OBSERVAÇÃO

Este relatório consta de duas partes. Uma executada pelo aluno Felipe Nogueira Bárbara, e outra pelo aluno Henrique Siqueira Silva Junger, que substituiu o primeiro em março de 2008.

Atenciosamente,
Luiz Fernando G. Soares

DESENVOLVIMENTO DE APLICAÇÕES DECLARATIVAS E TESTES DE CONFORMIDADE PARA A TV DIGITAL BRASILEIRA

Aluno: Felipe Nogueira Bárbara
Orientador: Prof. Luiz Fernando Gomes Soares

1. Introdução

A produção de conteúdo para TV Digital pode ser feita seguindo dois paradigmas: o declarativo e procedural. Os programas declarativos são aqueles em que é utilizada uma linguagem declarativa para o desenvolvimento da sua entidade inicial. Já os programas procedurais são aqueles em que a sua entidade inicial é desenvolvida usando-se uma linguagem procedural. É importante mencionar, que em TV Digital, o termo procedural representa linguagens não declarativas.

As linguagens declarativas resultam de uma descrição declarativa do resultado desejado, ao invés de decompor o problema em algoritmos. Já as linguagens procedurais exigem um estilo de programação algorítmica, permitindo ao programador estabelecer todo o fluxo e controle e execução do programa. Assim, desenvolver um programa procedural exige uma maior qualificação do que desenvolver um programa declarativo.

A utilização de uma linguagem de declarativa é favorável quando o foco da aplicação casa com o foco da linguagem, já que tais linguagens provêem maior nível de abstração ao criador de conteúdo. Porém, o foco das linguagens declarativas são bastante restritos. Dessa forma, quando o foco da aplicação não casa com o foco da linguagem, o uso de uma linguagem procedural se torna necessário.

A camada de software responsável por fornecer suporte a criação de conteúdo, através de um conjunto de APIs bem definidas, é denominada de *middleware*. O *middleware* tem como objetivo esconder os detalhes de implementação e mecanismos definidos pelos padrões e sistemas operacionais e hardware dos terminais de acesso, fornecendo uma visão única de plataforma. Dessa forma, o *middleware* de um sistema de TV Digital, tem de fornecer suporte a conteúdos declarativos e procedurais.

O *middleware* Ginga do Sistema Brasileiro de TV Digital [1] adota a linguagem NCL [2] para o ambiente declarativo (Ginga-NCL), que foi desenvolvido na PUC-Rio e processa documentos NCL através do Formatador NCL. O Formatador NCL é o componente de software responsável por controlar a exibição de um documento NCL, realizando o sincronismo entre objetos de mídia declarados no documento NCL.

Os trabalhos desenvolvidos pelo bolsista durante sua iniciação científica concentram-se no Ginga-NCL.

2. Objetivos

Conforme explicado na Seção 1, o Ginga-NCL faz parte do padrão do Sistema Brasileiro de TV Digital. Dessa forma, é necessário que haja aplicações declarativas, exemplificando a compilação de boas práticas de design de programas audiovisuais interativos. A criação de aplicações declarativas é justamente um dos objetivos desse projeto.

É objetivo deste projeto, também, criar um conjunto de aplicativos especificamente desenvolvidos para exercitar os requisitos do sistema. Essas aplicações farão parte da suíte de testes de conformidade do Ginga-NCL.

3. Conversor de versão da linguagem NCL

A linguagem NCL está na versão 3.0 [3] e não possui retrocompatibilidade com a versão anterior (2.3), devido a modificações em atributos e elementos da linguagem. Entretanto, existem aplicações em NCL escritos na versão 2.3 [4]. Dessa forma, como parte da criação de aplicações declarativas, era necessário converter essas aplicações em NCL 3.0. Tal conversão é possível porque a NCL 3.0 integra todas as funcionalidades existentes na versão 2.3, além de acrescentar outras facilidades de especificação declarativa.

Para não haver um desperdício de tempo ao desenvolver esses conteúdos novamente, é importante existir uma ferramenta de software que converta um documento NCL 2.3 em um documento NCL 3.0 automaticamente. Dessa forma, foi desenvolvida uma ferramenta que é capaz de receber um documento NCL 2.3 como entrada e gerar como saída um novo documento NCL equivalente, escrito na versão 3.0.

Para a implementação da ferramenta conversora, foi necessário, inicialmente, um estudo detalhado da linguagem NCL. Como conseqüência, foi gerada uma documentação precisa dessas modificações entre as versões 2.3 e 3.0 da linguagem NCL.

A ferramenta de software foi desenvolvida utilizando a linguagem Java e o trabalho de conversão de documentos NCL concentrou-se no perfil de TV digital da linguagem. A NCL é uma linguagem organizada em módulos, agrupados em áreas funcionais, o que permite a definição de outros perfis. Por essa razão, a ferramenta foi implementada de forma modular, sendo facilmente adaptável para outros perfis, para, no futuro, ser a base para a implementação de novos conversores, conforme a linguagem evolua em suas versões, o que facilitará o processo de criação de aplicações declarativas e testes de conformidade.

A Tabela 1 ilustra as áreas funcionais e os módulos da NCL 2.3 e 3.0.

NCL 2.3 (main profile)	**NCL 3.0 (DTV profile)**
1. **Structure** a. Módulo Structure	1. **Structure** a. Módulo Structure
2. **Layout** a. Módulo Layout	2. **Layout** a. Módulo Layout
3. **Components** a. Módulo Media b. Módulo Context	3. **Components** a. Módulo Media b. Módulo Context
4. **Interfaces** a. Módulo MediaContentAnchor b. Módulo CompositeNodeInterface c. *Módulo AttributeAnchor* d. Módulo SwitchInterface	4. **Interfaces** a. Módulo MediaContentAnchor b. Módulo CompositeNodeInterface c. *Módulo PropertyAnchor* d. Módulo SwitchInterface
5. **Presentation Specification** a. Módulo Descriptor b *Módulo DescriptorBind*	5. **Presentation Specification** a. Módulo Descriptor
6. **Linking** a. Módulo Linking	6. **Linking** a. Módulo Linking
7. **Connectors** a. *Módulo XConnector* b. *Módulo CompositeConnector*	7. **Connectors** a. *Módulo ConnectorCommonPart* b. *Módulo ConnectorAssessmentExpression* c. *Módulo ConnectorCausalExpression* d. *Módulo ConnectorTransitionAssessment* e. *Módulo CausalConnector* f. *Módulo*

	CausalConnectorFunctionality g. *Módulo ConnectorBase*
8. **Presentation Control** a. Módulo TestRule b. Módulo ContentControl c. Módulo DescriptorControl	8. **Presentation Control** a. Módulo TestRule b. *Módulo TestRuleUse* c. Módulo ContentControl d. Módulo DescriptorControl
9. **Timing** a. Módulo Timing	9. **Timing** a. Módulo Timing
10. **Composite Node Templates** a. Módulo XTemplate b. Módulo XTemplateUse	
11. **Reuse** a. Módulo Import b. Módulo ReuseEntity	10. **Reuse** a. Módulo Import b. *Módulo EntityReuse* c. *Módulo ExtendedEntityReuse*
	11. **Navigational Key** a. *Módulo KeyNavigation*
	12. **Animation** a. Módulo Animation
	13. **SMIL Transition Effects** a. Módulo TransitionBase b. Módulo BasicTransition c. Módulo TransitionModifiers
12. **Metainformation** a. *Módulo MetaInformation*	14. **SMIL Meta-information** a. *Módulo Meta-Information*

Tabela 1 – Diferenças nos módulos entre as versões 2.3 e 3.0 da NCL

4. Desenvolvimento de uma aplicação interativa usando busca para gerar conteúdos relacionados

Foi desenvolvida uma aplicação audiovisual declarativa utilizando NCL, com o objetivo de integrar o ambiente de TV Digital com ferramentas de busca.

Essa aplicação consiste em um vídeo de uma entrevista do Larry King com Victoria Beckham. Nos instantes iniciais da apresentação, um botão de interatividade é exibido na tela. Caso o usuário escolha interagir através do controle remoto, conteúdos relacionados ao que está sendo mencionado na entrevista são exibidos. O usuário tem a opção de percorrer pelos diversos conteúdos ou explorar detalhadamente cada um deles, através do botão *"explore topic"*. Ao apertar esse botão, o vídeo é imediatamente redimensionado e diversas opções de interatividade são exibidas, como por exemplo, visualizar o link da enciclopédia *Wikipedia* relacionada a Victoria Beckham.

