2
Conceitos Preliminares

No capítulo anterior foram citados os três *middleware* para TV Digital terrestre mais populares: o europeu (MHP), o americano (ATSC) e o japonês (ISDB). Além desses, foi citado um ambiente para a execução de aplicações procedurais em sistemas de mídia empacotada: o *Blue-Ray Disc*. Tais padrões estabelecem uma série de requisitos mínimos a serem implementados por seus terminais. Neste capítulo são mostrados alguns desses requisitos relevantes para a construção de um ambiente de apresentação declarativo como uma aplicação procedural. É também feita uma breve descrição dos padrões mencionados, que constituem as plataformas para as quais o sistema proposto é destinado.

No intuito de descrever melhor as tecnologias envolvidas na construção do sistema proposto, este capítulo mostra, também, a configuração mínima de uma máquina virtual Java para uma plataforma de TV Digital.

Para cumprir seus propósitos, este capítulo está organizado da forma a seguir. Na Seção 2.1, e em suas subseções, são apresentados os principais *middleware* procedurais para TV Digital aberta. A Seção 2.2 descreve o GEM como tentativa de harmonização desses padrões de *middleware*. A Seção 2.3 mostra o padrão Blue-Ray: o primeiro padrão de mídia empacotada a implementar o GEM. E, por fim, na Seção 2.4 é apresentada a especificação mínima do ambiente Java para sistemas de TV digital, inclusive o GEM.

2.1.
Middlewares de TV Digital

Numa arquitetura de serviços para TV Digital, o *middleware* consiste em uma abstração que atua entre duas camadas bem definidas. A razão de sua existência está intimamente ligada à forma como ele interage com tais camadas.

A camada inferior provê o serviço de transporte dos dados e o acesso à utilização dos recursos do terminal. Essa camada não é alvo de uma padronização formal, podendo ser especificada de acordo com os critérios do fabricante do

receptor. Ela engloba todo hardware e softwares nativos (*drivers* de dispositivos, sistema operacional, aplicações nativas).

A camada superior é composta pelos aplicativos e serviços comuns aos usuários do sistema de TV Digital: guias eletrônicos de programação, jogos, cotação do mercado de ações, dentre outros serviços produzidos pelos provedores de conteúdo/infra-estrutura.

De forma a permitir que os aplicativos e serviços do sistema de TV Digital sejam desenvolvidos independente da camada inferior, ou seja, do fabricante, foi criada uma camada intermediária: o *middleware*.

Os *middlewares* são padrões que especificam requisitos mínimos a serem implementados pelos fabricantes de forma a permitir uma execução global de aplicações e serviços para TV Digital. Tal padronização abrange, principalmente, APIs para acesso aos recursos da camada inferior, protocolos e formatos de conteúdo a serem suportados.

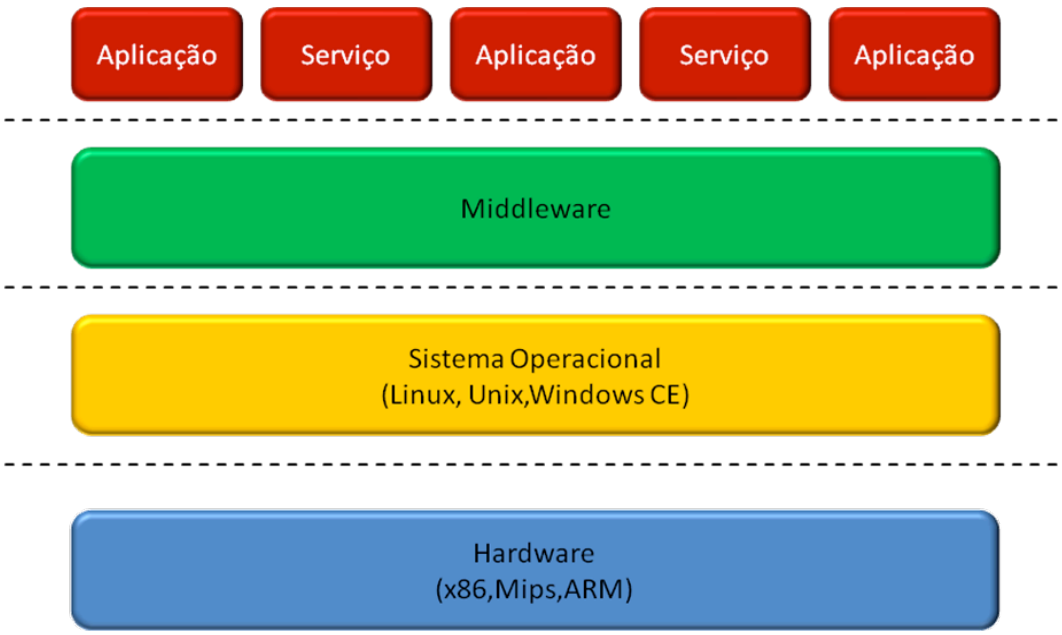

Figura 2 – Arquitetura de um sistema de TV Digital.

A Figura 2 ilustra o *middleware* exercendo seu papel de camada intermediária entre o sistema operacional e os aplicativos e serviços comuns aos usuários do sistema de TV Digital.

Os principais padrões de *middleware* para sistemas de TV Digital terrestre serão descritos nas seções a seguir.

2.1.1. O *Middleware* Europeu

O *middleware* europeu MHP (*Multimedia Home Platform*) foi desenvolvido por um consórcio de empresas denominado DVB (*Digital Video Broadcast*). Essas empresas são de áreas que incluem desde fornecedores de equipamentos até as emissoras de TV. Ao elaborar novas resoluções, tal consórcio as encaminha ao ETSI (*European Telecommunications Standards Institute* – um órgão responsável por padronizações na área de telecomunicações e ligado à União Européia) que, por fim, o estabelece como padrão. Seguindo esse processo, foram desenvolvidas, até então, duas versões do padrão MHP: o MHP 1.0 e o MHP 1.1.

No MHP são estabelecidos dois níveis de categorização de suas aplicações: o primeiro enquadra a aplicação de acordo com os perfis dos usuários para os quais elas são dirigidas, definindo sua área de abrangência e, como conseqüência requisitos de processamento e comunicação dos terminais; e o segundo enquadra as aplicações de acordo com seu tipo (procedurais, declarativas ou híbridas).

Os seguintes perfis de usuário/aplicação são definidos no padrão MHP 1.1 na ordem do mais básico ao mais avançado, sendo que cada perfil é uma extensão do perfil imediatamente mais básico:

- ***Enhanced Broadcast Profile:*** É o perfil básico, projetado para espelhar as funcionalidades fundamentais existentes em um sistema de *middleware*. Nesse perfil não existe interatividade direta com os provedores de conteúdo, pois não há canal de retorno;
- ***Interactive Broadcast Profile***: A principal característica desse perfil é permitir que aplicações possam se comunicar por meio de um canal de retorno. Esse perfil apresenta suporte para comunicação IP sobre o canal de retorno. Além disso, existe também grande suporte para interatividade com as APIs definidas;
- ***Internet Access Profile:*** Esse perfil fornece grande poder de processamento e armazenamento, permitindo acesso a conteúdo da Internet no terminal de acesso. É previsto, obrigatoriamente, a existência de um cliente de e-mail e um *browser* HTML sem, no entanto, tornar obrigatória a extensão promovida pela linguagem DVB-HTML (apresentada na próxima seção).

A Figura 3 apresenta os perfis citados ilustrando algumas das APIs e tecnologias suportadas por cada versão do MHP.

