Ginga-J: Implementação de Referência do Ambiente Imperativo do Middleware Ginga

Raoni Kulesza[1,2], Jefferson Ferreira[2], Sindolfo M. Filho[2], Álan Lívio[2],
Rafael R. de M. Brandão[2], Jônatas P. C. de Araujo[2], Guido L. de S. Filho[2]

[1]Centro de Informática - UFPE
Campus I - Cidade Universitária
Recife/PE - 50740-54

[2]Departamento de Informática - UFPB
Campus I - Cidade Universitária
João Pessoa/PB - 58059-900

{raoni, jefferson, sindolfo, alan, rafael, jonatas, guido}@lavid.ufpb.br;

ABSTRACT
This paper aims to present a Ginga-J's reference implementation. Although based on a particular platform, the implementation not only works as a proof of concept, but also raised several issues and difficulties on the software architecture project that should be taken into account to ease extensibility and porting to other platforms. Ginga is the standard middleware for the Brazilian DTV System and its imperative environment (Ginga-J) is mandatory for fixed terrestrial receptors.

RESUMO
Este artigo tem como objetivo apresentar a primeira implementação de referência do Ginga-J. Embora desenvolvida para uma plataforma particular, a implementação serviu não apenas como prova de conceito, mas também para levantar diversas questões e principais dificuldades sobre o projeto da arquitetura do software e propor estratégias de implementação que facilitassem a evolução e porte do middleware para outras plataformas. Ginga é o middleware padrão do Sistema Brasileiro de TV Digital e Ginga-J é o ambiente imperativo obrigatório para dispositivos fixos.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architecture, Language, Patterns*

General Terms
Algorithms, Documentation, Design, Standardization, Languages.

Keywords
DTV middleware, Java, JavaDTV, Digital TV, Ginga, SBTVD.

1. INTRODUÇÃO

Com o surgimento da TV Digital um novo conjunto de funcionalidades foi agregado tanto à programação oferecida pelas emissoras quanto aos receptores de TVD. Com isso, o ambiente da TV pode se tornar mais interativo, dado que o sistema de TV passou a oferecer ambientes para a execução de aplicações (software). Estas aplicações podem então ser transmitidas e executadas conjuntamente com conteúdos multimídia, áudio, vídeo, imagem, texto, entre outros, possibilitando o aumento da interatividade entre o telespectador e a televisão através de serviços de entretenimento, educação, saúde, comércio, dentre outros [1]. A execução dessas aplicações não seria possível sem o auxilio de uma camada intermediária de *software* denominada *middleware,* instalado em cada terminal de acesso.

O principal papel de um *middleware* de TV Digital é atuar como uma camada de *software* intermediária que faz a interface entre o sistema operacional e as aplicações, abstraindo características específicas da plataforma, além de prover uma série de serviços específicos para as camadas superiores, permitindo a execução de aplicações portáveis para dispositivos com capacidades distintas de processamento e arquiteturas de hardware.

As principais propostas de *middlewares* para TV Digital oferecem como suporte a execução de aplicações interativas em dois ambientes: declarativo e imperativo[2]. No Sistema Brasileiro de TV Digital (SBTVD), o ambiente declarativo é representado pelo Ginga-NCL [3], que suporta aplicações baseadas na linguagem NCL (*Nested Context Language*) e o ambiente imperativo é representado pelo Ginga-J [4], que suporta a execução de aplicações escritas na linguagem Java.

Este trabalho tem como principal objetivo apresentar uma implementação de referência do Ginga-J, citando suas singularidades quando comparada a outras implementações de *middleware*. Na norma ABNT NBR 15606-4 [4] podem ser encontradas todas as informações sobre a especificação das funcionalidades Ginga-J, Este artigo descreve uma implementação no contexto do projeto GingaCDN[1] com o

[1] Projeto GingaCDN. Disponível em http://dev.openginga.org

intuito de servir de base para outras implementações futuras do Ginga-J, específicas para cada fabricante e suas plataformas. Outro ponto discutido neste artigo é a evolução e validação da arquitetura da implementação de referência do Ginga-J, uma vez que esta é baseada em trabalhos anteriores de desenvolvimento de *middleware* realizado pelo grupo de pesquisadores do LAVID na UFPB [6] . Os principais resultados do trabalho aqui apresentado foram: (i) definição de uma arquitetura flexível que permite reuso e extensibilidade de *software* e; (ii) realização de implementação de referência em conformidade com a nova API JavaDTV adotada recentemente pelo Ginga.

O restante do artigo está organizado da seguinte forma: a seção 2 descreve o *middleware* Ginga. A seção 3 apresenta o histórico da especificação Ginga-J. A seção 4 discorre sobre a arquitetura da implementação proposta para o Ginga-J. A seção 5 detalha a implementação. A seção 6 realiza uma breve comparação entre outras implementações e o Ginga-J. E, por fim, a seção 7 apresenta as considerações finais, trabalhos futuros e em andamento.

2. O *MIDDLEWARE* GINGA

O Ginga é a especificação de *middleware* do SBTVD, resultado da junção das propostas FlexTV[6] e MAESTRO[7], desenvolvidos por consórcios liderados pela UFPB e PUC-Rio no projeto SBTVD[2], respectivamente.