Essa aplicação exemplifica a capacidade do Ginga-NCL para os produtores de conteúdo e como utiliza diversos requisitos do Ginga-NCL, ela também foi usada como teste.

5. Testes de Conformidade

O objetivo dos testes de conformidade é exercitar os recursos do sistema, validando as implementações de acordo com o que foi definido na norma.

A linguagem NCL é especificada de forma modular, o que permite a combinação de módulos em perfis de linguagem. Cada perfil pode agrupar um determinado conjunto de módulos, criando linguagens de acordo com as necessidades dos usuários. Por essa razão, os

testes de conformidade foram projetados também de forma modular, de acordo com a especificação modular da NCL.

Os testes de conformidade das áreas funcionais *Structure*, *Layout* e *Presentation Specification* já estão desenvolvidos. Na Tabela 1 estão detalhadas as áreas funcionais e respectivos módulos da NCL 3.0.

6. Conclusões

O desenvolvimento da ferramenta conversora permitiu automatizar e acelerar o processo de criação de aplicações declarativas para o Ginga-NCL. Enquanto que na aplicação em NCL construída com a utilização de ferramentas de busca para gerar conteúdos relacionados, foi um grande teste para o Ginga-NCL, uma vez que é uma aplicação interativa que exige grande capacidade de processamento, o que torna o ambiente mais propício a falhas. E os testes que tem o objetivo de validar a implementação de áreas funcionais da NCL no *middleware*. Dessa forma, todas as atividades contribuirão para detecção e correção de erros do *middleware* Ginga-NCL.

7. Referências

[1] - SOARES, L.F.G.; RODRIGUES, R. F.; MORENO, M.F. **Ginga-NCL: the Declarative Environment of the Brasilian Digital TV System.** Journal of the Brasilian Computer Society, Number 4, Vol 12 – Sociedade Brasileira de Computação, 2007**..**

[2] **-** SOARES, L.F.G.; RODRIGUES, R.F. **Nested Context Model 3.0 Part 8 – NCL (Nested Context Language) Digital TV Profiles**. Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio, 2006.

[3] – SOARES NETO, C.S.; SOARES, L.F.G; RODRIGUES, R.F.; BARBOSA, S.D.J. **Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0 e a Ferramenta Composer**

[4] - SOARES, L.G.F.; RODRIGUES, R.F.; BARBOSA, S.D.J. **Manual de Construção de Programas Audiovisuais Interativos Utilizando a NCL 2.3.**

Um Framework para o Módulo de Sintonização do Middleware Ginga

Aluno: Henrique Siqueira Silva Junger
Orientador: Luiz Fernando Gomes Soares

Introdução

Os aparelhos de televisão foram inventados por volta da década de 20 e no início nada mais eram do que aparelhos de rádio que produziam uma imagem vermelha do tamanho de um selo postal. O serviço de "alta definição" (vale ressaltar que o termo de alta definição aqui empregado, se refere apenas a apresentação de imagens em preto e branco, não tendo nada haver com o termo utilizado para definir as melhorias descritas a seguir) apareceu pela primeira vez em 1935, mas a primeira grande transmissão só ocorreu nos jogos olímpicos de 1936.

A partir daí, diversas melhorias foram aos poucos sendo implementadas, sempre buscando uma melhor qualidade de som e imagem, sendo a última novidade a TV Digital. Ela vem para substituir a TV Analógica, com alta definição de som e vídeo, além de possibilitar a interatividade dos usuários e a multi-programação.

Porém, a migração da TV Digital para a Analógica, assim como todas as melhorias feitas anteriormente, vem sendo conduzida de forma lenta e gradual nos países que já definiram seus sistemas de TV digital, para que os impactos da transição sejam minimizados e para que a TV continue se destacando como um dos meios de comunicação mais populares no mundo atualmente.

Assim, os principais centros tecnológicos do mundo, EUA, Japão e Europa, investiram durante anos em pesquisas e estudos sobre a melhor forma de atender seus usuários. O resultado foi a criação dos sistemas DVB (*Digital Vídeo Broadcasting* - europeu), ATSC (*Advanced Television Systems Committee* - americano) e ISDB (*Integrated Services Digital Broadcasting* - japonês), os três principais padrões utilizados hoje em todo mundo.

Este estudo tem o intuito de levantar as principais características e particularidades dos padrões de *middleware* dos sistemas citados acima: o MHP (*Multimedia Home Platform* - Europa), DASE (*Digital TV Application Software Environment* - EUA) e ARIB (*Association of Radio Industries and Businesses* – Japão). Além disso, deseja-se identificar como esses sistemas influenciaram nas decisões e padronização do Sistema Brasileiro de TV Digital (SBTVD).

Um estudo sobre o *middleware* brasileiro, o Ginga, também será apresentado, com o objetivo de expor sua arquitetura e as características de cada um de seus módulos. Dessa forma, pretende-se entender melhor todo o sistema e, assim, fundamentar adequadamente os trabalhos sobre o objetivo final deste projeto: o módulo de Sintonização do *Middleware* Ginga. O módulo de sintonização do Ginga, ainda em desenvolvimento, necessita uma reengenharia para abrigar as possíveis interfaces de sintonização que ambientes de distribuição de vídeo interativo podem apresentar, como os sintonizadores de TV Digital terrestre, interfaces de rede IPTV e até mesmo interfaces de redes pessoais, como Bluetooth.

O MHP

O sistema DVB adota na camada de *middleware* o padrão MHP. Ele foi inicialmente elaborado para que o receptor recebesse o sinal por um canal unidirecional, sendo assim

necessário um canal adicional de retorno, para que houvesse comunicação do telespectador com a emissora ou terceiros. Dessa forma, três camadas foram definidas e são representadas na Figura 1.

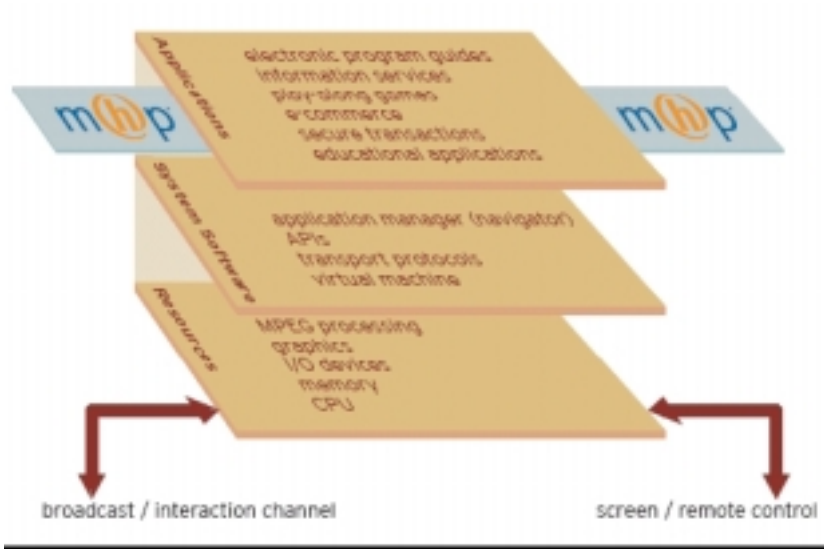

Figura 1 – Características do Middleware MHP

A camada mais baixa é a camada *Resources*, ou camada de recursos. Ela reúne os componentes de *hardware* e *software*, que servem de base para o sistema. Os mais comuns entre eles são o decodificador MPEG, memória, CPU, dispositivos I/O e gráfico. A segunda camada é chamada de *System Software*. Ela é a ponte entre a camada de recursos e as aplicações, isolando uma da outra, de forma a tornar a aplicação portátil, de modo que ela possa ser executada independente do *hardware* utilizado. Além disso, ela gerencia as aplicações, controlando seus ciclos de vida, e oferece acesso aos protocolos de transporte. Isto é feito através de um gerenciador de aplicações (também conhecido como navegador), que controla o *middleware* e todas as aplicações executadas por ele.

Por fim, a terceira camada é denominada *Applications*. Ela a camada na qual todas as aplicações interativas são executadas utilizando as APIs (*Application Programming Interface*) oferecidas pela camada *System Software*, como por exemplo o navegador citado no parágrafo anterior.

Conceito de Profiles

Outra característica importante do MHP é o conceito de *profiles* (perfis). Cada perfil especifica um conjunto de funcionalidades a serem suportadas pelo middleware, definindo, afinal, a capacidade dos *set-top-boxes* daquele perfil. Existem 3 tipos de *profiles* no MHP e eles são exibidos na Figura 2.