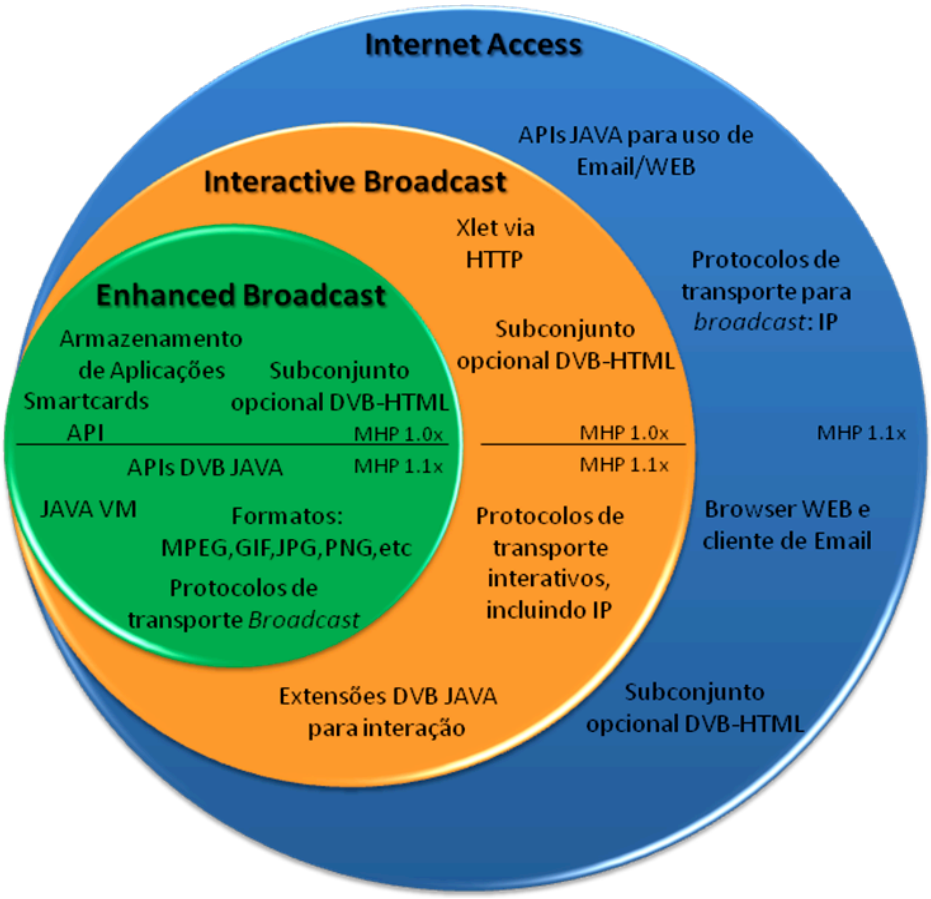

Figura 3 - Perfis MHP.

2.1.1.1. Tipos de aplicações MHP

As aplicações MHP declarativas e procedurais são, respectivamente, o DVB-HTML e DVB-J (ou DVB JAVA). As aplicações DVB-HTML não são muito difundidas pelo fato de terem sido especificadas apenas na última versão do padrão. Já as aplicações DVB-J possuem grande aceitação no mercado, pois as APIs de suporte são obrigatórias em todos os perfis.

Uma aplicação MHP, no entanto, não necessariamente precisa ser exclusivamente procedural ou declarativa. Existem ainda, como mostrado na Figura 4, as aplicações híbridas. Nessas aplicações é prevista a existência de elementos em ambas as linguagens, denso prevista uma ponte que permite a

comunicação entre as aplicações que pertençam a esses dois paradigmas diferentes.

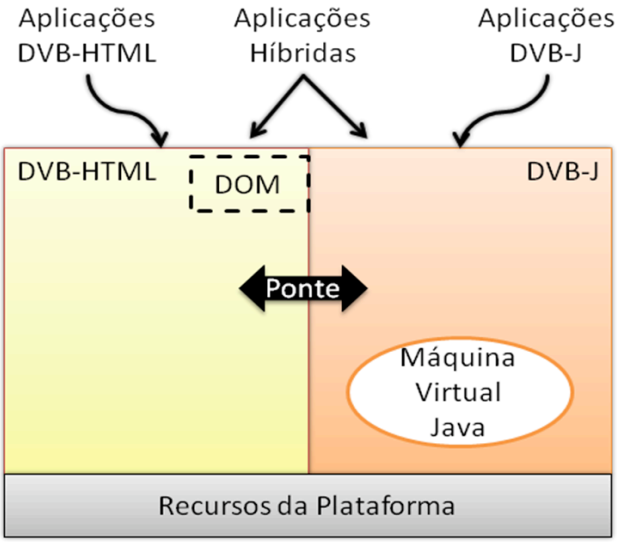

Figura 4 - Arquitetura básica do MHP

Aplicações DVB-HTML são baseadas em alguns padrões estabelecidos pelo W3C. Essas aplicações são descritas em uma linguagem que estabelece uma extensão ao XHTML adotando, em conjunto com este, os padrões, *Cascade Style Sheets* (CSS - responsável pela formatação e layout de páginas HTML), *ECMAScripts* (responsável por habilitar interatividade em documentos XHTML) e *Document Object Model* (fornece abstração que permite, através do *ECMAScript*, manipular estruturas e conteúdos de documentos XHTML).

Já as aplicações DVB-J constituem programas em código binário JAVA. Esses programas, no entanto, diferente das versões de aplicação JAVA destinada a desktops, devendo obedecer a uma série de APIs mais restritivas definidas pelo MHP (ETSI, 2005). São elas:

- **JavaTV**: Introduz uma série de funcionalidades específicas para o ambiente de TV interativa, como localização de conteúdo, controle do ciclo de vida da aplicação, controle de acesso, sintonização e informação sobre serviços. Os programas que utilizam essa API são também denominados *Xlets*.
- **JMF**: O *Java Media Framework* é uma API que gerencia o comportamento e a interação de objetos de mídia contínua. JMF é

utilizado para capturar, processar e apresentar alguns tipos de mídias contínuas.

- **HAVi *User-Interface***: é uma API que permite ao programador criar elementos para a interface com o usuário. Provê uma extensão ao pacote "java.awt", permitindo, assim, suporte a controle remoto, transparência, entre outros;
- **DAVIC**: Apresenta alguns requisitos de sistemas audiovisuais para prover interoperabilidade fim-a-fim;
- **DVB-MHP**: Essa API utiliza os serviços providos pelas classes das APIs já mencionadas (ou seja, na prática provendo a mesma funcionalidade) e adiciona serviços novos, como gerenciamento da conexão do canal de retorno, armazenamento de informações, manipulação de eventos, segurança, gerenciamento de aplicações e carregamento de *plug-ins*.

O MHP fornece dois mecanismos auxiliares que permitem a instalação de aplicações de forma persistente nos receptores dos usuários. São eles: o uso dos *plug-ins* e o *Application Storage* (armazenamento de aplicações). Nas subseções seguintes serão apresentados esses dois mecanismos.

2.1.1.2. O uso de *Plug-ins*

Plug-ins são conjuntos de funcionalidades que podem ser adicionadas a uma plataforma genérica de forma a prover a interpretação de formatos de aplicação registrados junto ao DVB. Atualmente, encontram-se registrados dois formatos de aplicações: o HTML 3.2, que também indica aplicações no formato XHTML e suas derivadas como o DVB-HTML, e o MHEG-5 (ISO, 1997). A Figura 5 mostra os dois tipos possíveis de plug-ins: nativos (na figura como *plug-in* B) e interoperáveis (na figura como *plug-in* A).

Os *plug-ins* nativos usam código específico da implementação do receptor, ou seja, código nativo. Esse tipo de código pode, inclusive, ser especificado na linguagem Java, desde que não sejam utilizadas as APIs do MHP, caso contrario,

deixam de ser considerados nativos. O processo de sinalização das aplicações[1], tanto do *plug-in* como das aplicações por ele processadas, também deverá ser nativo.