O FlexTV, proposta inicial de *middleware* procedural do SBTVD, incluía um conjunto de APIs compatíveis com outros padrões além de funcionalidades inovadoras, como a possibilidade de comunicação com múltiplos dispositivos, permitindo que diferentes usuários pudessem interagir com uma mesma aplicação interativa a partir de dispositivos remotos. Já o MAESTRO foi a proposta inicial de *middleware* declarativo do SBTVD. O foco era oferecer facilidade do sincronismo espaço-temporal entre objetos multimídia, utiliza a linguagem declarativa NCL e agregado as funcionalidades da linguagem de script Lua.

O Ginga integrou estas duas soluções, agora chamadas de Ginga-J [8]e Ginga-NCL, tomando por base as recomendações internacionais da ITU [10]. Desta forma, o Ginga é subdividido em dois subsistemas interligados (Figura 1), também chamados de Máquina de Execução (Ginga-J) e Máquina de Apresentação (Ginga-NCL). A execução do conteúdo imperativo é possível através da Máquina Virtual Java. Dependendo do contexto, uma solução pode ser mais adequada que a outra.

Outro aspecto importante é que esses dois ambientes não são necessariamente independentes, uma vez que a recomendação do ITU inclui uma "ponte", que deve disponibilizar mecanismos para intercomunicação entre os mesmos, de um modo que as aplicações imperativas utilizem serviços disponíveis nas aplicações declarativas, e vice-versa. Portanto, é possível a execução de aplicações híbridas em um nível acima da camada dos ambientes de execução e apresentação, permitindo agregar as facilidades de apresentação e sincronização de elementos multimídias da linguagem NCL com o poder da linguagem orientada a objetos Java

[2] SBTVD. 2005. Projeto do Sistema Brasileiro de TV Digital. Disponível em: http://sbtvd.cpqd.com.br. Acesso em: 01/05/2010

Figura 1 - Visão geral do middleware Ginga

O Núcleo Comum 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 a informações de serviço, sintonização de canais, entre outros.

3. ESPECIFICAÇÃO DO GINGA-J

O Ginga-J (Figura 2) é composto por um conjunto de APIs definidas para atender funcionalidades necessárias para o desenvolvimento de aplicativos para TVD, 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 (ISDB ARIB B.23) [14], pela especificação Java DTV [11] (que inclui a API JavaTV[12]) , além de um conjunto de APIs adicionais de extensão ou inovação.

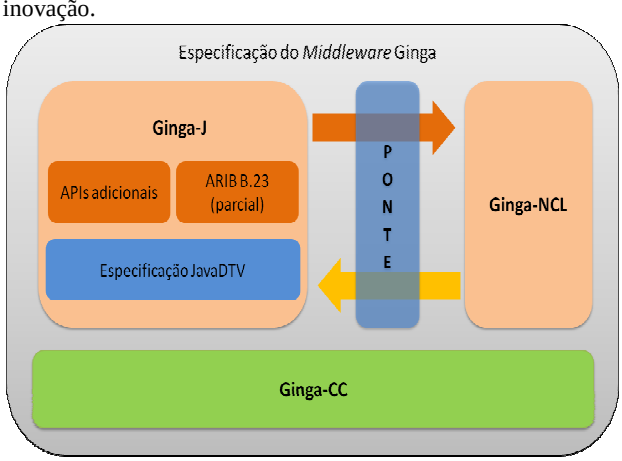

Figura 2 - Visão geral do Ginga-J

As APIs adicionais incluem um conjunto de classes disponíveis para a ponte entre os aplicativos NCL e Java, funcionalidades adicionais para sintonia de canais, envio de mensagens assíncronas pelo canal de interatividade e integração de dispositivos externos ao *middleware*, viabilizando a interação simultânea de múltiplos usuários e dispositivos em aplicações de TVD. [21][22]

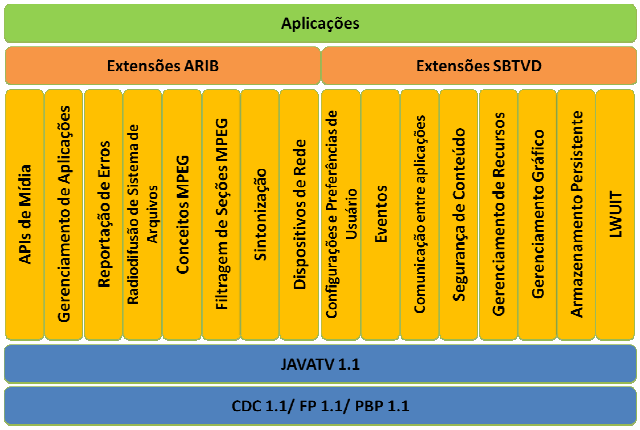

Figura 3 - Conjunto de APIs Ginga-J

A especificação Java DTV é uma plataforma aberta e interoperável que permite a implementação de serviços interativos com a linguagem Java, tendo sido inserida recentemente ao conjunto de APIs do Ginga-J. Funcionalmente, a JavaDTV substitui a coleção de APIs definidas inicialmente para o Ginga-J[8] e utilizadas pelo padrão GEM [13] (*Globally Executable* MHP), tais como DVB (*Digital Video Broadcast*), DAVIC (*Digital Audio Video Council*) e HAVi (*Home Audio Video Interoperability*). O objetivo é prover uma solução livre de *royalties* para permitir a fabricação de aparelhos de TVD e/ou conversores e desenvolvimento de aplicações com um custo mais acessível. Uma diferença importante da Java DTV em relação ao GEM, é a API LWUIT (*LightWeight User Interface Toolkit*), responsável por definir elementos gráficos, extensões gráficas para TVD, gerenciadores de leiaute e eventos do usuário.

