Graph possibilitam aos usuários compartilhar suas histórias com os amigos.

O gráfico social pode ser acessado através da Facebook Platform de inúmeras maneiras (APIs). O gráfico em si é um grafo que é definido na ciência da computação e matemática através da Teoria dos Grafos. Não trata-se apenas de uma simples tabela de dados, é uma série de nós os quais conectam-se uns aos outros.

A API utilizada nesse trabalho é a Graph API (FACEBOOK, 2012c) que trata-se da principal ferramenta de acesso ao gráfico social, permitindo ler e gravar dados no Facebook. Essa API apresenta uma visão simples e consistente do gráfico social Facebook, que representa objetos uniformemente no gráfico (pessoas, fotos, eventos e páginas) e as conexões entre eles (relacionamentos de amizade, conteúdo compartilhado e *tags* de fotos) (FACEBOOK, 2012d).

Cada objeto no gráfico social tem um identificador (ID) único e suas propriedades podem ser acessadas através de uma requisição a https://graph.facebook.com/ID. Todos os objetos no gráfico social estão ligados através de relacionamentos. O usuário Gustavo Rupp Santos é um fã da página do jogo Fifa 13, Gustavo e Fiodor Castro são amigos. Essas relações são chamadas de "conexões" na API e elas podem ser acessadas usando a estrutura de URL: https://graph.facebook.com/ID/TIPO_CONEXAO.

4. Sistema de Integração Ginga Social

A solução proposta pelos autores para tanto deu-se o nome de **Sistema de Integração entre TV Digital Interativa e Facebook**. Dado o objetivo e uma análise do atual cenário, com ferramentas e tecnologias disponíveis, verificou-se que a aplicação se estenderia para além do ambiente da TV.

4.1. Arquitetura da Implementação

Para demonstrar de forma prática o sistema proposto, foi definido uma arquitetura englobando todos os itens desse sistema que está ilustrada na Figura 5:

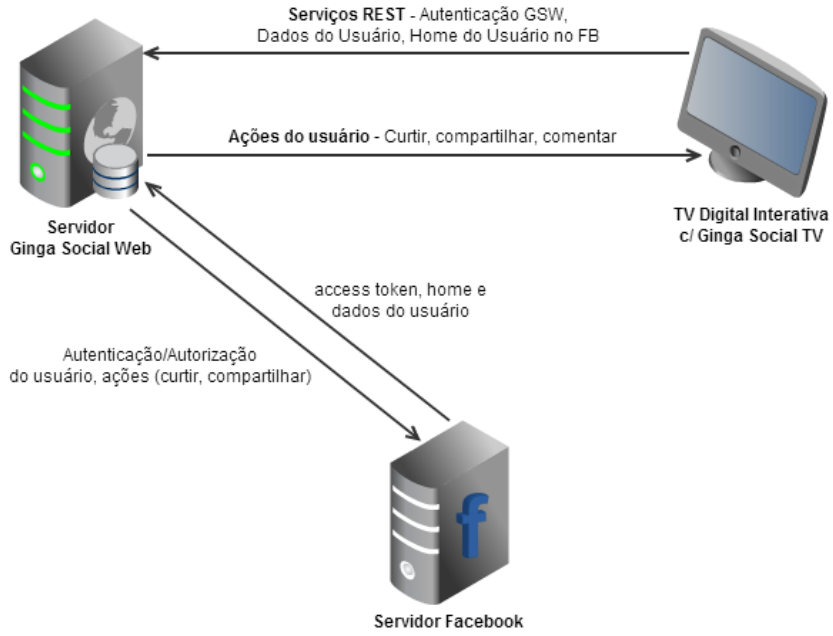


Figura 5. Arquitetura do Sistema de Integração entre TV Digital Interativa e Facebook

Na figura pode ser observado que há um ponto central de conexão entre TV e Facebook, trata-se do servidor *web* do Ginga Social, intitulado como **Ginga Social Web (GSW)**. O servidor Ginga Social atua como um *gateway* entre a aplicação de TV, doravante chamada **Ginga Social TV (GSTV)**, e o **Facebook** comunicando-se bidirecionalmente com ambos através da Internet.

Antes da execução da aplicação na TV o usuário deve permitir que seus dados, residentes no Facebook, sejam fornecidos à terceiros. O caso de uso Autorização/Autenticação Facebook contempla esta etapa, que é realizada através do Servidor GSW. Para tanto o usuário acessa a página do servidor "index.jsp" e autoriza explicitamente à aplicação Ginga Social obter seus dados junto ao Facebook. Após a autorização, são gerados automaticamente login e senha a esse usuário de forma que os demais casos de uso possam ser utilizados na TV.

Ao executar a aplicação na TV, uma autenticação se faz necessária junto ao GSW que possui o *access token* do usuário persistido no BD. Uma vez que o usuário autentique-se junto ao sistema, o Ginga Social Web obtém os dados do usuário junto ao Facebook para que seja possível a Exibição do Facebook *Home* do usuário. Com o *home* disponível em tela, pode-se curtir ou compartilhar um *post* e, a critério do usuário, atualizar seu *home* com os *posts* mais recentes ou mais antigos.

4.2. Servidor Ginga Social Web (GSW)

A lógica de negócio do sistema, o modelo de dados e sua persistência estão situados nesse servidor, justificando sua presença como elemento fundamental e principal na arquitetura projetada. Ademais, um servidor intermediando a comunicação elimina uma

dependência direta da aplicação com relação ao Facebook, bem como torna o ambiente mais amigável para possíveis extensões do conjunto de funcionalidades.