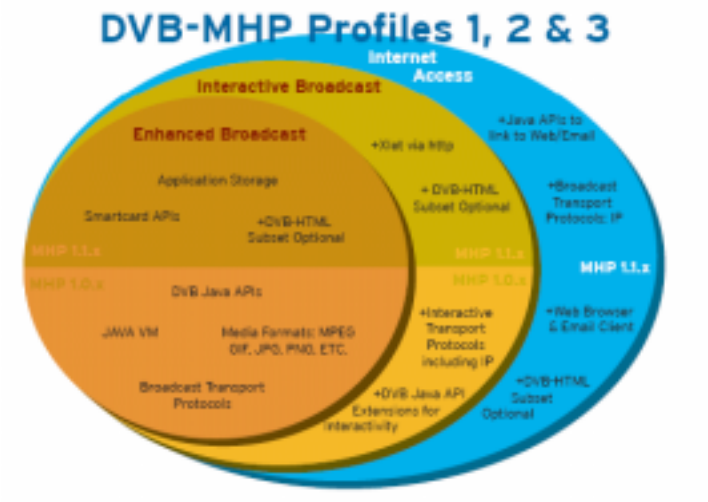

Figura 2 – Profiles do MHP

O *Enhanced Broadcast* foi o primeiro perfil a ser definido e é o mais básico dos três. Ele não fornece serviço para conexão IP e possui um canal de retorno muito limitado, para o envio de dados diretamente ao difusor.

O *Interactive TV* possui um desempenho melhor, principalmente em relação ao desempenho do canal de retorno. Uma das diferenças é que nesse *profile* é possível fazer *downloads* através do canal de retorno, já que no *Enhanced Broadcast* só era possível pelo canal de difusão. Ele também oferece um suporte melhor ao canal de retorno por meio das APIs Java existentes.

O último *profile* é o *Internet Access*, que, ao contrário dos dois anteriores, só foi definido na versão MHP 1.1. Muito mais sofisticado que os seus antecessores, ele tem um poder de processamento e memória bem maior, além de oferecer suporte a aplicações com acesso a internet. Ele também contém o componente DVB-HTML, que será mais bem discutido mais adiante.

Arquitetura, Ciclos de Vida e Plug-ins

A arquitetura do MHP é baseada em uma máquina virtual Java, ou JVM (*Java Virtual Machine*). Existem um conjunto de bibliotecas e de códigos específicos para TV Digital, que fornecem APIs genéricas. Essas APIs são as responsáveis por possibilitarem que as aplicações acessem a plataforma. Também são elas que permitem a produção de todos os serviços para a televisão com processamento local no *set-top-box*, além de gerenciar o ciclo de vida dessas aplicações, chamadas de *Xlets*.

Um Xlet pode vir previamente armazenado no *set-top-box* ou enviado pelo canal de difusão. Eles possuem 4 estados diferentes (Carregado, Pausado, Ativo e Destruído) e para variarem de um estado para o outro, os Xlets utilizam a API Java TV *javax.tv.xlet*. O ciclo de vida de um Xlet é definido na Figura 3.

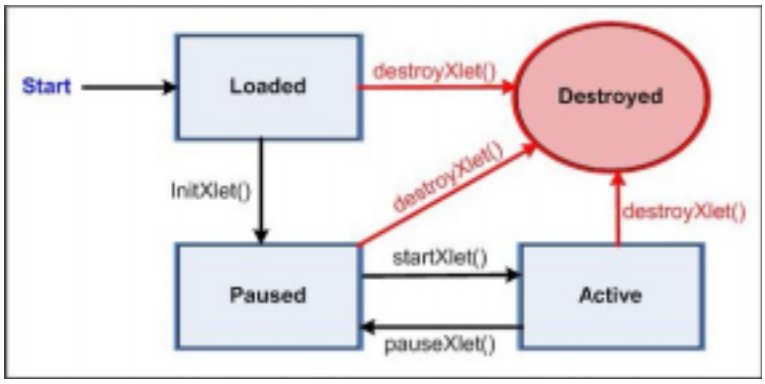

Figura 3 – Ciclo de Vida do padrão MHP

Como citado anteriormente, um novo formato das aplicações foi integrado nas versões mais novas do MHP. DVB-HTML é baseado na linguagem XHTML. O DVB-HTML suporta os padrões CSS (*Cascade Style Sheets*), ECMAScript e DOM(*Document Object Model*). Esses padrões são responsáveis, respectivamente, pelo layout das páginas HTML, por adicionar scripts imperativos a linguagem XHTML e por permitir que as estruturas do XHTML sejam manipuladas por APIs Java.

O MHP trabalha ainda com o conceito de *plug-ins*, que são um conjunto de funcionalidades que podem ser adicionadas a uma plataforma genérica, possibilitando ao middleware interpretar aplicações que não tenham sido desenvolvidas em formatos especificados pelo padrão. A escolha de qual *plug-in* instalar é feita pelo usuário e, uma vez carregado e operando na plataforma, deve se comportar da mesma forma que uma plataforma sem o uso de *plug-ins* se comportaria. Os plug-ins podem ser de dois tipos: usando o código específico da implementação MHP ou ser uma aplicação, ou seja, ele pode dar suporte à uma aplicação ou ser a própria. A Figura 4 mostra esses dois tipos.

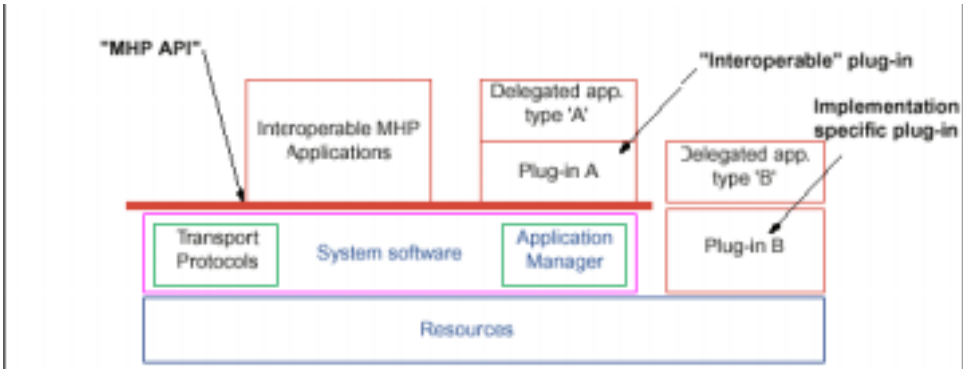

Figura 4 – Arquitetura do MHP

O DASE

O padrão escolhido para a camada de *middleware* do ATSC, inicialmente o DASE (*Digital TV Application Software Environment*). Assim como o MHP, o DASE pode ser dividido em três níveis (ou versões), como os *profiles* do *middleware* europeu. O DASE nível 1 foi a primeira versão a ser lançada e é responsável por dar suporte a aplicações de função local na TV, ou seja, ela não possui um canal de retorno para que o usuário envie dados para a operadora. Também nessa versão foi formalizada a especificação do padrão, que foi dividida em 8 partes. A Figura 5 ilustra essa especificação.

Parte	Título
Parte 1	Introdução, Arquitetura e Facilidades em comum
Parte 2	Aplicações Declarativas e Ambiente
Parte 3	Aplicações Procedurais e Ambiente
Parte 4	API
Parte 5	Recursos de Fontes Portáteis
Parte 6	Segurança
Parte 7	Transmissão de Aplicativos
Parte 8	Conformidade

Figura 5 – Divisão do DASE em partes

A segunda versão do *middleware*, ou DASE nível 2, além de manter todas as funções da primeira versão, oferece melhorias em relação ao DASE nível 1. Uma delas é o fato do segundo nível já possuir um canal de retorno possibilitando ao usuário, além de receber, também enviar dados. Além disso, ela também oferece um *framework* de segurança dos dados trocados entre o usuário e a prestadora de serviços. Por fim, temos a terceira versão. O DASE de nível 3 veio apenas para unir Internet e televisão, oferecendo acesso total a *web* e proporcionando uma interatividade pela rede e melhores serviços.

Arquitetura DASE

O padrão define que as aplicações DASE podem ser de duas maneiras, hipermídia ou programas compilados, ou seja, podemos classificar as aplicações como declarativas ou imperativas. Assim, a arquitetura do *middleware* foi organizada conforme a Figura 6.