Adicionalmente, o Ginga-J é composto pela API JavaTV, APIs de segurança[15][16][17] e o ambiente de execução Java para sistemas embarcados (JavaME), incluindo a plataforma CDC (*Connected Device Configuration*)[18], e as APIs dos perfis: FP (*Foundation Profile*)[19] e PBP (*Personal Basis Profile*) (Figura 3)[20].

4. ARQUITETURA DA IMPLEMENTAÇÃO DE REFERÊNCIA

A especificação da arquitetura de referência do Ginga-J utilizou como base a arquitetura do FlexTV [6], que considerou a arquitetura do ITU J.200 [10]. Entretanto, além de seguir um conjunto diferente de definições de APIs (baseado no JavaDTV, e não no GEM), outras características foram adicionadas com o objetivo de proporcionar melhor reuso e qualidade de *software*. A Figura 4 ilustra a arquitetura conceitual modularizada: (i) sistema operacional; (ii) camada de núcleo comum e; (iii) máquina de execução Ginga-J. A seguir são descritas as três etapas que foram seguidas para definir a solução de arquitetura.

A primeira etapa de definição da arquitetura foi escolher a plataforma de execução adequada às características e limitações diferenciais de um terminal fixo de TV Digital. Nesse sentido, optamos pelo sistema operacional Linux para computadores pessoais (x86) e a máquina virtual Java PhoneME[4], que é a implementação oficial do ambiente JavaME/CDC. O principal motivo da escolha foi a disponibilidade do Linux e da PhoneME como plataformas abertas e também o oferecimento de várias ferramentas de desenvolvimento sem custo adicional. Além disso, o Linux suporta sistemas heterogêneos. [23] O objetivo foi permitir o desenvolvimento da implementação num ambiente muito próximo de um terminal de acesso, mas que também pudesse estar disponível para o maior número possível de desenvolvedores. No caso, um computador pessoal, sem a necessidade de aquisição de nenhum *hardware* específico.

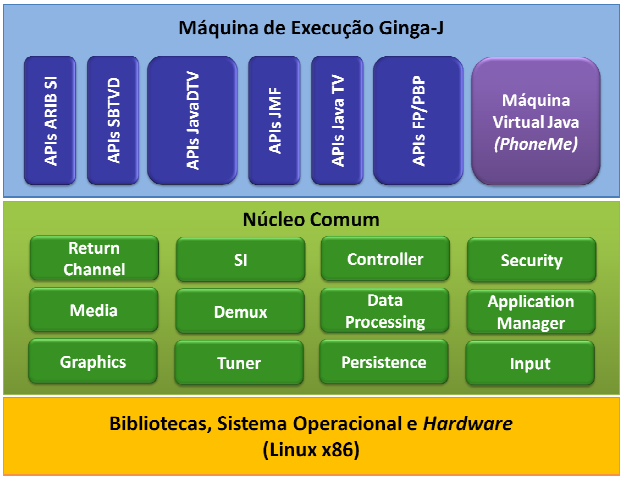

Figura 4 - Arquitetura Conceitual

A segunda etapa foi evoluir e refinar a arquitetura do núcleo comum do FlexTV. Quase nenhuma mudança foi realizada em relação à definição conceitual dos módulos desse subsistema, apenas houve um reagrupamento para obter melhor coesão das funcionalidades. A principal mudança foi especificar o núcleo comum utilizando uma abordagem baseada em componentes, adotando um modelo e ambiente de execução de componentes: FlexCM[24]. O objetivo foi enfatizar a modelagem de *software* através da decomposição do sistema em componentes funcionais com interfaces de interação bem definidas. Neste contexto, um modelo de componentes padroniza o esquema de instanciação, composição e o ciclo de vida dos componentes do sistema e um ambiente de execução de *software* responsável por gerenciar os componentes garantindo as especificações definidas pelo respectivo modelo de componentes.

O modelo FlexCM segue uma abordagem declarativa, onde os componentes definem suas dependências explicitamente (interfaces requeridas) e o ambiente de execução carrega e provê as dependências através do padrão de injeção de dependências. Desta forma, o modelo FlexCM permite que seus componentes conheçam apenas as interfaces, as implementações são tratadas pelo ambiente de execução. Além das interfaces requeridas, os componentes também podem declarar parâmetros de configuração cujos valores também são injetados pelo ambiente de execução permitindo que o desenvolvedor configure facilmente o componente no produto final onde ele será instalado. O ambiente de execução FlexCM é capaz de carregar

[4] Projeto PhoneME. Disponível em: https://phoneme.dev.java.net/ Acesso em: 01/05/2010

todo o sistema a partir de um arquivo de descrição arquitetural no qual são especificadas as conexões e as configurações dos mesmos.

Além das vantagens comumente conhecidas do desenvolvimento baseado em componentes como, por exemplo, modularidade, manutenibilidade e reuso, a adoção do modelo de componentes FlexCM ofereceu uma série de vantagens específicas para a implementação do GingaCC para o Ginga-J. Pode-se citar: (1) conhecimento de arquitetura em nível de modelo; (2) facilidades na configuração dos componentes individuais e; (3) na configuração do sistema como um todo. Por fim, essas características trazem a possibilidade de gerenciar diferentes arquiteturas facilitando também a execução de testes de unidade e integração de diferentes porções da arquitetura. Uma proposta de processo de testes para o Ginga-J baseada no FlexCM pode ser encontrada em [25].