Para comunicação entre o GSW e o GSTV foi implementado no servidor um *web service* REST para que a aplicação cliente possa consultar dados e/ou enviar ações do telespectador. Os serviços disponibilizados são os seguintes: obtenção do dados do usuário cadastrado no Ginga Social, obtenção dos *home* do usuário no Facebook, autenticação do telespectador na aplicação cliente, envio da ação 'curtir' e 'compartilhar' advinda da TV.

4.3. Servidor Ginga Social Web (GSTV)

Da parte da televisão foi desenvolvida uma aplicação em consonância com o padrão brasileiro de TV digital, mais especificamente com a máquina de execução Ginga-J. Optou-se por esse ambiente devido a um conhecimento sólido da linguagem Java pelos autores. Para a implementação do Ginga Social TV foi utilizado como ambiente de desenvolvimento uma máquina virtual denominada Astro Box (TOTVS, 2012a).

As figuras 6 e 7 ilustram na prática como a aplicação cliente é apresentada para o telespectador:

Figura 6. Tela de autenticação do usuário Ginga Social TV

Figura 6. Tela com o *home* do Facebook apresentada na TV

5. Conclusão

O primeiro grande desafio deste trabalho foi encontrar um ambiente apropriado para o desenvolvimento das aplicações Ginga-J. Para tanto, seria necessário dispor de uma plataforma que contivesse a implementação, ao menos de grande parte, do *middleware* Ginga pois, como vimos, a máquina de execução depende das camadas inferiores do *middleware*.

Apesar das dificuldades enfrentadas, devemos lembrar que praticamente todos os lares brasileiros possuem um televisor, fazendo deste um dos principais meios de comunicação do país. Com este cenário em mente, somos levados a pensar que, sem dúvidas, a TV Digital no Brasil possui um grande potencial considerando o perfil sócio-econômico de sua população. Embora o mercado da televisão, com emissoras e fabricantes de TV, esteja inserido dentro deste contexto favorável, o fato é que observa-se muito pouco avanço quando o assunto é interatividade.

Este trabalho pode ser encarado, também, como uma forma de disseminação da TV Digital e suas capacidades, já que, aliando-a às redes sociais, mais especificamente ao Facebook, há muito mais chances das pessoas tomarem conhecimento das potencialidades dessa tecnologia.

Por fim, acreditamos que o objetivo principal desta empreitada foi alcançado, qual seja, a integração entre redes sociais e TV Digital. Através do desenvolvimento do sistema Ginga Social, proporcionamos o contato entre estas duas diferentes mídias, considerando as limitações de ambas, principalmente no lado da TV. Outrossim, os estudos referentes aos dois temas também foram satisfatórios à medida que deram suporte para que o trabalho fosse concluído.

Referências

ABNT - ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 15606-1. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital. Parte 1: Codificação de dados. 2011.

BRASIL. Decreto nº 4.901, 26 de novembro de 2003. Institui o Sistema Brasileiro de Televisão Digital - SBTVD, e dá outras providências. Diário Oficial da União, 27 nov. 2003, p7.

BRASIL. Decreto nº 5.820 - 29 de junho de 2006. Dispõe sobre a implantação do SBTVD-T, estabelece diretrizes para a transição do sistema de transmissão analógica para o sistema de transmissão digital do serviço de radiodifusão de sons e imagens e do serviço de retransmissão de televisão, e dá outras providências. Diário Oficial da União, 30 jun. 2006, p51

FACEBOOK. ESTATÍSTICAS FACEBOOK. Disponível em: <http://www.facebook.com/press/info.php?statistics>. Acesso em: Outubro, 2012a.

FACEBOOK. Open Graph Overview. Disponível em: <https://developers.facebook.com/docs/concepts/opengraph/overview/>. Acesso em: Outubro, 2012b

FACEBOOK. Getting Started: The Graph API. Disponível em: <http://developers.facebook.com/docs/getting-started/graphapi/>. Acesso em: Outubro, 2012c.

FACEBOOK. Graph API. Disponível em: http://developers.facebook.com/docs/reference/api/. Acesso em: Outubro,2012d.

GALO, Bruno. Você pode ganhar muito dinheiro no Facebook. Istoé Dinheiro. 2011. Disponível em: http://www.istoedinheiro.com.br/noticias/65157_VOCE+PODE+GANHAR+MUITO+DINHEIRO+NO+FACEBOOK. Acesso em: Setembro, 2012.

KULESZA, Raoni; et al.; Ginga-J: Implementação de Referência do Ambiente Imperativo do Middleware Ginga, 2010. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/webmedia/2010/05_webmi_c.pdf>. Acesso em: Novembro, 2012.

OLIVEIRA, Antônio Carlos Albuquerque de; LACERDA, João Paulo Lopes de. A TV digital no Brasil e o desenvolvimento de aplicações interativas para o middleware Ginga. 2008. Trabalho de Conlusão de Curso (Graduação em Ciência da Computação). Universidade Federal de Sergipe. Aracaju. 2008.

PAULINELLI, Fernanda; KULESZA, Raoni. Histórico Ginga-J. GingaCDN. Disponível em: http://gingacdn.lavid.ufpb.br/projects/openginga/wiki/Hist%C3%B3rico_Ginga-J. Acesso em: Julho, 2012.

RECUERO, Raquel da Cunha. Comunidades em Redes Sociais na Internet: Proposta de Tipologia baseada no Fotolog.com. Porto Alegre: UFRGS, 2006. Tese (Doutorado em Comunicação e Informação), Universidade Federal do Rio Grande do Sul. 2006.

RIAN NOVOSTI. The world map of social networks. Rian Novosti. 2011. Disponível em:

<http://en.rian.ru/infographics/20110228/162792394.html>. Acesso em: Dezembro, 2011.

TOTVS. Astro Box > Sobre. Disponível em: <https://www.astrodevnet.com.br/AstroDevNet/restrict/astrobox_sobre.html>. Acesso em: Março, 2012a.