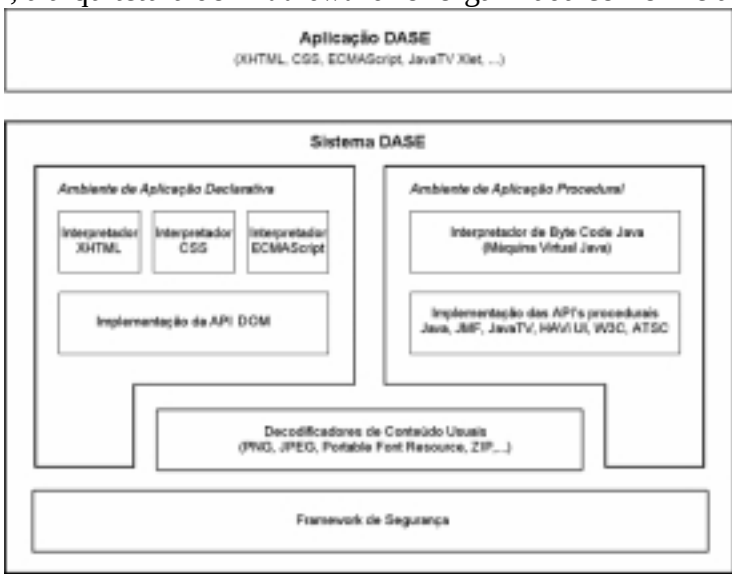

Figura 6 – Arquitetura do DASE

A Figura 3.4 mostra que o sistema DASE foi dividido em quatro módulos. O primeiro é o Ambiente de Aplicação Declarativa, que fornece suporte a aplicações declarativas, definidas numa linguagem chamada XDML (*Extensible DTV Markup Language*) e têm como objetivo a apresentação de dados de forma estática, permitindo o uso de tecnologias web, tais como linguagens baseadas em *tags*, XHTML, CSS e DOM.

O segundo módulo é o Ambiente de Aplicação Imperativo, que é o responsável por executar aplicações que são implementadas por alguma linguagem de programação

imperativa, no caso do DASE é Java. Essas aplicações são capazes de processar tarefas bem mais complexas que as declarativas. O padrão americano oferece suporte para diversas tecnologias em Java, como, por exemplo, o Java TV. É importante salientar, que embora os padrões DASE e MHP sejam muito parecidos, eles não são compatíveis, ou seja, aplicações desenvolvidas para o padrão americano não funcionam no europeu.

Por fim, temos os dois últimos módulos, o Decodificador de Conteúdo e o *Framework* de Segurança. O primeiro atende às necessidades dos dois módulos anteriores, dando suporte adicional às aplicações dos dois tipos de aplicações. O segundo, como citado anteriormente, foi adicionado na segunda versão do padrão e é responsável por criptografar os dados transmitidos entre o usuário e o provedor.

Ciclos de Vida e Apresentação

Assim como no MHP, no DASE também existe um modelo que representa o ciclo de vida das aplicações e ele se chama *DASE Content Model*. A Figura 7 mostra um diagrama de estados de uma aplicação.

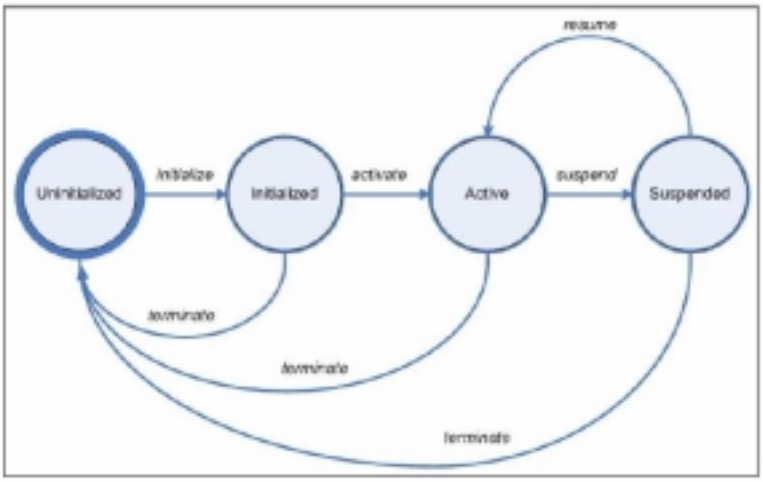

Figura 7 – Ciclo de Vida de aplicações DASE

Podemos perceber outra semelhança com o padrão europeu, no diagrama americano também existem 4 estados. O primeiro (e também o último) estado de todas as aplicações é o *Uninitialized*. Enquanto estiverem neste estado, as aplicações não consomem recursos do ambiente de execução. Em seguida temos o *Initialized*, que é o estado em que a aplicação já está apta a consumir qualquer recurso que necessite. Essa transição só ocorre se houver uma resposta positiva do carregamento da aplicação.

O estado *Active* é o estado em que a aplicação está realmente executando. Todas as aplicações devem estar associadas a uma *thread*. Como o sistema deve oferecer suporte para que várias aplicações ocorram ao mesmo tempo, mas apenas uma aplicação pode estar no modo *Active* por vez, sempre que uma aplicação for colocada em *Active* as outras devem ser passadas para *Suspended*. Nesse estado, as aplicações não consomem recursos, estão apenas na espera de serem ativadas novamente.

Por fim, existe um modelo que define como as aplicações são distribuídas, recebidas e processadas, chamado de *DASE Enviroment Model*. Os elementos que definem o DASE Enviroment Model devem oferecer suporte a entrada de dados por diversos tipos de dispositivos, tais como controle remoto, teclado e mouse; suporte a áudio, vídeo e conteúdo visual gráficos em tempo real, com resolução mínima de 640X480 e modelos de cores de 8, 16, 24 e 32 bits; e suporte a um modelo de display, organizado em planos sobrepostos que incluem um plano de background, um plano de vídeo, um plano gráfico e um plano para cursores, exatamente nessa ordem, como mostra a Figura 8.

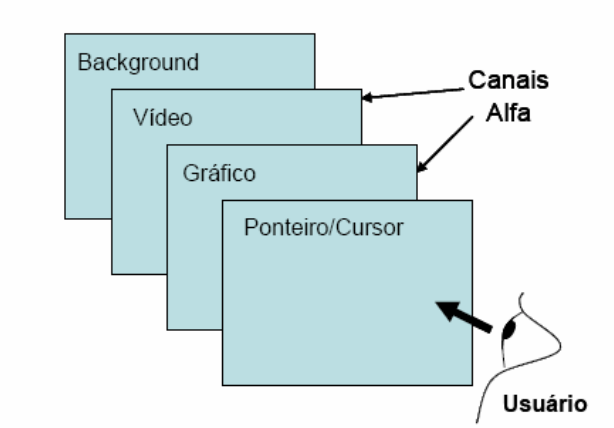

Figura 8 – Modelo de Display do middleware DASE

OCAP e GEM

Paralelamente ao desenvolvimento do DASE, a CableLabs começou a desenvolver o OCAP (*Open Cable Application Platform*), um padrão de *middleware* que atenderia as necessidades das TVs a cabo dos EUA. Este padrão começou a ser desenvolvido em cima do MHP e muitos conceitos que eram apropriados foram aproveitados.

A partir daí, a CableLabs requisitou ao DVB um estudo sobre quais as partes do MHP, que eram exclusivas do DVB, poderiam ser removidas do *middleware*, mantendo a compatibilidade entre os padrões. Com isso, o DVB criou a harmonização GEM (*Globally Executable MHP*), um padrão cujo objetivo é unificar os padrões de middleware, tornando-os mais compatíveis, a partir de um subconjunto de funcionalidades e APIs em comum do MHP.

Com isso, outros middlewares começaram a utilizar os estudos do GEM para fins de compatibilidade. Entre os países que adotaram o GEM, podemos citar o Japão com a especificação ARIB STD-B23, os EUA com o lançamento do ACAP (*Advanced Common Aplication Plataform*), um padrão de *middleware* que deveria ser uma base comum para todos os meios de transmissão no sistema americano, e, por fim, o Ginga, o middleware brasileiro.

O ARIB

O ARIB (*Association of Radio Industries and Businesses*) é a camada de *middleware* do ISDB e possui o mesmo nome da organização que define seu conjunto de características. O *middleware* japonês, diferentemente do americano e do europeu, foi definido inicialmente apenas com o ambiente declarativo, tendo o ambiente imperativo sido incorporado mais tarde. Com isso, a arquitetura do *middleware* é vista na Figura 9.