A terceira e última etapa foi definir um modelo de integração da camada de Núcleo Comum com a Máquina de Execução Ginga-J. Como já mencionado, o Núcleo Comum é responsável por oferecer serviços a máquina de execução Ginga-J. Conseqüentemente, contém código nativo (em linguagem C ou C++) e é dependente de bibliotecas da plataforma de execução. Nesse sentido, foi importante definir um modelo de comunicação de forma a diminuir o acoplamento e a dependência entre os dois subsistemas. A solução adotada foi baseada nos padrões de projeto *Proxy*, *Facade* e *Adapter* [26]. A idéia é que todas as chamadas da máquina de execução Java sejam centralizadas num módulo Controller, que expõe serviços para aplicações (do inglês, *Application Services*). A Figura 5 ilustra o módulo Controller com duas interfaces AS: ITunerAS e IDemuxAS. Esses serviços serão oferecidos para as aplicações Ginga-J através de chamadas JNI(*Java Native Interface*) implementadas internamente pelos pacotes Java. O Controller chama por delegação o componente que implementa a funcionalidade requisitada. Por exemplo, chamadas para ITuner e IDemux. Caso um componente do Núcleo Comum necessite de uma funcionalidade de um componente do Núcleo Comum, o mesmo pode chamar o outro componente diretamente. A principal vantagem dessa abordagem é isolar a(s) camada(s) acima do Núcleo Comum, de forma a evitar a dependências com a plataforma, assim como diminuir o acoplamento entre a especificação das APIs Java e a implementação em C e/C++. Por exemplo, o porte de uma máquina de apresentação Ginga-NCL ou uma máquina de execução Java de outro sistema de TV Digital (GEM) para o Núcleo Comum utilizado neste trabalho seria facilitado.

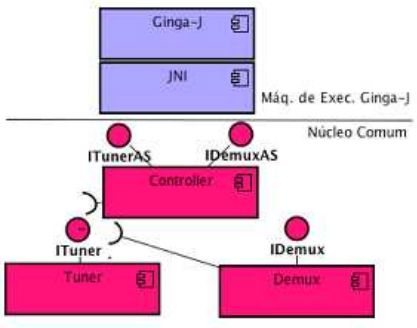

Figura 5 - Integração da Camada de Execução Ginga-J com o Núcleo Comum

5. IMPLEMENTAÇÃO

Nesta seção é descrita a implementação dos componentes do Núcleo Comum que oferecem serviços para o Ginga-J A Figura 6 exibe uma visão geral desse subsistema do *middleware*, que contém os seguintes componentes:

(1)*Tuner* - sintoniza e controla o acesso aos múltiplos fluxos de transporte da rede; (2) *SI* – obtém informações de serviço do fluxo de transporte, ou seja, quais fluxos elementares (semântica) de áudio, vídeo e dados estão sendo transmitidos, além de informações como classificação indicativa, sinopses e horários; (3) *Demux* - disponibiliza filtros específicos para selecionar fluxos; (4) *Media*: comunica-se com os decodificadores (*hardware* ou *software*), a fim de gerenciar e exibir a apresentação dos fluxos elementares de áudio e vídeo; (5) *Data Processor* - processa e separa dados transmitidos (como aplicações) de forma multiplexada no fluxo de transporte (do inglês, *Transport Stream*) MPEG-2. (6) *Graphics* - permite o desenho de componentes gráficos; (7) *Input Manager* - tratamento de eventos disparados pelo usuário através do controle remoto, pelo painel do próprio terminal de acesso, por um teclado, ou por algum outro dispositivo de entrada; (8) *Return Channel* - provê interfaces para utilização do canal de retorno, por exemplo, através de um modem de linha discada, ADSL, Ethernet, Wimax ou 3G; (9) *Application Manager*: carrega, instancia, configura e executa aplicações; (10) *Persistence* - gerencia recursos de armazenamento não voláteis; (11) *Security* - verifica a autenticação e permissões de aplicações interativas; (12) *Middleware Manager* - responsável pelo gerenciamento funcional do *middleware*.

Como já citado, para a implementação da máquina de execução Ginga-J utilizou-se a versão *RC2 da PhoneME Advanced* para a plataforma Linux. Na máquina virtual foi realizado o porte nativo da parte gráfica da API AWT para *DirectFB*[5]. O código gerado foi baseado na implementação nativa em Qt, embutido na *PhoneME*. A partir daí foi possível implementar as APIs JavaDTV utilizando as classes bases Java já existentes na *PhoneME*, por exemplo, as APIs para desenhos gráficos e tratamento de eventos do usuários. Tais funcionalidades foram encapsuladas nos componentes *Graphics* e *Input*, respectivamente. Para as funcionalidades que não estavam presentes no ambiente Java foi necessária uma integração com componentes do Núcleo Comum. Além disso, para permitir o gerenciamento das aplicações Java, também foi necessário integrar a JVM com o Núcleo Comum através dos componentes Controller e *ApplicationManager*. Este componente implementou um novo elemento *proxy* que possibilitou a execução de *Xlets*. Para isto foi utilizada a função ansiJavaMain() disponivel no código da JVM. Essa função inicia a JVM e executa uma classe Java, que inicia as camadas gráficas do receptor como especificado no padrão SBTVD, além de carregar os dados das aplicações (*Xlets*) que são iniciados como *Threads* separadas, uma vez que o Ginga permite a execução de mais de uma aplicação no receptor.