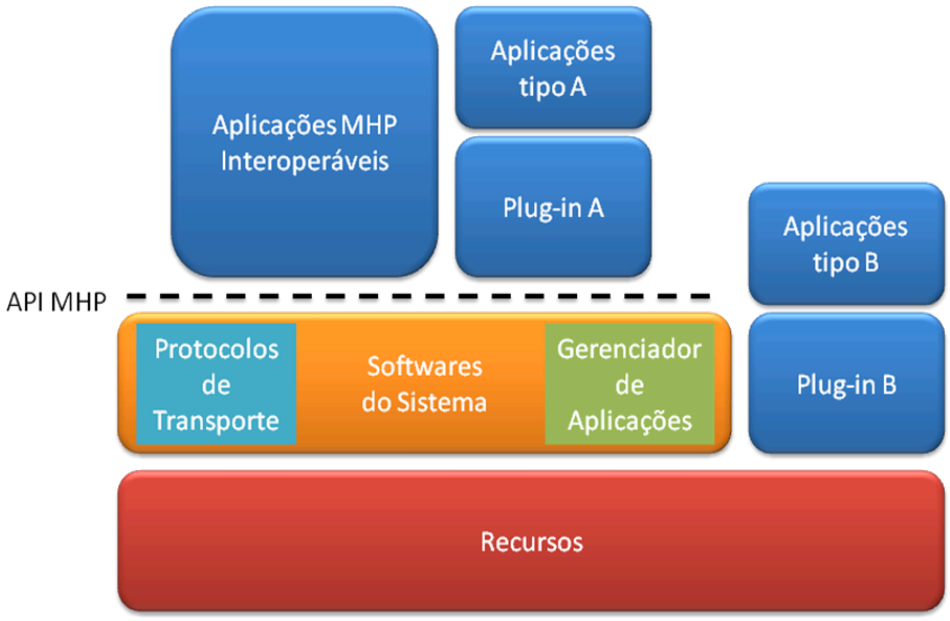

Figura 5 – Opções de implementação de plug-ins.

Já um *plug-in* interoperável é aquele que necessita apenas das APIs especificadas pelo padrão MHP, permitindo dois tipos de sinalização: uma fornecida pelo padrão e outra específica da implementação.

A sinalização padrão do MHP para *plug-ins*, prevê a existência de dois tipos de descritores a serem utilizados pela tabela AIT: um descritor de aplicação para o *plug-in* e outro para a aplicação por ele processada. Esses descritores são mostrados em mais detalhes no Apêndice B. A sinalização prevê, ainda, que o ponto de entrada do *plug-in* seja uma classe que implemente a interface `org.dvb.application.plugins.Plugin` mostrada na Figura 6. Com isso, as aplicações processadas por ele serão reconhecidas como *Xlets* e controladas pelo Gerenciador de Aplicações.

Na sinalização específica da implementação, a aplicação processada pelo *plug-in* não será reconhecida como uma aplicação MHP (*Xlet*). Com isso todo seu controle (ciclo de vida, etc.) deverá ser realizado pelo próprio *plug-in*.

[1] A sinalização de aplicações define formas de se identificar e iniciar aplicações associadas a um serviço. Ela deve também oferecer mecanismos pelos quais as difusoras possam gerenciar o ciclo de vida de suas aplicações.

```
public interface Plugin{
      plubic Xlet initApplication(AppAttributes app)
            throws InvalidApplicationException;
      public Xlet initApplication(InnerApplication app)
            throws InvalidApplicationException;
      public boolean initPlugin();
      public isSupported(AppAttributes app);
      plubic void terminatePlugin();
}
```

Figura 6 – A interface Plugin.

A interface `org.dvb.application.plugins.Plugin`, mostrada na Figura 6, visa, como mencionado, permitir ao Gerenciador de Aplicações manter o controle do ciclo de vida das aplicações sendo executadas pelo *plug-in* interoperável. Para isso, essa interface reusa o modelo de programação *Xlet* de forma a representar aplicações em formatos diferentes do DVB-J.

A classe que implementar essa interface não poderá realizar nenhuma operação que consuma muito processamento em seu construtor. Todas as rotinas de iniciação serão realizadas em seu método *initPlugin*. Esse método será chamado sempre antes do método utilizado pelo Gerenciador de Aplicações para iniciar a aplicação (*initApplication*). No método *initApplication* serão passados, ainda, os parâmetros do descritor da aplicação, presente na tabela AIT, para que a aplicação possa ser tratada pelo *plug-in*. O método *terminatePlugin* será chamado ao término da execução da aplicação.

A grande vantagem introduzida por essa facilidade é o fato de uma aplicação de conteúdo genérico ser sinalizada para os receptores e interpretada por eles como uma aplicação MHP (*Xlet*). Isso contribui para a adoção do padrão DVB-HTML para aplicações declarativas, permitindo sua introdução como *plug-in* MHP nos perfis *Enhanced Broadcast* e *Interactive Broadcast* (Perrot, 2001).

2.1.1.3. O uso do armazenamento de aplicações

A especificação do MHP prevê uma API opcional denominada *Application Storage*. Os receptores que implementam tal API são capazes de interpretar um tipo de sinalização de aplicação que permite o armazenamento de programas. Esse mecanismo provê uma série de vantagens:

- Permite a aplicação armazenada seja iniciada a qualquer momento pelo usuário ou pela difusora;
- Permite o controle de versão da aplicação;

- Permite a instalação automática de aplicações sem a necessidade de elaborar outras aplicações com tal finalidade (ou seja, aplicações que armazenam aplicações);
- Permite a desinstalação de aplicações.

Utilizando a API padrão de persistência da especificação MHP seria possível, também, armazenar aplicações. Contudo, tornaria necessário o envio prévio de outra aplicação para executar o carregamento das aplicações originais do disco.

Mais informações sobre o mecanismo de sinalização do MHP podem ser encontradas no Apêndice B.

2.1.2. O *Middleware* Americano

O *Advanced Television Systems Committee* (ATSC) é uma organização americana responsável por estabelecer padrões para TV. Esse comitê foi formado inicialmente pela *Electronic Industries Association* (EIA), *Institute of Electrical and Electronic Engeneers* (IEEE), *National Association of Broadcasters* (NAB), *National Cable Television Association* (NCTA) e *Society of Montion Picture and Television Engineers* (SMPTE).

Inicialmente, foi desenvolvido pelo ATSC um padrão denominado *DTV Application Software Environment Level 1* (DASE-1). O DASE adota modelos de aplicações baseados em linguagem procedural e declarativa. No ambiente declarativo é adotado o XHTML e no procedural a linguagem JAVA.

Em sua primeira versão o DASE era restrito apenas à interatividade local, pois o sistema não previa um canal de retorno. Contudo, surgiram ainda o DASE nível 2 e o DASE nível 3. No primeiro foram resolvidas questões relativas à segurança, como a criptografia dos dados; e, no segundo, já prevendo o canal de retorno, foi estabelecida uma integração da TV com a internet, dando acesso a programas de correio eletrônico, navegadores *web* entre outros, no terminal de acesso.

Paralelamente a essa iniciativa, desenvolveu-se o *OpenCable Applications Plataform* (OCAP) (OCAP, 2005). Esse padrão foi estabelecido pela CableLabs: um consórcio formado por membros da indústria de TV a cabo. Ele conta **apenas**

com o ambiente procedural para execução de aplicações denominadas OCAP-J e desenvolvidas para plataforma JAVA.

Dado o cenário mencionado, com dois tipos de *middleware* despontando no território americano e a clara tendência à harmonização proposta pelo GEM, o ATSC introduziu a especificação do *Advanced Common Application Plataform* (ACAP).