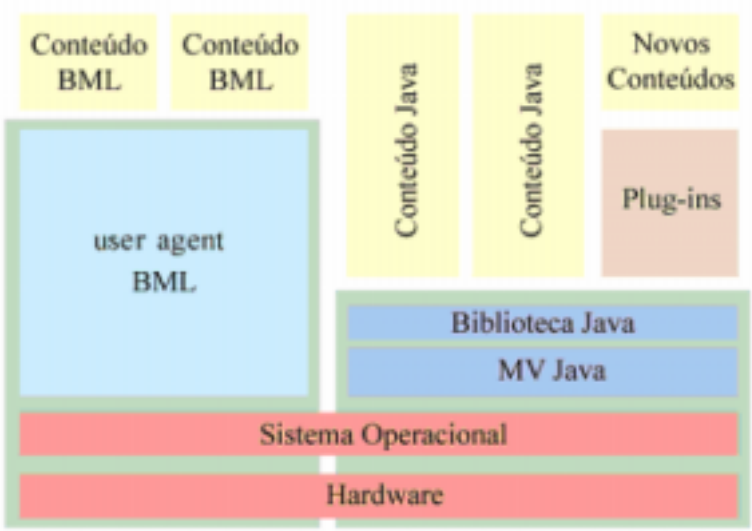

Figura 9 – Arquitetura do middleware ARIB

Ambientes Imperativo e Declarativo

Assim como em todos os *middlewares* apresentados até o momento, o ambiente imperativo do ARIB não é diferente e utiliza a tecnologia Java como base. Isso acontece porque, como foi mencionado anteriormente, o ambiente imperativo só foi adicionado ao ARIB posteriormente, na especificação ARIB STD-B23. Essa especificação foi criada baseando-se no GEM 1.0 e do DVB MHP 1.0, tendo como principal objetivo estabelecer uma compatibilidade do ARIB com os *middlewares* já existentes.

O ambiente declarativo, porém, é especifico do padrão japonês e utiliza uma linguagem desenvolvida pelo próprio ARIB, chamada BML (*Broadcast Markup Language*). Essa linguagem, assim como nos casos americano e europeu, também é baseada em XHTML, dando suporte à CSS, ECMAScript e DOM, tendo sido modificada para melhor atender a demanda do mercado japonês.

Tipos de Transmissão de Dados

O BML foi definido na especificação ARIB STD-B24, no volume 2. Essa especificação foi dividida em três volumes: *Data Coding System*, *XML – Based Multimedia Coding System* e *Data Transmission System*. Ela define que todos os serviços de dados são transmitidos em um "fluxo empacotado" (*Transport Stream – TS*), que podem ser de três tipos:

- Um sistema de transmissão que é especificado como fluxo de dados, onde o armazenamento de pacotes é utilizado como fluxo de pacotes no PES (*Packetized Elementary Stream*);
- Um sistema conhecido como carrossel de dados, onde os dados são armazenados no primeiro *download* e podem ser reutilizados sempre que necessário;
- E o terceiro que só será especificado quando for necessário, como uma expansão do padrão.

Funções do Receptor:

O receptor deve possuir funções como: recepção, apresentação, comunicação (interatividade) e funções básicas de um receptor normal de TV. Além de receber, o receptor

deve ser capaz de armazenar os dados e para isso, ele separa os vídeos (que devem ser armazenados em dispositivos de maior capacidade) dos dados (que podem ser armazenados em um dispositivo de menor capacidade).

Todo este processo de transmissão e recepção pode ser dividido em três etapas. A primeira é a decodificação dos dados em dados monomídia, ou seja, os caracteres, vídeos, imagens e áudio são separados para serem transmitidos por fluxo de dados ou carrossel de dados. A segunda etapa é a recepção desses dados monomídias e a junção deles por meio de decodificadores apropriados. Por fim, tem-se a apresentação desses dados na TV e a Figura 10 ilustra a forma como isto deve ocorrer.

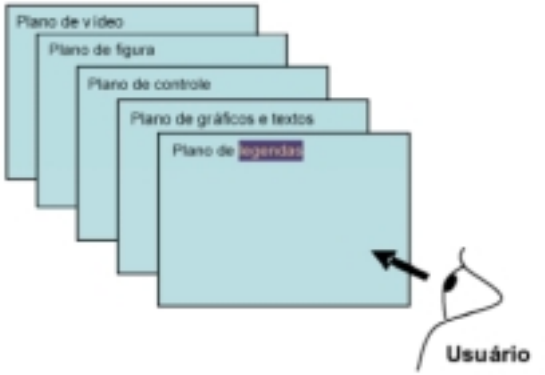

Figura 10 – Display do middleware ARIB

Cada plano de apresentação mapeia o conteúdo a ser exibido, utilizando um sistema de coordenadas X e Y, com um fator N de redução. Dessa forma, cada par de coordenadas é do tipo (X/N, Y/N), onde N pode assumir os valores 1, 1.5 e 2. No primeiro caso, dizemos que há um mapeamento 1:1 e o sistema suporta 1920 X 1080 posições. No segundo caso 1280 X 720 e no terceiro 960 X 540. É dessa forma que o ARIB altera a resolução da imagem. A Figura 11 mostra mais detalhes desse sistema.

PLANO	ESPECIFICAÇÃO
VÍDEO	1920 X 1080 X 16 – Y, Cr, Cb (4:2:2) 8 bits
FIGURA	1920 X 1080 X 16 – Y, Cr, Cb (4:2:2) 8 bits
CONTROLE	1920 X 1080 X 1 – 1 bit de controle
TEXTO & GRÁFICO	1920 X 1080 X 24 – Y, Cr, Cb (4:4:4) 8 bits - composição α em 256 valores
LEGENDAS	1920 X 1080 X 8 – 8 bits para endereçamento de mapa de cores - composição α em 256 valores

Figura 11 – Sistema de controle do tamanho da imagem

SBTVD e Ginga

Em 1999, a Anatel (Agência Nacional de Telecomunicações) estabeleceu um termo de cooperação técnica com o CPqD (Centro de Pesquisa e Desenvolvimento de Telecomunicações) e deu início a avaliação técnica e econômica sobre a melhor forma de transmitir TV digitalmente no país. Em 2003, foi fundado o comitê do SBTVD (Sistema Brasileiro de Televisão Digital), que ficaria responsável por essa escolha.

A partir daí, este comitê, juntamente com as companhias de comunicação e as universidades brasileiras, iniciaram diversos estudos sobre qual seria, entre os sistemas

anteriormente citados, o mais adequado para atender a realidade brasileira, em termos geográficos, de mercado e sócio-econômicos.

O padrão americano foi o primeiro a ser descartado, pois ele não dá suporte para a transmissão para sistemas portáteis. Sendo assim, restaram dois padrões, o europeu e o japonês. Depois de diversos estudos e testes, chegou-se a conclusão de que o padrão japonês era o que melhor atendia as características do mercado brasileiro.

No que se refere à camada física, ou seja, as camadas de Transmissão e Modulação dos dados, o SBTVD (também conhecido como ISDB-TB), é igual ao padrão japonês, utilizando o sistema MPEG-2 para transmissão dos dados e a técnica de modulação COFDM.

No entanto, as camadas de cima foram alteradas, buscando a utilização de tecnologias mais avançadas e especialmente desenvolvidas para atender o mercado brasileiro. Na camada de codificação, o SBTVD opta por utilizar o padrão H.264 para compressão de vídeo, também conhecido como uma das normas que compõem o MPEG-4. Essa técnica permite manter a mesma qualidade da imagem gerada pelo MPEG-2, porém reduzindo significativamente a taxa de bits.

Já na camada de *middleware*, o SBTVD resolveu utilizar o Ginga. Este *middleware* é uma tecnologia brasileira e resultado do trabalho conjunto entre a PUC-Rio e UFPB. O Ginga, assim como todos os outros *middlewares* anteriormente citados possui uma camada declarativa, chamada Ginga-NCL e desenvolvida pela universidade carioca, e uma camada imperativo, conhecida como Ginga-J e desenvolvida pela universidade paraibana. Estas camadas foram a evolução de trabalhos desenvolvidos durante anos e eram chamados de FlexTV e MAESTRO. Além disso, existe uma camada de menor nível, que é responsável por dar suporte as camadas declarativas e imperativos, oferecendo uma interface destas camadas com o sistema operacional, chamada de Ginga-CC. Estas camadas serão mais bem estudadas nos próximos capítulos e a Figura 12 mostra como o middleware Ginga está dividido.