[5] DirectFB é um projeto de código aberto que provê aceleração gráfica, tratamento de eventos de entrada, gerenciamento de camadas gráficas e reprodução de mídias diversas através de provedores multimídia: Disponível em: http:// www.directfb.org

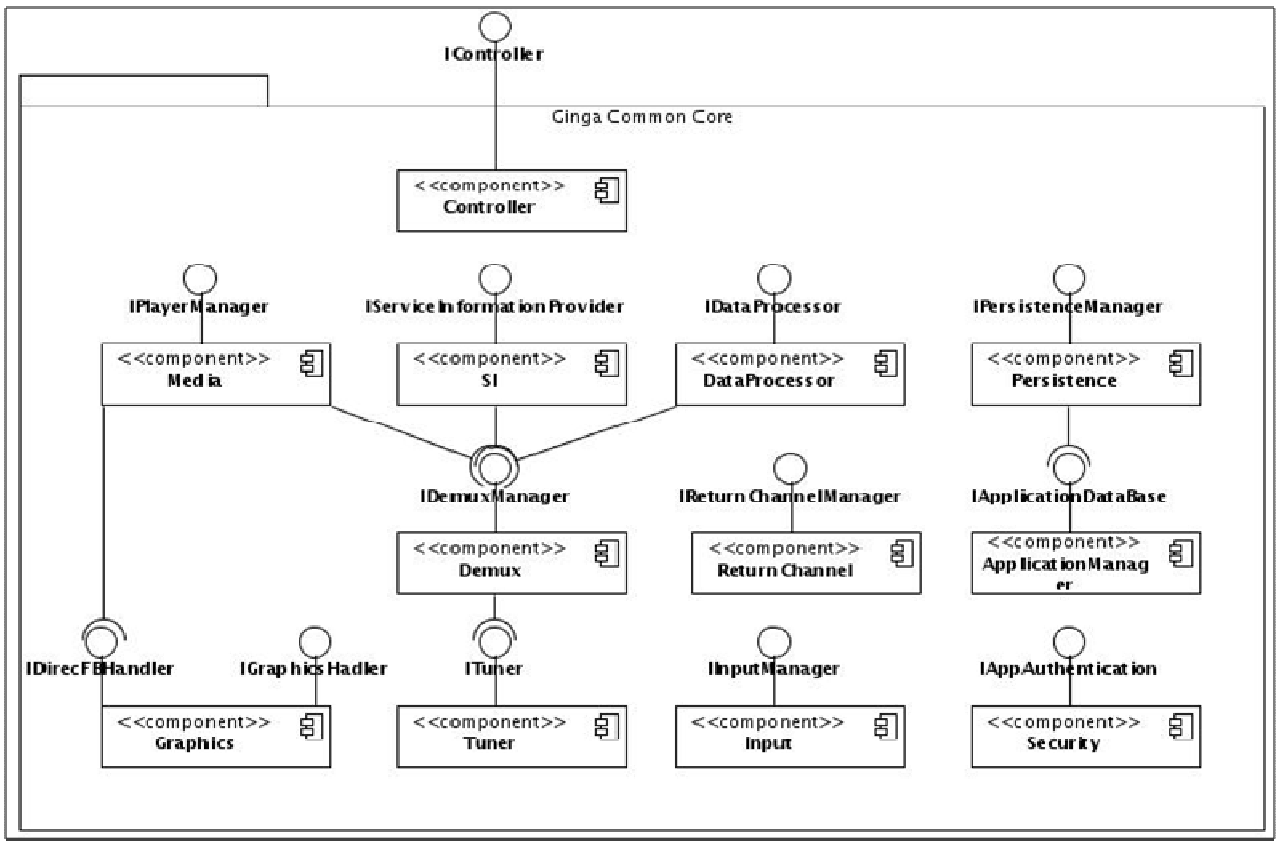

Figura 6 - Implementação do GingaJ Núcleo Comum (no caso, Controller não é componente, apenas uma fachada)

O *Tuner* implementa os serviços definidos pelo pacote *com.sun.dtv.tuner*, utilizando um processo de varredura não-bloqueante para identificação de canais, baseado em eventos e no padrão *Observer*[26].

O componente *Demux* contém funcionalidades do pacote *com.sun.dtv.filtering*, que permite selecionar diferentes tipos de fluxos elementares. Internamente utiliza um "*buffer* em fila circular" com diferentes ponteiros de início para alimentar cada requisição, buscando evitar que um usuário do componente (API ou outro componente) receba o conteúdo já consumido por outros.

O componente SI obedece os requisitos das APIs que tratam de informações de serviço (ARIB, JavaTV e JavaDTV), além de permitir ao usuário do componente a obtenção de informações finais sobre o fluxo, sem a necessidade de outro processamento, pois implementa um mecanismo de *cache*. Todas as abstrações para informações de serviço previstas nas normas do SBTVD[27] podem ser gerados a partir de uma fábrica de objeto,que utiliza o padrão de projeto *Factory Method* [26]. Este componente também notifica o *DataProcessing* para o mesmo realizar o processamento de dados de sinalização e execução de aplicações e carrossel de dados.

Já componente *Media* é responsável por todo o processamento de mídias contínuas (áudio e vídeo) no *middleware* e também se utiliza da API *DirectFB*. Este componente foi projetado considerando os requisitos da API JMF, já que ele provê as funcionalidades básicas de reprodução para a API Java através de chamadas JNI.

Os componentes *Persistence* e *Security* seguem rigidamente o modelo JavaDTV para empacotamento, autenticação e autorização de aplicações e armazenamento de arquivos. Já o componente *Return Channel* implementa comunicação TCP/IP para diferentes tecnologias de rede.