Middleware Americano ATSC/ACAP

O padrão ACAP surgiu a partir de um trabalho em conjunto com participantes do DVB em setembro de 2004. Sua especificação é primariamente baseada no GEM e no DASE e inclui funcionalidades adicionais do OCAP. Ela foi criada com o intuito de substituir o DASE e permitir uma compatibilidade com o GEM e com o OCAP. Dessa forma, além de permitir a criação de aplicações portáveis dentre os middlewares americanos e os compatíveis com o GEM, o padrão permite uma compatibilidade com as aplicações já existentes em território americano.

As aplicações ACAP são classificadas em duas categorias: conteúdos procedurais ou conteúdos declarativos. As aplicações contendo **apenas** conteúdo procedural são denominadas ACAP-J e as contendo conteúdo declarativo ACAP-X. Similar ao MHP, o ACAP classifica as plataformas que o implementam em dois perfis: o perfil ACAP-J, no qual a plataforma é obrigada a implementar apenas o suporte às aplicações procedurais; e o perfil ACAP-J e ACAP-X, que obriga a plataforma a dar suporte a programas procedurais, declarativos e híbridos. Esse enquadramento torna a existência de um ambiente declarativo facultativo e reforça a idéia de compatibilidade com o GEM.

No ambiente ACAP-X, as aplicações são representadas por documentos hipermídia compostos por meio de uma linguagem de marcação (e.g. XHTML), regras de estilo (CSS), scripts (ECMAScritpt), Xlets embutidos (aplicações ACAP-J), e elementos audiovisuais. Toda a definição desse ambiente é baseada em extensões/restrições da definição do DVB-HTML.

O ambiente ACAP-J é uma combinação do GEM, OCAP e DASE. Por isso, todo conteúdo obrigatório estabelecido no ambiente procedural desses padrões

são incluídos no ambiente ACAP-J. São previstas ainda, as seguintes extensões à API JAVA:

- API para Closed Captioning;
- API para localizadores de conteúdo específicos para o ACAP;
- API para o tratamento de eventos na interface com usuário;
- API para identificação do conteúdo;
- API estendendo a SI (*Service Information*)
- Integração DOM entre ambientes, que visa realizar a ponte do ambiente procedural com o declarativo quando este existir;

Contudo, apesar da interoperabilidade provida pelo padrão soar promissora, a especificação do padrão é relativamente recente. Isso contribui para haver ainda quase nenhuma penetração sua dentre os consumidores americanos.

Middleware Americano CableLabs/OCAP

A especificação OCAP 1.0 foi estabelecida em 2001. No entanto, apesar do estabelecimento da proposta de *middleware* ACAP pelo ATSC, a evolução do OCAP continuou a ocorrer paralelamente. Depois de algumas revisões, o padrão passou a implementar diretamente a API do GEM: o OCAP 1.0 I-16 (revisão 16), estabelecido em 2005, baseia-se no DVB-GEM 1.0.2 e DVB-MHP 1.0.3. Depois disso, acompanhando a evolução do GEM, em 2006 foi lançado o OCAP 1.1 correspondendo ao DVB-GEM 1.1 e DVB-MHP 1.1.2.

2.1.3. O *Middleware* Japonês

A especificação do sistema para transmissão digital terrestre japonês foi produzida em 1997 pela associação de indústrias e negócios de rádio japonesa (ARIB – *Association of Radio Industries and Business*), tendo contado com uma aprovação final pelo governo Japonês. Tal especificação estabeleceu o padrão conhecido como *Terrestrial Integrated Services Digital Broadcasting* (ISDB-T).

Assim como na maioria dos demais sistemas, a arquitetura ARIB é composta por dois subsistemas: um para a execução de programas procedurais; e outro para a apresentação de programas declarativos. Entretanto, no padrão japonês não foram definidos elementos capazes de estabelecer uma ponte entre esses dois subsistemas.

A linguagem utilizada no ambiente declarativo é denominada BML (*Broadcast Markup Language*). Essa linguagem encontra-se definida na especificação ARIB STD-B24 Volume 2 (*Data Coding and Transmission Specification for Digital Broadcasting Vol. II*). Ela baseia-se no XHTML e fornece suporte a CSS e ECMAScript.

No BML, o ECMAScript foi estendido de forma a fornecer uma API capaz de oferecer, principalmente, as seguintes funcionalidades:

- Controle do canal de retorno;
- Controle da apresentação;
- Alteração do conteúdo da página em tempo real;
- Processamento de eventos gerados pela emissora; e
- Processamento de eventos oriundos da interação com o usuário.

O ambiente de execução procedural foi definido na especificação ARIB STD-B23 (*Application Execution Engine Platform for Digital Broadcasting*) em 2003. Atualmente o padrão se encontra em sua revisão 1.1 estabelecida em 2004. O ambiente consiste de um sistema capaz de executar programas em *bytecode* JAVA e foi criado a partir de uma extensão do GEM 1.0 e do DVB MHP 1.0. Entretanto, ao contrário dos outros padrões, existe no mercado japonês uma maior penetração de aplicações declarativas e receptores que implementem suporte a tal paradigma. Isso se deve, possivelmente, ao fato da padronização tardia do ambiente procedural.

2.1.4. O *Middleware* Brasileiro

A proposta para o *middleware* utilizado pelo ISDTV-T, cuja arquitetura é mostrada na Figura 7, compreende a implementação de dois subsistemas:

- Um provê uma infra-estrutura para a execução de aplicações baseadas na linguagem procedural Java, sendo denominado Ginga-J (ilustrado na figura à direita da ponte); e
- O outro provê uma infra-estrutura para a apresentação de aplicações baseadas em documentos hipermídia escritos na linguagem declarativa NCL, sendo denominado Ginga-NCL (ilustrado na figura à esquerda da ponte).

No entanto, as aplicações não necessitam ser exclusivamente procedurais ou declarativas. Existem elementos em ambos os subsistemas que permitem a construção de aplicações híbridas: uma aplicação declarativa (NCL) pode possuir outra aplicação procedural embutida; ou, por outro lado, uma aplicação procedural pode referenciar o conteúdo declarativo, como, por exemplo, criando e iniciando apresentações.

O uso da linguagem NCL em ambientes de TV Digital interativa visa garantir o sincronismo quando da reprodução de programas interativos multimídia, hipermídia e não-lineares. A entidade responsável pela apresentação de documentos NCL é denominada Formatador NCL.

A partir da especificação do documento NCL recebida, o Formatador constrói um plano de apresentação que contém as características de apresentação de cada objeto de mídia, a programação das tarefas a serem escalonadas e as informações dos relacionamentos de sincronização entre os objetos de mídia. Baseado nos eventos gerados pelos exibidores de mídia e nos eventos gerados pelo usuário, o escalonador de apresentação controla a execução sincronizada de todo o documento, realizando ajustes quando esses se fazem necessários (Rodrigues, 2003).

Na Figura 7 são mostrados, ainda, dois exibidores que devem ser implementados pelo *middleware*: o exibidor XHTML, que deve possuir um interpretador ECMAScript e suporte a CSS; e o exibidor Lua (Ierusalimschy, 2003), que representa uma máquina virtual dessa linguagem.

Dependendo da implementação fornecida pelo módulo XHTML o *middleware* pode se tornar compatível com outros padrões declarativos (BML, ACAP-X, DVB-HTML). Mais ainda, ao padronizar a existência de tal módulo, torna-se o *middleware* compatível com o padrão ITU-T J.201.

Para que outros tipos de conteúdo, além dos citados anteriormente, possam ser apresentados no ambiente declarativo, é necessária a criação de adaptadores que respeitem a API de Adaptadores do Formatador. Essa API permite que os exibidores desses outros tipos de conteúdo sejam controlados de acordo com o modelo de apresentação do Formatador NCL.