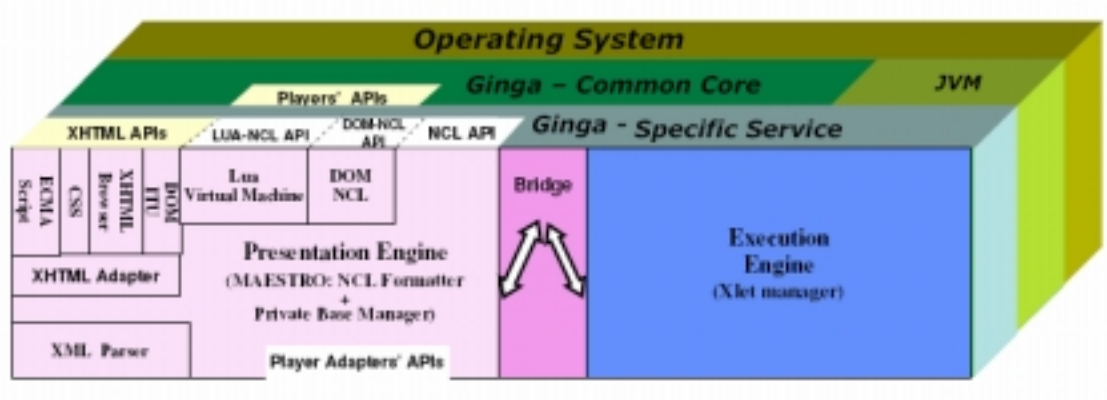

Figura 12 – Arquitetura do middleware Ginga

Ginga Declarativo

O Ginga-NCL é a parte declarativa do *middleware* brasileiro, ou seja, é a responsável processar e apresentar as aplicações que são baseadas em uma linguagem desenvolvida pela própria PUC-Rio e batizada de NCL (*Nested Context Language*). Essa linguagem não mistura a definição do conteúdo de um documento com sua estruturação, não definindo assim nenhum objeto de mídia. O que o NCL faz é definir um descritor, que funciona como uma espécie de cola e mantém todos estes objetos juntos durante uma apresentação. Com isso, faz-se necessário adicionar ao Ginga ferramentas que tratem os objetos de mídia.

Outros objetos que devem ser considerados são os baseados em XHTML, que são tratados como um caso particular de objetos de mídia.. Dependendo do *browser* escolhido como exibidor de mídia XHTML, pode-se haver compatibilidade com os outros *middlewares* existentes no mercado. É importante observar, que o *browser* que dá suporte aos objetos

baseados em XHTML, pode também dar suporte a linguagem imperativo ECMAScript, sendo, no entanto, desencorajado o uso do XHTML para a definição de relacionamentos temporais, de forma a não perder a independência das camadas do Ginga. Além do ECMAScript, outros objetos imperativos podem ser tratados pelo NCL como objetos de mídia. É o caso do Xlet, que inclusive faz parte da ponte entre o ambiente declarativo e imperativo.

Outro objeto que também faz parte da ponte entre os dois ambientes, tendo assim suporte em Ginga, são os objetos Lua. A linguagem Lua tem diversas características que a fazem dela ideal para conFiguração, automação e prototipagem rápida. Sendo assim, sua máquina virtual foi acoplada ao Formatador NCL, permitindo às aplicações NCL trocar informações com programas Lua. Além disso, outra vantagem de se utilizar Lua é que ela foi concebida na PUC-Rio e tem grande aceitação mundial, tornando-se ideal para a utilização no ambiente declarativo.

Por fim, temos o mais importante dos módulos do Ginga-NCL. Já citado anteriormente, o Formatador NCL é responsável por controlar as apresentações NCL, mantendo a relação de sincronismo entre os objetos de mídia. Enquanto um objeto está sendo exibido, um evento pode ser gerado. Por exemplo, durante a exibição de um vídeo, pode haver a seleção de um botão que dispara uma imagem e um texto. Como esses objetos são tratados por ferramentas específicas de cada um, é papel do Formatador NCL reconhecer esse novo evento e avisar as outras ferramentas que devem exibir seus arquivos. Para que isto ocorra, uma API padrão é definida, fazendo com que todos as ferramentas de exibição padrão, obedeçam ao Formatador NCL. De forma similar, sempre que um *browser* é acoplado pelo fabricante, este deve ser adaptado para que também obedeça ao Formatador NCL, ficando assim compatível com o Ginga.

Tudo isso permite que a linguagem NCL seja extremamente flexível e adaptável, atendendo assim a nova demanda que surgiu nos últimos anos, em que os padrões buscam, cada vez mais, compatibilidade entre eles.

Ginga Imperativo

A camada imperativo do Ginga é chamada Ginga-J. Esta camada é a responsável por executar e controlar todas as APIs implementadas em Java e o seu principal componente é o JVM (*Java Virtual Machine*). Entretanto, assim como na camada declarativa, as APIs imperativos também podem utilizar aplicações com recursos declarativos, como, por exemplo, conteúdos gráficos. Sabendo disso, podemos definir as APIs do Ginga de três tipos: declarativas, como aquelas que utilizam apenas recursos declarativos, imperativos, como aquelas que utilizam apenas recursos imperativos e híbridas, que utilizam recursos dos dois ambientes.

Como já foi citado anteriormente, um dos principais objetivos do Ginga é a interação com dispositivos portáteis. Mais do que apenas transmitir para esses dispositivos o Ginga-J deve também ser capaz de receber e interpretar os dados dos celulares, PDAs, controles, etc, para que haja interação com o usuário. A Figura 13 ilustra este conceito, que também é válido para o Ginga-NCL.

Figura 13 – Interação entre o Ginga e os dispositivos de transmissão e controle

Além disso, o *middleware* deve ser capaz de receber e enviar de diversos dispositivos diferentes ao mesmo tempo e ser capaz de identificar todos eles e distingui-los para que não haja conflitos. Essa demanda é especifica do Brasil e se deve ao fato de que, hoje em dia, 79.5 milhões de brasileiros possuem celulares, sendo que esse número só tende a aumentar.

Mas o Ginga-J também foi especificado de acordo com as recomendações GEM, para que houvesse compatibilidade com os outros *middlewares* já existentes no cenário mundial. Por isso, as APIs do Ginga-J podem ser divididas em três partes: Verde, Amarela e Azul. A Verde representa todas as APIs que são compatíveis com o GEM e, conseqüentemente, com os *middlewares* dos outros países. As Amarelas são aquelas estendidas das Verdes, de forma a atender os requisitos do mercado brasileiro. Por fim, existem ainda as APIs classificadas como Azuis que são totalmente inovadoras, sendo executadas somente em ambientes que possuem o Ginga. A Figura 14 mostra este modelo.

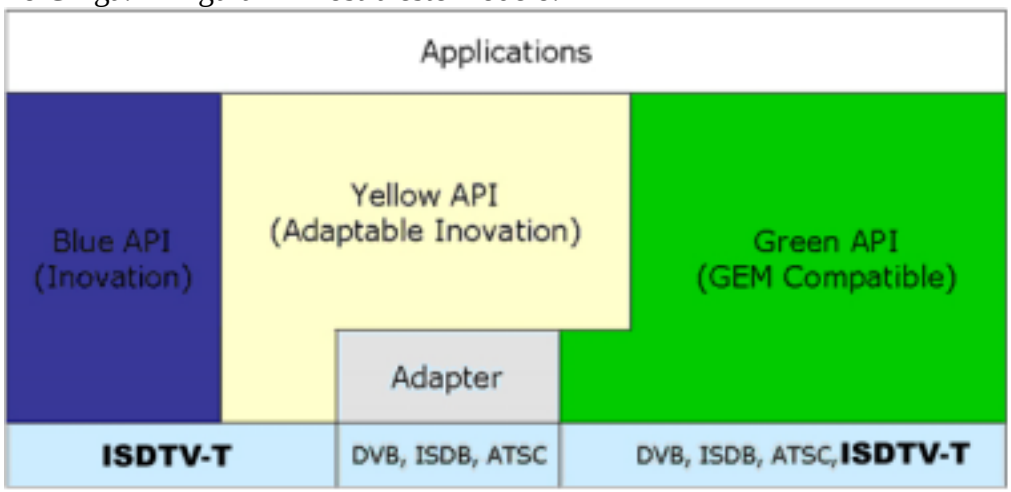

Figura 14 – Tipos de APIs do Ginga

Concluindo, um exemplo das inovações proporcionadas pelo Ginga-J inclui a definição de que as aplicações não mais possuem seu ciclo de vida preso na programação da TV, podendo ser salvas, para que sejam recuperadas depois. Com isso, professores poderiam salvar aulas, para apresentar aos seus alunos em sala.