A Figura 7 ilustra quatro cenários de uso da implementação do Ginga-J. A Figura 7(a) exibe uma aplicação Java (*Xlet*) utilizando as APIs de acesso a Informações de Serviço (JavaTV SI e ARIB SI) e APIs para elementos gráficos (LWUIT) do Ginga-J. A Figura 7(b) mostra uma aplicação exibindo 3 fluxos de vídeo (2 locais e 1 ao vivo) como cenário de validação da implementação da API de execução de mídias (JMF). Já as Figura 7(c) e Figura7(d) ilustram a possibilidade de várias configurações do middleware através de uma aplicação residente OSD (do inglês, *On Screen Device*). Os dois últimos exemplos apoiaram a validação as APIs para controle de ciclo de vida da aplicação (JavaTV), carrossel de dados, persistência e segurança (JavaDTV). Os testes foram realizados utilizando um computador pessoal com a seguinte configuração: processador Core 2 Duo 2.16GHz; 1GB de memória RAM; sistema operacional Ubuntu 9.10 Kernel 2.6.31-14 e; um disco rígido de 100 GB.

Figura 7 - Cenários de uso do Ginga-J

6. TRABALHOS RELACIONADOS

As principais implementações de *middleware* existentes no contexto de TV Digital podem ser divididas em duas categorias: (1) ambientes declarativos (2) ambientes imperativos. O primeiro grupo é representado por: (i) LASeR (*Ligheweight Application Scene Representation*)

[28] ;(ii) BML (*Broadcast Markup Language*) [29]; (iii) GingaNCL para dispositivos portáteis[30] e; (iv) Ginga-NCL para dispositivos fixos[31]. Já para o segundo, pode-se citar: i) FlexTV[6] e (ii) OCAP-RI (*OpenCable Application Platform – Reference Implementation).* [32]
Informações sobre o LASeR, BML e Ginga-NCL para dispositivos portáteis podem ser encontradas em [30]. A principal diferença dessas soluções em relação a esta proposta (Ginga-J) está no projeto da arquitetura. Nenhuma dessas soluções utiliza uma abordagem orientada a componentes, não definindo um modelo e ambiente de execução para os módulos dos sistemas. Além disso, pode-se observar que esses ambientes procuram implementar as seguintes funcionalidades: sincronismo das mídias, adaptabilidade, suporte a múltiplos dispositivos, suporte a edição em tempo de exibição e, também, suporte ao reuso. O Ginga-NCL para dispositivos portáteis é a única solução que suporta múltiplos dispositivos e que atende da melhor forma o suporte ao reuso. A solução aqui proposta também atende todos esses requisitos apresentados pelos ambientes declarativos, porém utiliza-se de uma abordagem imperativa, através da linguagem orientada a objetos Java. O uso desse tipo de linguagem é muito mais difícil e suscetível a erros e também exige um maior *footprint* das aplicações. Porém, possuem um poder de expressão maior que aquele oferecido pelas linguagens declarativas. O objetivo é oferecer aplicações mais avançadas que precisam utilizar, por exemplo, mecanismos de segurança e acesso e controle mais fino a informações e conteúdo áudio-visual.

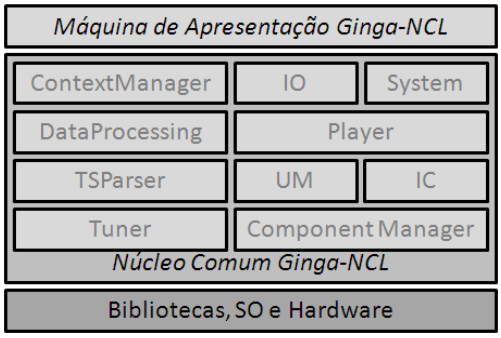

Figura 8 - Visão Geral do Ginga-NCL para dispositivos fixos

A Figura 8 exibe uma visão geral da arquitetura da implementação do Ginga-NCL para dispositivos fixos [31]. A Máquina de Apresentação Ginga-NCL é um subsistema lógico capaz de iniciar e controlar aplicações NCL. O Núcleo Comum Ginga-NCL é responsável por oferecer os serviços anteriormente mencionados à Máquina de Apresentação Ginga-NCL. Tal solução, apesar de atender um conjunto diferente de aplicações, apresenta bastante semelhança na definição de funcionalidades do Núcleo Comum do Ginga-J. As diferenças são a ausência de funcionalidades de segurança e um conjunto menor de informações de serviço oferecido. Já os componentes *Tuner*, *DataProcessing*, *ContextManager*, *InputOutput* (IO) e *InteractiveChannel* (IC) do Ginga-NCL, são equivalentes, respectivamente, ao *Tuner*, *DataProcessing*, *ApplicationManager*, *Input* e *ReturnChannel* do Ginga-J. As funcionalidades do módulo *Player* do Ginga-NCL são representadas pelo *Media* e *Player* do Ginga-J. Já o *TSParser* do Ginga-NCL, é representado pelos módulos *Demux* e *SI* do Ginga-J. O módulo *System* do Ginga-NCL é implementado internamente no GingaJ. A principal justificativa de representar as funcionalidades do Ginga-J com mais módulos, é permitir melhor coesão e, conseqüentemente, maior flexibilidade de extensão e manutenção do código. Por exemplo, para permitir alterações mais pontuais, como o padrão de informações de serviço suportado. Outra diferença importante está na implementação do mecanismo de gerenciamento dos módulos. No Ginga-NCL isto é implementado pelo *ContextManager* e *UpdateManager* (*UM*) e no Ginga-J são definidos um modelo e ambiente de execução de componentes de *software* (FlexCM). A Tabela 1exibe essa comparação.