Outra estrutura importante na arquitetura é o Gerenciador de Bases Privadas. Esse gerenciador é responsável por lidar com um conjunto de bases responsáveis por armazenar documentos NCL. Tais estruturas são denominadas privadas devido ao fato de cada base ser de uso exclusivo de um canal de TV específico.

As atividades do Gerenciador de Bases Privadas incluem, além de assegurar o acesso exclusivo a essas bases, o processamento de Comandos de Edição NCL (Soares et al, 2006). Esses comandos permitem, por exemplo, a alteração de um documento NCL durante sua apresentação.

Figura 7 – Arquitetura do *middleware* Ginga.

O subsistema procedural (Ginga-J) constitui uma extensão à especificação do GEM, mantendo, dessa forma, a compatibilidade com todos os demais padrões que implementam tal especificação (DVB MHP, ATSC ACAP e OCAP, ISDB ARIB). A API proposta pelo Ginga-J pode ser dividida em três porções: a porção definida pelo GEM; a porção definida pelo Ginga-J mas adaptável ao GEM; e a porção definida apenas pelo Ginga-J.

Dentre as inovações introduzidas pelo subsistema procedural do *middleware* brasileiro, pode-se destacar o suporte a interações multiusuário, multidispositivo e multi-rede de comunicação, com o receptor (Filho et al., 2007).

Abaixo desses dois subsistemas lógicos (Ginga-NCL e Ginga-J) existe uma camada, que lhes oferece suporte, denominada Núcleo Comum. Essa camada é composta por decodificadores comuns de conteúdo (como PNG, JPG, GIF ou MPEG) e rotinas para obter conteúdos transportados no Fluxo de Transporte MPEG-2 (ISO, 1993) e via canal de retorno. Possui ainda uma Máquina Virtual Java (*JVM – Java Virtual Machine*) para a execução de códigos binários dessa linguagem.

2.2. Definição do padrão GEM

O GEM *(Globally Executable MHP*) foi proposto, inicialmente, para que as aplicações MHP pudessem ser utilizadas sobre as plataformas do *middleware* dos EUA (*CableLabs*) e do Japão (ARIB). O GEM é considerado um acordo de harmonização. Isso porque, além de capturar as interfaces e toda a semântica definidas pelo MHP (independentes da plataforma DVB), o GEM inclui as necessidades impostas por outros padrões internacionais. A *CableLabs* participou da composição da primeira versão do GEM, para tornar compatível seu *middleware* para a TV a cabo americana, o OCAP. E, mais recentemente, o *middleware* japonês ARIB e o candidato a padrão americano ACAP também tiveram suas necessidades de harmonização com o GEM concluídas e podem, dessa forma, ser classificados como padrões compatíveis com o GEM.

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 um terminal de acesso pode ser implementado, 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.

Por definir um *framework* baseado no padrão MHP, o documento de especificação do GEM é na realidade uma listagem de referências a outros documentos do consórcio DVB. Ele descreve as diversas partes do MHP que são independentes do padrão DVB e salienta aquelas que devem ser substituídas de acordo com a infra-estrutura de implementação das novas especificações compatíveis. Esses pontos de flexibilização do GEM devem ser, então, preenchidos por "equivalentes funcionais" – mecanismos que lançam mão de outras tecnologias (definidas pelas respectivas organizações de TV digital) para implementar uma funcionalidade análoga àquela proposta pelo DVB. Tais tecnologias passam a ser qualificadas como equivalentes funcionais somente após negociações entre o consórcio DVB e cada uma das organizações que requisitam a conformidade com o GEM.

No GEM, diferente do DVB, são definidos apenas os dois primeiros perfis: o *Enhanced Broadcast* e o *Interactive Broadcast.*

2.2.1. Políticas de segurança para execução de aplicativos no GEM

Todo *Xlet* a ser executado em ambientes que implementem o GEM está sujeito a uma série de restrições de segurança. Nem todos os recursos do receptor, disponíveis no ambiente Java, têm seu uso diretamente permitido a uma aplicação. De acordo com o padrão, as aplicações que são enviadas para um receptor serão classificadas como não confiáveis e não possuirão acesso a toda a API Java.

Para garantir um nível maior de confiabilidade à aplicação, é necessária sua assinatura digital. Uma aplicação assinada pode ter suas permissões de acesso alteradas através de um Arquivo de Requisição de Permissão (*Permission Request File* – PRF).

Os PRFs são arquivos XML que devem estar presentes no mesmo diretório do arquivo inicial de uma aplicação que necessite de alterações em seu nível de acesso padrão. Nesses arquivos, o identificador formal da DTD deve corresponder à organização que especificou o documento. Além disso, o prefixo do PRF deve identificar a especificação de terminal GEM (Ex.:

ocap.<nome_do_aplicativo>.perm). Um exemplo de tal arquivo pode ser encontrado no Apêndice A.

Os objetivos das alterações realizadas pelos PRFs são as permissões JAVA. As permissões Java representam o acesso a um recurso do sistema (SUN, 2002). A permissão tem um nome (normalmente o nome do alvo) e, comumente, uma lista de um ou mais parâmetros (ex.: java.io.FilePermission "/tmp/abc/", "read" – permite a leitura dos arquivos no diretório "/tmp/abc/"). As permissões mais relevantes disponíveis para utilização em um PRF e necessárias para a implementação de um ambiente declarativo virtual são:

- ***org.dvb.net.tuning.TunerPermission***: permissão para realizar a troca entre os fluxos;
- ***org.dvb.user.UserPreferencePermission***: permissão para leitura/escrita das preferências do usuário. Sendo a leitura permitida também para aplicativos não-assinados;
- ***java.net.SocketPermission***: permissão para se comunicar com *hosts* remotos (como por exemplo abrir uma página HTML externa); e
- ***java.io.FilePermission***: permissão de leitura/escrita dos arquivos. Sendo a leitura permitida também para aplicativos não-assinados;
- ***org.dvb.application.AppsControlPermission***: permissão para controlar o ciclo de vida da aplicação;
- ***org.dvb.net.rc.RCPermission***: permissão para comunicação utilizando o canal de retorno (*connect*, *listen*, *resolve*).

2.2.2. Sinalização de Aplicações no GEM

Na especificação do GEM são fornecidos requisitos mínimos de sinalização que devem ser implementados pelos padrões baseados nessa especificação:

- para qualquer aplicação: seu descritor deve possibilitar a identificação do nome da aplicação; o identificador único da

aplicação e da organização que a produziu; e a localização da aplicação e demais arquivos por ela utilizados; e

- para aplicações procedurais (e.g. Java): deve ser fornecida informação suficiente para sinalizar os parâmetros da aplicação e indicar sua classe inicial.

Uma forma de implementar tais mecanismos é utilizando a Tabela de Informação de Aplicações (*Application Information Table* – AIT). Toda a informação sobre as aplicações que podem ser obtidas via carrossel de objetos ou via canal de retorno são armazenadas nessa tabela. Ela é enviada em conjunto com outros fluxos elementares em um fluxo de transporte MPEG-2.

Na Figura 8 é mostrado o processo de identificação da AIT em um fluxo de transporte MPEG-2. A Tabela de Associação de Programas (*Program Association Table* – PAT) pode ser encontrada nos pacotes de fluxo de transporte identificados pelo PID 0. Supondo que a AIT da aplicação está associada com o programa 2, a Tabela de Mapeamento de Programa (*Program Map Table* – PMT) pode ser identificada nos pacotes com o PID 23. Na PMT, se o tipo de fluxo possuir o valor 0x05, que representa o fluxo do tipo AIT, então os pacotes com o PID 0x200 estarão carregando a AIT correspondente a esse programa.