Ginga Common Core

A última camada do Ginga é a camada de mais baixo nível, composta por treze módulos, que oferecem aos ambientes declarativo e imperativo o acesso aos recursos do hardware de TV, entre eles, o acesso aos conteúdos recebidos. A Figura 15 ilustra como esses treze módulos estão organizados na camada Common Core e serão melhor explicados, um a um.

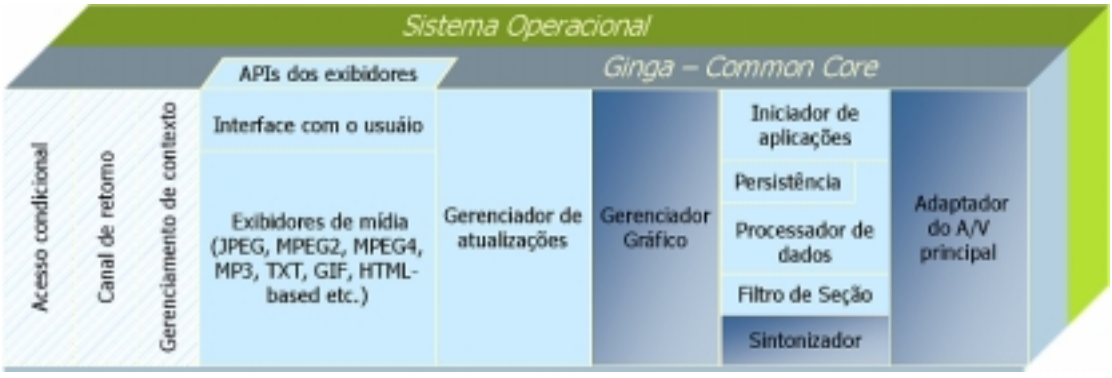

Figura 15 – Arquitetura da camada Common Core

- **Sintonizador:** É o módulo responsável por “sintonizar” o canal, escolhendo um canal físico e um dos fluxos de transporte que estão sendo enviados neste canal. É o principal objetivo deste trabalho e será melhor explicado mais a frente.
- **Filtro de Seções:** Uma vez sintonizado, o middleware deve ser capaz de acessar partes específicas do fluxo de transporte. Para isso, existe o Filtro de Seção, ele é capaz de buscar no fluxo a parte exata que as APIs necessitam para suas execuções. Funcionando exatamente como um filtro, deixando passar apenas as informações requeridas pela API.
- **Processador de Dados:** É o elemento responsável por acessar, processar e repassar os dados recebidos pela camada física. Ele também fica encarregado de notificar os outros componentes, sobre qualquer evento que tenha sido recebido.
- **Persistência:** Como foi mencionado, o Ginga é capaz de salvar arquivos, mesmo depois do processo que o criou tenha sido finalizado, para que este possa ser aberto em outra oportunidade. Este é o módulo que dá suporte para que isso ocorra.
- **Iniciador de Aplicações:** Esse módulo é o gerenciador de aplicações, ou seja, ele fica encarregado de carregar, conFigurar, inicializar e executar qualquer aplicação dos ambientes declarativo e imperativo. Ele também é responsável por controlar o ciclo de vida dessas aplicações, retirando-as quando necessário, além de controlar os recursos utilizados por essas APIs.
- **Adaptador do A/V Principal:** Com o Adaptador de A/V Principal, as aplicações conseguem enxergar o fluxo de áudio e vídeo. Isso se faz necessário quando uma aplicação precisa controlar suas ações, de acordo com o que está sendo transmitido.
- **Gerenciador de Gráfico:** Como foi visto anteriormente, os padrões de middleware definem como as imagens, videos, dados, etc, são apresentados para o usuário, gerenciando as apresentações conforme da mesma forma que é definida no padrão ARIB.
- **Gerenciador de Atualizações:** Componente que gerencia as atualizações feitas no sistema, verificando, baixando e atualizando o middleware sempre que necessário, para correção de possíveis erros encontrados em versões anteriores. Isto deve ser feito em tempo de execução, sem incomodar o uso normal da TV pelo usuário.
- **Exibidores de Mídia:** São as ferramentas necessárias para exibir os arquivos de mídias recebidos, como por o exemplo os tipos MPEG, JPEG, TXT, MP3, GIF, HTML, etc.
- **Interface com o Usuário:** Este módulo é responsável por captar e interpretar os eventos gerados pelo usuário, tais como, comandos do controle remoto e avisar aos outros módulos interessados.
- **Gerenciamento de Contexto:** É o responsável por capturar as preferências do usuário, alertando os outros componentes interessados nessas preferências. Essas informações podem ser os horários em que o usuário assiste a TV, ou bloqueio e desbloqueio de canais, entre outras.
- **Canal de Retorno:** Ele provê a interface das camadas superiores com o canal de interação (ou canal de retorno). Além disso, ele deve gerenciar o canal de retorno de forma que dados sejam transmitidos assim que o canal esteja disponível, ou “forçar” a transmissão caso o usuário ou uma aplicação tenha definido o horário exato.

- **Acesso Condicional:** Esse componente está encarregado de restringir conteúdos inapropriados recebidos pelos canais de programação, oferecendo assim segurança ao *middleware*.

O próximo capitulo visa entender melhor o componente do Sintonizador, estudando à fundo como ele está definido e implementado.

O Sintonizador:

O Sintonizador, como já foi dito anteriormente, é o módulo do Ginga responsável por sintonizar os canais, ou seja, ele é o encarregado por receber o fluxo de dados do meio físico e repassá-los para as APIs que utilizarão estes dados, que são chamadas de *listeners*. Eler se faz necessário, pois, como se sabe, o fluxo de transporte pode ser recebido por diferentes sistemas de difusão (terrestre, satélite e/ou cabo) e seus canais podem estar multiplexados de diferentes formas e como o principal objetivo do trabalho é construir um *framework* para que novas especificações sejam trivialmente inclusas nas suas funcionalidades, um estudo sobre o estado atual do módulo, se fez necessário, para entender como o Sintonizador está implementado e de que forma ele poderá ser alterado para que o objetivo final seja alcançado.

Atualmente, o Sintonizador só recebe dados de três tipos de provedores, Multicast, Unicast e arquivos locais. Isso é feito através de três classes, que se comunicam entre si para receber os fluxos e repassá-lo aos *listeners*. Essas classes são: *Tuner*, *NetworkInterface* e *DataProvider* e são discutidas a seguir.

Classe Tuner:

O Tuner é composto por 14 métodos que gerenciam toda a recepção dos dados, controlando as interfaces de redes existentes e também todos os *listeners*. A Figura 16 mostra uma parte do código do Tuner.h, com todos os seus métodos.

```
public:
    Tuner();
    virtual ~Tuner();

private:
    void initializeChannels();
    string searchMulticastAdd();
    string searchUnicastAdd();
    void createChannel(string network, string protocol, string address);
    bool listen(NetworkInterface* channel);
    void receive(DataProvider* provider);

public:
    void channelUp();
    void channelDown();

private:
    void changeChannel(int factor);
    void searchChannel(int factor);

public:
    void addListener(TunerListener* listener);
    void removeListener(TunerListener* listener);

private:
    void notifyListeners(char* buff, unsigned int val);
    virtual void run();
```

Figura 16 – Arquivo Tuner.h

A recepção é gerenciada através do método privado *initializeChannels()*, que inicia os canais de recepção (atualmente apenas os três citados anteriormente, Multicast, Unicast e arquivos locais), chamando outro método privado, o *createchannel()*, responsável por instanciar as interfaces de rede. O *initializeChannels()* é chamado no construtor. *Os métodos searchUnicastAdd()* e *searchMulticastAdd()* também utilizados pelo *initializeChannels()* ficam encarregados de retornar o *socket* dos respectivos meios de recepção. Existem ainda os

métodos *listen()* e *receive()*. O primeiro é responsável por "escutar" e receber os dados enviados pelo meio físico. Já o *receive()* é responsável por repassar estes dados aos *listeners*.

Outra função importante do Tuner é gerenciar os canais. Isso é feito através das duas variáveis *currentChannel* (canal atual) e *majorChannel* (canal principal). Estes canais são alterados pelas funções públicas *channelUp()* e *channelDown()*, que alteram o canal para cima e para baixo.