Tabela 1- Comparação entre Ginga-J e Ginga-NCL para dispositivos fixos em relação ao modelo de componentes

	Ginga-J	Ginga-NCL para fixos
Modelo de Criação	Injeção de dependências	Fábrica
Conhecimento de arquitetura	No nível de modelo	No nível de código
Ciclo de vida	Criação, inicialização, pausa e destruição	Criação e destruição
Configuração de componentes	Modelo padronizado	Ausente

No que se refere aos critérios de avaliação, observamos que o modelo do Ginga-NCL utiliza uma abordagem de fábrica de objetos, impondo que o conhecimento de arquitetura esteja espalhado pelo código fonte do sistema. Esta característica limita a flexibilidade com a qual a arquitetura pode ser instanciada. Além disso, a falta de uma padronização na forma de configurar os componentes impede um gerenciamento efetivo

dos módulos do sistema. Dessa forma, acredita-se que o modelo utilizado na implementação do Ginga-J atende melhor requisitos de modularidade, manutenibilidade e reuso do projeto e implementação do código do Núcleo Comum.

A implementação FlexTV foi realizada pelo mesmo grupo do Ginga-J e a proposta atual é uma evolução da mesma em dois pontos: (1) funcionalidades (novo conjunto de APIs Java baseado no JavaDTV) e (2) arquitetura (adoção de um modelo e ambiente de execução de componentes).

OCAP-RI[32] é uma proposta de implementação de *middleware* imperativo baseado no padrão americano de TV Digital a cabo. Ela oferece menos funcionalidades que o Ginga-J, uma vez que as APIs Java do OCAP não oferecem suporte a múltiplos dispositivos e usuários, gerenciamento dos fluxos multimídia e mensagens assíncronas. Outra diferença é o seu projeto de arquitetura (Figura 9), que é dividida em: (i) OCAP Java - APIs Java disponível para as aplicações e definido pelo padrão americano de TVD; (ii) *JVM* e *OCAP Native* - máquina virtual Java e códigos nativo específico para implementar funcionalidades da *OCAP Java*; (iii) MPE *(Multimedia Platform Extensions*) - camada que abstrai a plataforma de hardware para a JVM e o código OCAP Native); (iv) MPEOS *(Multimedia Platform Extensions Operating System)* - implementa código dependente de plataforma oferecendo serviços para a MPE, ou seja, é o código que é portado para cada plataforma e; (iv) *RI Platform* - representa o sistema operacional e o *hardware* que executa o *middleware*. A MPE e MPEOS são equivalentes ao conjunto de componentes do Núcleo Comum do Ginga-J, onde MPE é representada pelas interfaces do *Controller* e MPEOS pelas implementações internas de cada componente. Como já citado, tal característica facilita o porte da máquina de execução Java para diferentes plataformas. Porém, MPE e MPEOS são implementadas utilizando linguagem C e não utilizam nenhum modelo e ambiente de execução de componentes. Assim, a arquitetura do OCAP-RI não oferece uma boa divisão modular das funcionalidades, dificultando o reuso e flexibilidade do código.

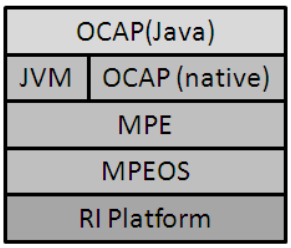

Figura 9 - Arquitetura da OCAP-RI

Com base nos pontos discutidos nesta seção, pode-se entender as principais divergências acerca da implementação do Ginga-J em relação a outras propostas. A primeira diferença está relacionada ao modelo de programação e conjunto diferentes de funcionalidades oferecidas por um ambiente imperativo em relação às ambientes declarativos ou ambientes imperativos baseados no GEM. A segunda diz respeito à arquitetura, que procurou atender melhor o reuso, manutenção e flexibilidade do código. Tal aspecto é importante para uma implementação de referência, uma vez que a mesma fornece um ponto de partida e pode ser adaptada a várias plataformas por fabricantes e outros desenvolvedores.

7. CONSIDERAÇÕES FINAIS

Este trabalho descreve a implementação de referência do Ginga-J, *middleware* imperativo do SBTVD. O desenvolvimento foi baseado na especificação JavaDTV e numa arquitetura baseada em componentes. Como forma de validação da proposta, a arquitetura foi instanciada para o ambiente Linux em um computador pessoal. Foram implementados os principais pacotes Java da norma Ginga-J, através da integração de funcionalidades básicas do ambiente JavaME e implementações de funcionalidades específicas para TV Digital (Núcleo Comum Ginga-J).

O projeto e implementação da arquitetura usando desenvolvimento baseado em componentes trouxe uma série de benefícios, entre eles: (i) conhecimento de arquitetura em nível de modelo; (ii) facilidades na configuração dos componentes individuais; (ii) configuração do sistema como um todo e (iv) possibilidade de gerenciar os componentes facilitando também a execução de testes de unidade e integração de diferentes porções da arquitetura. Tais aspectos são fundamentais para uma implementação de referência.

Como resultado dessa experiência, vários trabalhos já vem sendo realizados, dentre os quais: (i) porte da máquina de apresentação do Ginga-NCL da PUC-Rio para o Núcleo Comum; (ii) desenvolvimento de ferramentas para gerenciamento da arquitetura e geração de diferentes versões de middleware; (iii) definição de um modelo de testes para middlewares para TVD.