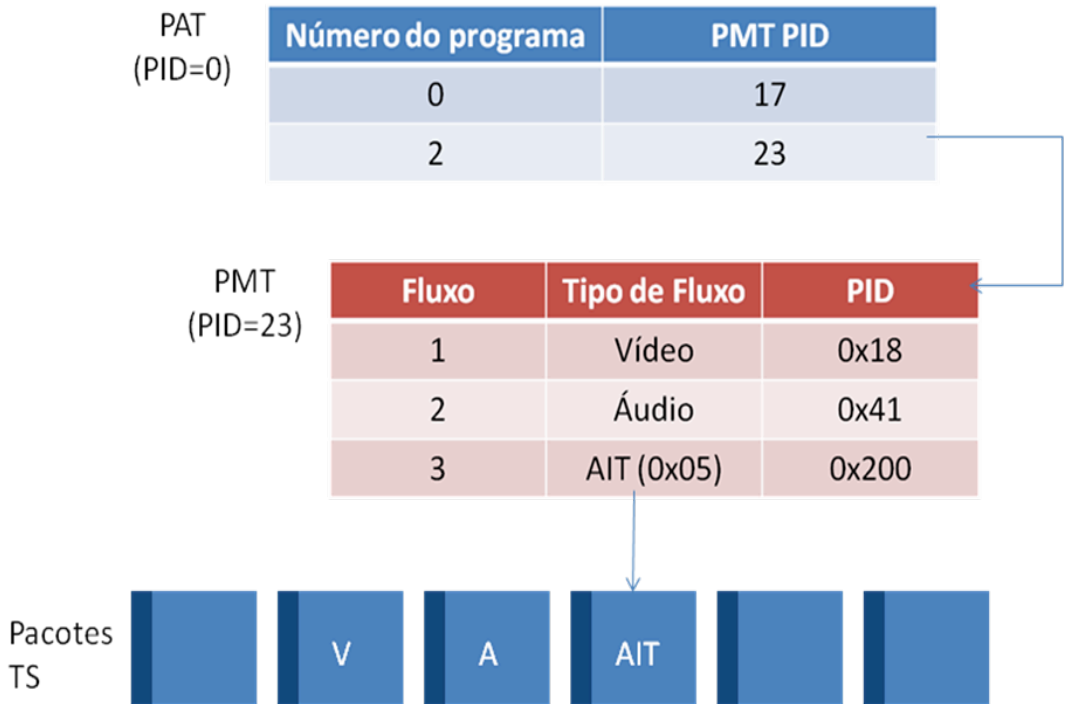

Figura 8 – O processo de identificação da AIT.

No Apêndice B é ilustrada como é formada a AIT em um sistema de TV Digital que utilize o DVB-MHP.

2.3. O Blue-Ray Disc

O disco *Blue-Ray* (BD – *Blue-Ray Disc*) é uma revolução em termos de armazenamento em mídia ótica para consumidores de produtos eletrônicos, PCs e consoles de vídeo game (Playstation 3). Ele proporciona: uma resolução de alta definição de 1920 x1080, comparada aos 720x480 proporcionados pelo DVD e os 352x240 pela TV analógica; e uma capacidade de armazenamento de 25GB e 50 GB, atualmente, e até 200GB, com o lançamento futuro de discos multicamadas.

Com relação à interatividade, para uma plataforma BD encontram-se definidos dois modos de operação que podem, inclusive, coexistir:

- High Definition Movie (HDMV) – esse modo enfatiza uma compatibilidade com os processos de produção dos DVDs atuais. O modo HDMV suporta todas as funcionalidades proporcionadas pelos DVDs atuais e seus formatos. Contudo, proporciona melhoras na qualidade do vídeo, áudio, definição, experiência do usuário, entre outros; e
- BD-J – um ambiente de aplicações programáveis com possibilidades de canal de retorno, possibilitando aos produtores de conteúdo uma alta interatividade e títulos BD-ROM atualizáveis.

Foi estabelecida pela BDA (*Blue-ray Disc Association*) a linguagem JAVA como tecnologia a ser adotada em aplicações BD-J. Isto possibilita a criação de aplicações com interatividade avançada proporcionando-se as seguintes vantagens ao BD em relação ao DVD: liberdade para o desenho da interface com o usuário; controle da execução do áudio e vídeo; atualização dinâmica do conteúdo (trailers, legendas, materiais de bônus) via um canal de retorno; outras formas de conteúdo (jogos interativos no disco e online, eventos ao vivo, compras online).

Ao adotar como linguagem o JAVA, a BDA, buscando seguir a tendência global de harmonização dos ambientes de execução procedural para sistemas de TV digital, estabeleceu que as plataformas BD deveriam implementar o *framework* GEM.

Seguindo a especificação do GEM, o padrão BD-J também define dois perfis para suas aplicações:

- BD-VIDEO – esse perfil não requer canal de retorno; e
- BD-LIVE – esse perfil requer canal de retorno proporcionando o download de conteúdos e aplicações novas.

2.4. Ambientes de Execução JAVA para TV

Um ambiente de execução Java (*Java Runtime Environment* – JRE) é uma implementação da tecnologia Java para uma plataforma específica. Ele é instalado e executado como uma aplicação nativa com o propósito de executar e gerenciar aplicativos JAVA. O JRE fornece, portanto, uma abstração comum para as aplicações JAVA e é implementado em código nativo.

A tecnologia Java pode ser subdividida como mostra a Figura 9. Esta divisão procura enquadrar essa tecnologia em algumas plataformas-alvo:

- A edição Java EE (antiga J2EE) é voltada para o segmento dos servidores, é uma edição que possui recursos avançados para aplicações empresariais;
- A edição Java SE (antiga J2SE) é a edição padrão recomendada para a maioria das plataformas, incluindo, principalmente, desktops; e
- A edição Java ME (antiga J2ME) é a edição para plataformas com recursos limitados como PDAs, celulares, set-top boxes, quiosques, terminais de impressão, entre outros.

A Java ME é, dentre todas as edições citadas, a que precisa se adequar à maior variedade de dispositivos. Além disso, para ganhar relevância ela deve ser amplamente adaptável, pois a categoria de produtos por ela compreendida está em constante evolução: os fornecedores estão constantemente adicionando novas funcionalidades e identificando novos nichos de produtos (SUN, 2005a). Para isso, a edição Java ME prevê algumas alternativas de configurações, perfis e pacotes opcionais para um ambiente de execução Java específico para um produto.

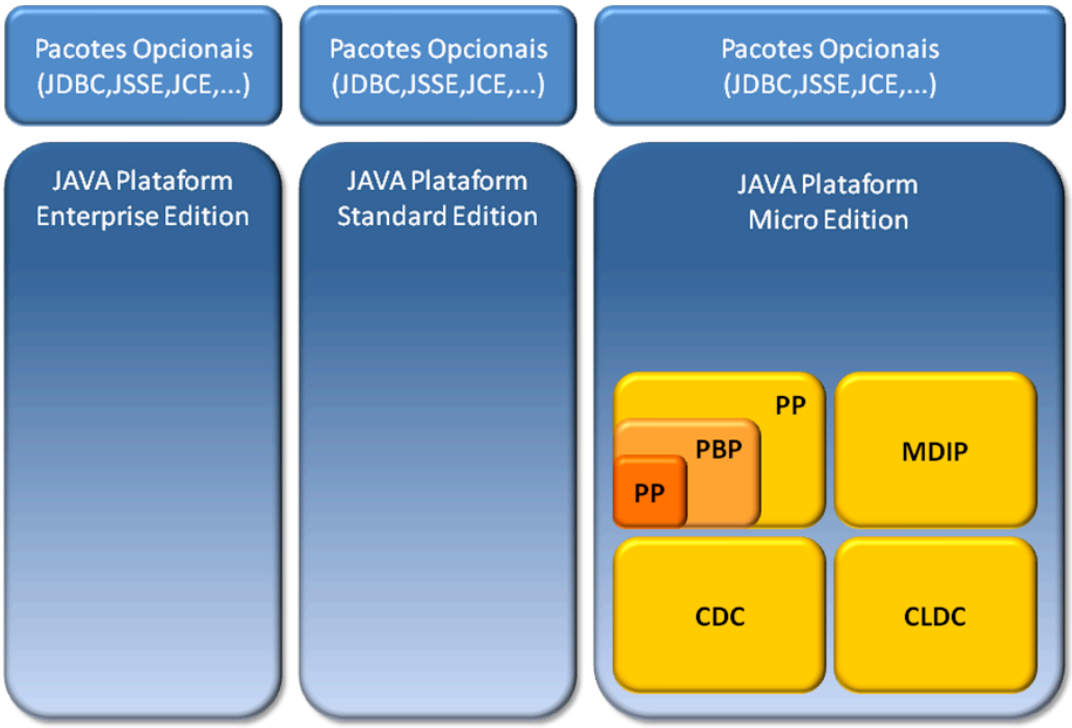

Figura 9 – Arquiteturas de plataformas JAVA.