Por fim, há métodos dedicados ao gerenciamento dos *listeners*. Toda vez que uma API deseja receber dados do Sintonizador, ela se "cadastra" através do *addListener()*. Depois de receber algum dado, o Sintonizador notifica as APIs, através do *notifyListeners()* e repassa os dados para os listeners. Caso algum listener deseja parar de receber os dados, basta acionar o *removeListener()*.

Classe NetworkInterface

A classe *NetworkInterface* define as interfaces de rede que estão disponíveis. Essa classe possui cinco atributos que definem a interface de rede, conseqüentemente, como ela será tratada pelas outras classes. Esses atributos são: Id, Name, Protocol, Address e Provider.

Logo quando é instanciada, a *NetworkInterface* só não define o atributo provider. Este é escolhido pelo método público *setDataProvider()*, chamado pela própria API. Os outros métodos são todos para recuperar os atributos da interface de rede.

Classes Data Provider

Por fim, a classe abstrata DataProvider define o provedor, ou seja, como seus herdeiros receberão os arquivos. Para isso, o DataProvider define que seus herdeiros devem possuir quatro métodos obrigatórios. A Figura 17 mostra uma parte do código do DataProvider.h, com esses métodos.

```
class DataProvider {
    public:
        virtual ~DataProvider(){};
        virtual int receiveData(char* buff)=0;
        virtual bool tune()=0;
        virtual bool changeChannel(int factor)=0;
        virtual void close()=0;
};
```

Figura 17 – Código DataProvider.h

O *receiveData()* é o método responsável pela passagem dos dados recebidos aos listeners do Sintonizador. O *tune()* é o método que busca o meio físico para receber os dados, ele "sintoniza" a recepção dos dados. O *changeChannel()* altera os canais de recepção, este método está atualmente "desabilitado", pois não há sentido falar em alteração de canal nos provedores disponíveis. E finalmente temos o método *close()*, que deve finalizar a recepção dos dados. A Figura 18 ilustra a árvore atual de como estes provedores estão distribuídos.

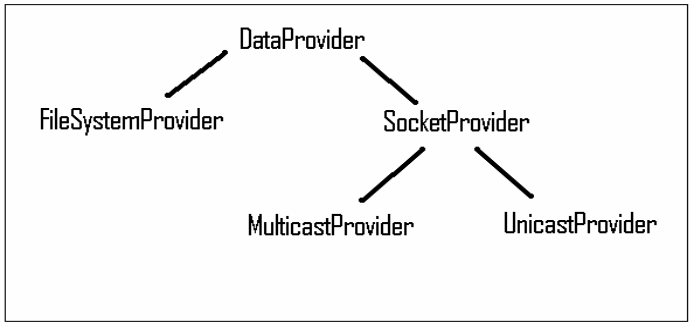

Figura 18 – Árvore dos Tipos de Provedores

Conclusão

Pode-se concluir que para alcançar os objetivos traçados, algumas mudanças no Sintonizador deverão ser feitas. Uma delas, será a expansão da árvore do DataProvider, com a inclusão de duas novas classes. A primeira será a responsável por tratar o recebimento de informações enviadas via *broadcast* e a segunda será responsável pelo recebimento de redes pessoais *bluetooth*. A nova árvore do DataProvider deverá ficar parecida com a Figura 19. Além disso o DataProvider terá seu nome StreamProvider.

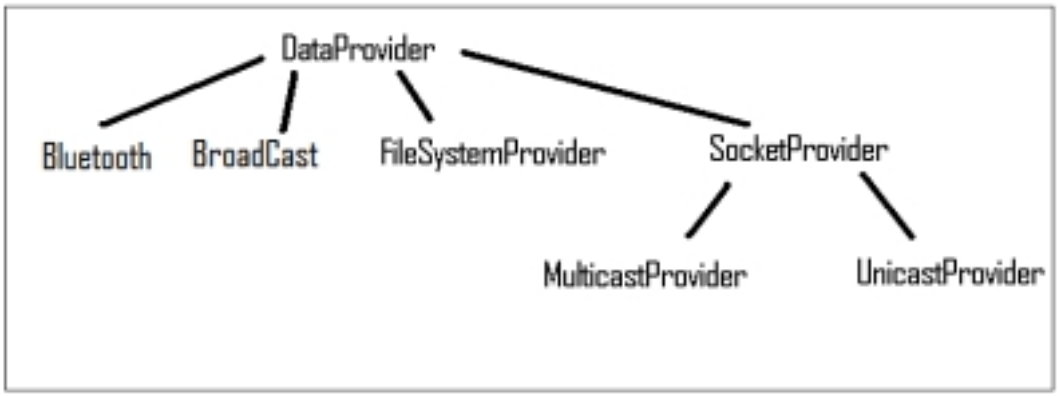

Figura 19 – Futura Árvore dos tipos de provedores

Além disso, as classes Tuner e NetworkInteface deverão ser modificadas para tratar os novos provedores de informação. Na primeira as função oferecidas por ela, deverão ser uma espécie de interface das aplicações, com as NetworkInterfaces, oferecendo formas para que as aplicações possam se tornar listeners, alterar canais, etc.

A classe NetworkInterface sofrerá também alterações na função createProvider() para tratar os novos tipos de interface de rede. Esse método é o encarregado de instanciar os provedores, por isso deve saber tratar todos eles. A Figura 20 ilustra o o código como está atualmente.

```
    if (name == "eth") {
        ip = address.substr(0, address.find(":"));
        cout << "IP = '" << ip << "'";
        portNumber = address.substr(
                address.find(":") + 1, address.length());

        cout << "PORT NUMBER = '" << portNumber << "'" << endl;

        if (protocol == "udp_unicast") {
            provider = new UnicastProvider(ip, (int)stof(portNumber));
            return true;

        } else if (protocol == "udp_multicast") {
            provider = new MulticastProvider(ip, (int)stof(portNumber));
            return true;
        }

    } else if (name == "fs") {
        provider = new FileSystemProvider(address);
        return true;
    }

    return false;
}
```

Figura 20 – Trecho do código NetworkInterface.cpp

Novamente podemos perceber, que há ifs no código, que tratam os diferentes tipos de provedores, tornando uma adição de uma nova especificação muito complexa. A solução encontrada, foi tornar a NetworkInterface abstrata, onde a árvore gerada por ela será exatamente igual a árvore do DataProvider, onde para cada tipo de NetworkInterface terá um tipo de provedor associado.

É claro que apenas estas mudanças não serão suficientes. No entanto, novas soluções serão discutidas e encontradas, assim que novos problemas forem se apresentando. Dessa forma, aos poucos o framework, irá ser moldado, até que esteja apto a ser implementado.

Referências

[1] Paes, A.; Antoniazzi, R. . – Padrões de Middleware para TV Digital. Artigo, Departamento de Engenharia de Telecomunicações, UFF. 2005;
[2] Mendes, L. L.; Fasolo, S. A. – Introdução a Televisão Digital. Artigo, INATEL, Minas Gerais. 2002;
[3] Oliveira, Carina T. – Um Estudo sobre os Padrões de Middleware para a TV Digital Interativa. Monografia Final do Curso, CEFET-CE. 2005;
[4] Bittencourt, M. M.; Araújo, T. P. – Infra-Estrutura para o Desenvolvimento de Aplicações para TV Digital Interativa. UFF. 2006;
[5] Site Oficial do Ginga - http://www.ginga.org.br/;
[6] Moreno, Márcio F. – Um Middleware Declarativo para Sistemas de TV Digital Interativa. Dissertação de mestrado, Departamento de Informática, PUC-Rio. 2006;
[7] Mendes, Luciano L. – SBTVD – Uma visão sobre a TV Digital no Brasil. Artigo, INATEL, Minas Gerais. 2007.
[8] Filho, G. L. S.; Leite, L. E. C.; Batista, C. E. C. F. – Ginga-J: The Imperativo Middleware for the Brazilian Digital TV System, Departamento de Informática, UFPB;
[9] Um Middleware Declarativo para Sistemas de TV Digital Interativa. Laboratório Telemídia, Departamento de Informática, PUC-Rio. 2007;
[10] Batista, E.; Matos, J.; Santos L. – Digital Broadcasting for Handheld Devices. Instituto Superior Técnico – Taguspark, Portugal;
[11] Site sobre TV Digital em Portugal. Disponível em: http://www.img.lx.it.pt/~fp/cav/ano2006_2007/MEEC/Trab_17/artigo_divulgacao.html
[12] Código do Ginga Common Core: Gingacc-Tuner.