8. REFERÊNCIAS

[1] Peng, C. 2002. “Digital Television Applications”, Tese (Doutorado) – Helsinki University of Technology, Espoo.

[2] Morris, S. Smith-Chaigneau, 2005. A. Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV. Focal Press,

[3] ABNT NBR 15606-2. 2007 - Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital – Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações.

[4] ABNT NBR 15606-4. 2010. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais..

[5] ABNT NBR 15606-5; 2008. “Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital Parte 5: Ginga-NCL para receptores portáteis – Linguagem de aplicação XML para codificação de aplicações”. NBR 15606-5.

[6] Leite, L. E. C., et al. 2005. 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.

[7] Soares, L. F. G. 2006. MAESTRO: The Declarative Middleware Proposal for the SBTVD. Proceedings of the 4th European Interactive TV Conference. 2006

[8] Souza Filho, G. L. de, et al. Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. Journal of the Brazilian Computer Society. 2007, Vol. v12, pp. 47-56.

[9] Soares, L. F. G., et al. 2007 Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System. Journal of the Brazilian Computer Society.Vol. v12, pp. 37-46.

[10] ITU J.200. 2001. ITU-T Recommendation J.200: Worldwide common core – Application environment for digital interactive television services.]

[11] JavaDTV API.. Java DTV API 1.3 Specification, Sun Microsystems, 2009. Disponível em: http://java.sun.com/javame/technology/javatv/. Acesso em: 01/05/2010.

[12] JavaTV API. 2008. “Java TV Specification 1.1 - JSR 927”. Sun Microsystems. Disponível em URL: http://jcp.org/en/jsr/detail?id=927

[13] ETSI TS 102 819 V1.3.1 (2005) “Digital Video Broadcasting (DVB): Globally Executable MHP (GEM) Specification 1.0.2”. European Telecommunications Standards Institute, TS 102 819 V1.3.1.

[14] ARIB STD-B23. 2006 “Application Execution Engine Platform for Digital Broadcasting”. ARIB Standard B23.

[15] JCE 1.0.1. 2006. “Java Cryptography Extension - Security Optional Package Specification v1.0.1 – JSR 219”. Sun Microsystems. Disponível em URL: http://jcp.org/en/jsr/detail?id=219

[16] JSSE 1.0.1. 2006. “Java Secure Socket Extension – Security Optional Package Specification v1.0.1 – JSR 219”. Sun Microsystems. Disponível em URL: http://jcp.org/en/jsr/detail?id=219

[17] SATSA 1.0.1. 2007. “Security and Trust Services API for J2ME – JSR 177”. Sun Microsystems, Disponível em URL: http://jcp.org/en/jsr/detail?id=177

[18] CDC 1.1. 2008. “Connected Device Configuration 1.1 - JSR 218”. Sun Microsystems, Disponível em URL: http://jcp.org/en/jsr/detail?id=218

[19] FP 1.1. 2008. “Foundation Profile 1.1 - JSR 219”. Sun Microsystems, disponível em http://jcp.org/en/jsr/detail?id=219

[20] PBP 1.1. 2008. “Personal Basis Profile 1.1 – JSR 217”, Disponível em URL: http://jcp.org/en/jsr/detail?id=217

[21] Silva, L. D. N., et al. 2007. Suporte para desenvolvimento de aplicações multiusuário e multidispositivo para TV Digital com Ginga. T&C Amazônia Magazine. N. 12, pp. 75-84.

[22] Silva, L. D. N. (2008) “Uma Proposta de API para Desenvolvimento de Aplicações Multiusuário e Multidispositivo para TV Digital Utilizando o middleware Ginga”, dissertação de mestrado, Universidade Federal da Paraíba, 2008.

[23] Yaghmour , K. . Building Embedded Linux Systems. O'Reilly Media, Inc, 2003.

[24] Miranda Filho, S. et al. Flexcm - A Component Model for Adaptive Embedded Systems. 2007 In: COMPSAC IEEE International Computer Software and Applications Conference, , Beijing. p. 119-126.

[25] Caroca, C.; Tavares, T. A. . 2009 Um Modelo de Processo de Testes para os Componentes do Núcleo Comum do Middleware OpenGinga. In: IX Workshop de Teses e Dissertações do WEBMEDIA.

[26] Gamma, E.,et al. 2000. Padrões de Projeto: soluções reutilizáveis de software orientado a objetos. Bookman.

[27] ABNT NBR 15603-2. 2008. Televisão digital terrestre – Multiplexação e serviços de informação (SI) Parte 2: Estrutura de dados e informações básicas de SI.

[28] ISO 14496-20. Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF). Jun. 2006

[29] B24 Appendix 5 – Operational Guidelines for Implementing Extended Services for Mobile Receiving System. fev. 2004.

[30] Cruz, V. M., Moreno, M. F., and Soares, L. F. 2008. Ginga-NCL: implementaçao de referência para dispositivos portáteis. In Proceedings of the 14th Brazilian Symposium on Multimedia and the Web (WebMedia '08). ACM, New York, NY, 67-74

[31] Moreno, F. M. 2006. Um Middleware Declarativo para Sistemas de TV Digital Interativa. Dissertaçao de mestrado; PUC-Rio, DI.

[32] OCAP-RI (OpenCable Application Platform - Reference Implementation). Disponível em: http://ocap-ri.dev.java.net. Acesso em: 01/05/2010.