O CLDC (*Connected Limited Device Configuration*) é uma configuração voltada para telefones celulares e PDAs de pequeno porte. Essa configuração define um pequeno subconjunto da API fornecida pelo Java SE e compartilhada com o CDC. O objetivo principal do CLDC é a economia da memória. Para isso o ambiente de execução não oferece funcionalidades como API de reflexão e o carregamento de classes pela aplicação. Com essa economia os dispositivos que implementam tal configuração possuem uma necessidade de memória de apenas 128 KB a 256 KB.

O CDC (*Connected Device Configuration*) tem como objetivo principal a compatibilidade com o Java SE e suportar dispositivos com recursos limitados. Ele suporta a implementação completa da máquina virtual Java incluindo carregamento de classes e suas classes essenciais. Contudo, para atender ao requisito de operar sobre plataforma de recursos limitados, o CDC realiza modificações no Java SE: algumas de suas classes tiveram suas interfaces modificadas e outras retiradas inteiramente. Dessa forma, o requisito dessa configuração fica na casa dos 2 MB de RAM e 2MB de ROM.

Existem definidos para o CDC três perfis. A definição desses perfis proporciona aos fabricantes uma maior flexibilidade, trazendo diferentes níveis de

sofisticação à implementação e possibilitando o suporte a diferentes tipos de dispositivos com um ambiente de execução Java compatível. São eles:

- Foundation Profile (FP) – é o perfil mais básico. Possui apenas bibliotecas básicas como rede e Entrada/Saída (I/O). Esse perfil não suporta a criação de gráficos nem interfaces gráficas com o usuário (GUIs);
- Personal Basis Profile (PBP) – É o perfil intermediário e incorpora o FP. Permite a construção de GUIs através de um subconjunto limitado do AWT (Geary & McClellan, 1997) e também implementa o modelo de programação Xlet que será visto a seguir; e
- Personal Profile (PP) – É o perfil CDC mais avançado e engloba o PBP além de suportar integralmente o AWT e *applets* (SUN, 1994).

Para o desenvolvimento de um ambiente de execução Java em uma edição Java ME, é obrigatória a escolha de uma configuração e um perfil. No entanto, podem ser adicionados pacotes extras de acordo com o critério do fabricante, como mostrado na Figura 10. A figura mostra a criação de um ambiente onde foi escolhida a configuração CDC, o perfil PBP, mais o pacote opcional JSSE (*Java Secure Sockets Extension*), para realizar conexões de rede seguras.

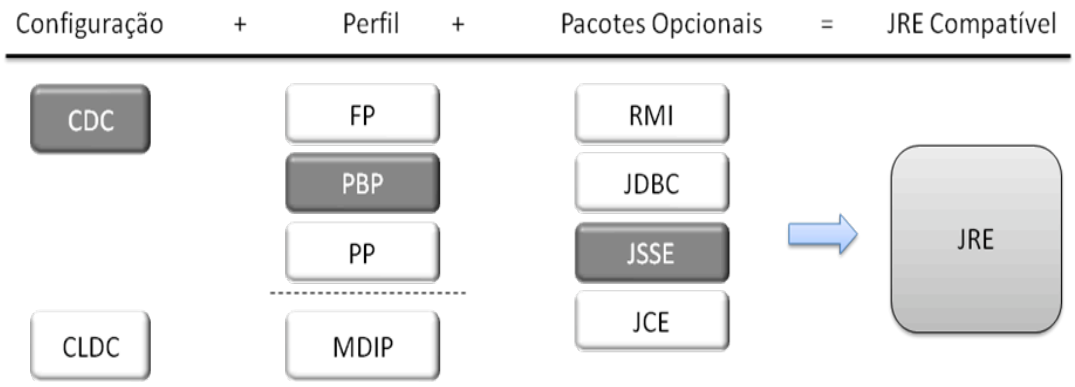

Figura 10 – Exemplo de ambiente de execução Java.

A construção do ambiente de execução Java mínimo para TV digital é muito semelhante ao exemplo citado anteriormente. Apresenta a configuração CDC, o perfil PBP (sendo que o PP é permitido) e algumas bibliotecas opcionais. A soma do PBP com o CDC mantém, ainda, a compatibilidade com o PersonalJava (uma configuração de máquina virtual Java prevista no GEM/MHP e cuja normalização foi descontinuada sendo substituída pelo CDC/CLDC).

No GEM, por se tratar de um ambiente de TV e por ser um acordo de harmonização, algumas bibliotecas tornaram-se obrigatórias. Dentre elas, a biblioteca Java TV introduz o modelo de programação *Xlet*.

- download de conteúdos e aplicações novas.

2.4.1. O Modelo de programação *XLET*

O modelo de programação *Xlet* é voltado para aplicações desenvolvidas para sistemas de TV Digital. Nesse modelo é possível realizar um controle do ciclo de vida dessas aplicações através de uma interface Java, mostrada na Figura 12, por elas implementada. As diversas entidades envolvidas nesse modelo, e o protocolo de comunicação entre elas, são mostrados na Figura 11.

Os *Xlets*, como são conhecidas as aplicações, podem ser lançados automaticamente, via sinalização, ou iniciados através da navegação pelo controle remoto dos telespectadores. Eles podem encontrar-se residentes no receptor, ser extraídos do carrossel de objetos e dados de um fluxo DSM-CC (ISO, 1998), ou ser obtidos pelo canal de retorno.

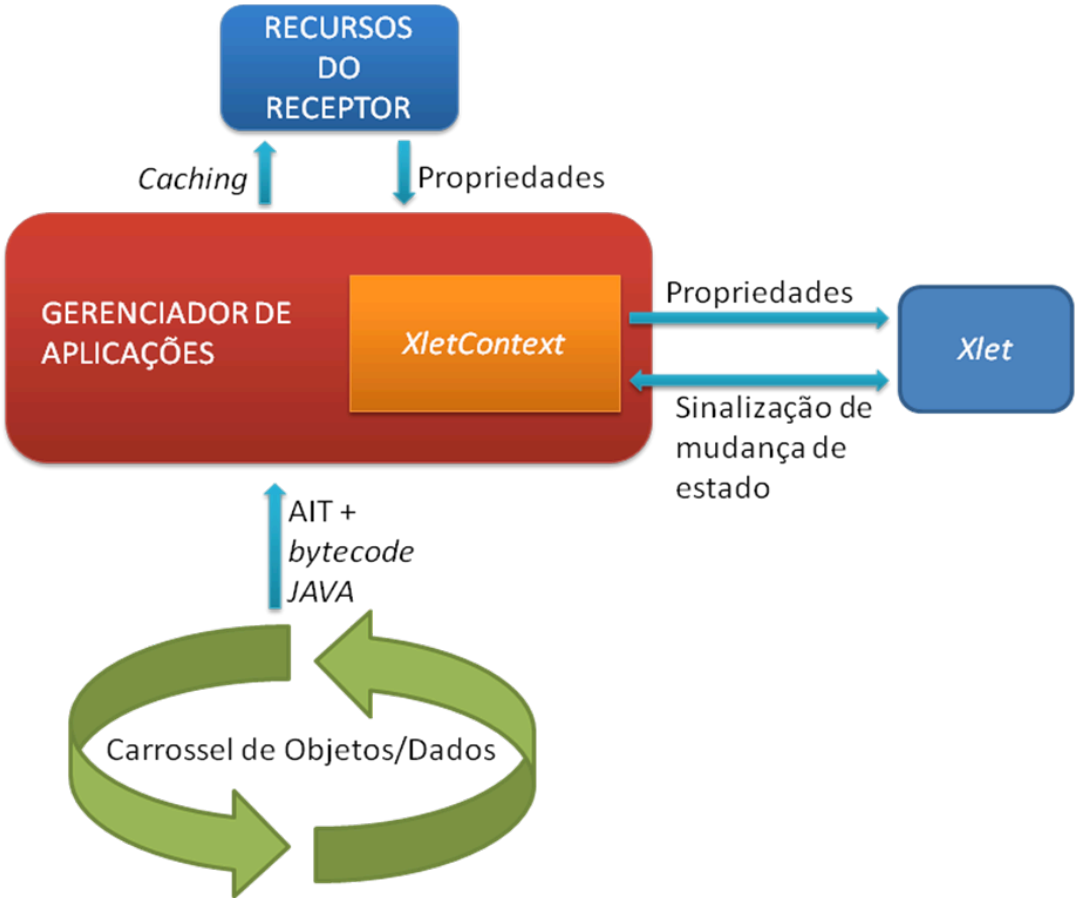

Figura 11 – Protocolo entre as entidades presentes no modelo e programação *Xlet*.

O Gerenciador de Aplicações (*Application Manager*) é a entidade responsável pelo carregamento e o controle direto do ciclo de vida dos *Xlets*. Ele dever ser capaz de interpretar os dados provenientes da AIT (ou seja, localização da aplicação, classe inicial etc.) para, de acordo com aqueles recebidos, gerenciar a execução das aplicações. Através de um mecanismo conhecido como *caching*, o Gerenciador de Aplicações mantém uma tabela onde monitora as mudanças de estado e recomeça a execução de *Xlets* presentes no receptor. O Gerenciador de Aplicações é parte do sistema e reside no receptor.

No modelo de programação *Xlet*, as aplicações devem implementar a interface mostrada na Figura 12. A classe Java que implementar essa interface é a porta de entrada no aplicativo. Ela possuirá métodos que refletem diretamente os estados do ciclo de vida de um *Xlet*.


```
public interface Xlet {
      public void initXlet(XletContext xtx)
            throws XletStateChangeException;
      public void startXlet()
            throws XletStateChangeException;
      public void pauseXlet();
      public void destroyXlet(boolean unconditional)
            throws XletStateChangeException;
}
```

Figura 12 - Interface do *Xlet*.

Uma aplicação *Xlet* poderá encontrar-se nos estados ilustrados pela máquina de estados da Figura 13. Quando a classe Java inicial de uma aplicação é carregada, do carrossel de objetos ou do receptor, e instanciada, ela entra no estado *Loaded* (carregada). O estado *Loaded* significa que a aplicação já foi carregada, mas ainda não foi iniciada. No momento seguinte o Gerenciador de Aplicações sinaliza o *Xlet* para que ele seja iniciado (chamando seu método "*initXlet"*). Após iniciado ele entra no estado *Paused* (pausado). Uma aplicação no estado *Paused* está minimizando o uso de recursos para maximizar sua sobrevivência, e está pronta para executar. No estado *Active*, a aplicação está funcionando plenamente, e no estado *Destroyed* já liberou todos os recursos e terminou sua execução.

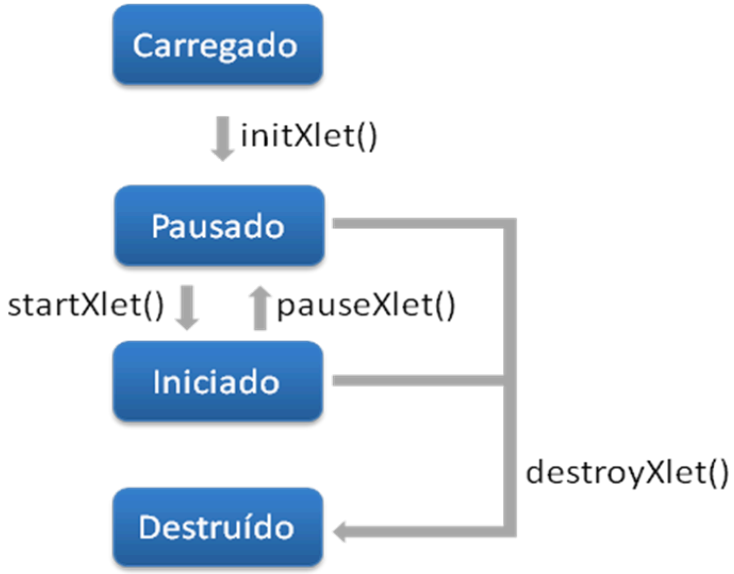

Figura 13 - Máquina de estados do ciclo de vida de um *Xlet*.

Além dos *Xlets* e do Gerenciador de Aplicações, existe o Contexto ou *XletContext*. Todo *Xlet* possui um contexto associado, ou seja, uma instância da classe `javax.tv.xlet.XletContext`. Isso é similar à classe *AppletContext* que é associada a um *applet*.

O *XletContext* é uma interface Java (mostrada na Figura 14) cuja implementação é fornecida pelo Gerenciador de Aplicações. Através dessa interface, o Gerenciador de Aplicações pode controlar o estado de um *Xlet*, tendo a capacidade de encerrar sua execução a qualquer momento. O *Xlet* pode, ainda, utilizar o *XletContext* para acessar propriedades do receptor ou realizar uma mudança em seu estado.

```
public interface XletContext {
     public static final String ARGS = javax.tv.xlet.args";
     public void notifyPaused();
     public void notifyDestroyed();
     public resumeRequest();
}
```

Figura 14 – Interface do *XletContext*.

Na requisição de mudança de estado iniciada pelo *Xlet*, ele notifica seu novo estado desejado ao Contexto. A partir daí, o Gerenciador de Aplicações é notificado e, em seguida, realiza a mudança do estado do *Xlet*. Utilizando esse mecanismo de *callback*, o Gerenciador de Aplicações pode manter atualizado o status dos *Xlets* por ele controlados. Portanto, o Contexto é uma ponte de comunicação entre o *Xlet* e o Gerenciador de Aplicações.

Os métodos "*notifyDestroyed*" e "*notifyPaused*" permitem ao *Xlet* notificar o terminal sobre a possibilidade de terminá-lo ou pausá-lo. O *Xlet* pode usar esses métodos para ter certeza que o terminal sabe o estado de toda aplicação e pode tomar a ação apropriada. Esses métodos devem ser chamados imediatamente antes do *Xlet* entrar nos estados de *Paused* ou *Destroyed*, isto porque o terminal pode tomar uma ação que a aplicação não esteja preparada.

Uma aplicação pode requisitar a mudança do estado *Paused* para o *Started* usando o método "*resumeRequest*". Isso acontece quando um determinado evento occorreu, como, por exemplo, um tempo certo foi atingido, um evento certo foi detectado num fluxo MPEG, etc. Com esse método, é possível reiniciar uma aplicação após esta ter sido suspensa.