PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rodrigo Laiola Guimarães

Romualdo M. de Resende Costa

Interatividade e Sincronismo em TV Digital

MONOGRAFIA DA DISCIPLINA DE TÓPICOS DE HIPERTEXTO E MULTIMÍDIA II - "SEMINÁRIOS SOBRE TV DIGITAL INTERATIVA"

DEPARTAMENTO DE INFORMÁTICA

Programa de Pós-Graduação em Informática

Rio de Janeiro

Fevereiro de 2006

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rodrigo Laiola Guimarães

Romualdo M. de Resende Costa

Interatividade e Sincronismo em TV Digital

Monografia da Disciplina de Tópicos de Hipertexto e Multimídia II - "Seminários sobre TV Digital Interativa"

Monografia apresentada como requisito parcial para aprovação na disciplina de Tópicos de Hipertexto e Multimídia II - "Seminários sobre TV Digital Interativa" do Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Luiz Fernando Gomes Soares

Rio de Janeiro, fevereiro de 2006

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rodrigo Laiola Guimarães

Romualdo M. de Resende Costa

Interatividade e Sincronismo em TV Digital

Monografia apresentada como requisito parcial para aprovação na disciplina de Tópicos de Hipertexto e Multimídia II - "Seminários sobre TV Digital Interativa" do Programa de Pós-Graduação em Informática da PUC-Rio.

Luiz Fernando Gomes Soares
Orientador
Departamento de Informática - PUC-Rio

Rio de Janeiro, fevereiro de 2006

Rodrigo Laiola Guimarães

Graduado em Engenharia de Computação pela Universidade Federal do Espírito Santo (UFES) em 2004. Atualmente, integra o grupo de pesquisadores do Laboratório TeleMídia da PUC-Rio, desenvolvendo pesquisa na área de Redes de Computadores e Sistemas Hipermídia.

Romualdo M. de Resende Costa

Mestre em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) em 2005. Atualmente, integra o grupo de pesquisadores do Laboratório TeleMídia da PUC-Rio, desenvolvendo pesquisa na área de Redes de Computadores e Sistemas Hipermídia.

Ficha Catalográfica

Guimarães, Rodrigo Laiola; Costa, Romualdo M. de Resende

Descrição da Ficha Catalográfica

Informação Técnica da Ficha Catalográfica

Natureza da Ficha Catalográfica

Inclui referências bibliográficas.

Interatividade; TV Digital; Sincronismo; Ferramentas de autoria

Dedicamos este trabalho a todos àqueles que acreditam que a ousadia e o erro são caminhos para as grandes realizações.

Agradecimentos

Nossa sincera gratidão e admiração pelo nosso orientador Luiz Fernando Gomes Soares por sua tamanha dedicação e incessante esforço em me ajudar durante o desenvolvimento deste trabalho de pesquisa científica.

Aos nossos colegas do TeleMídia, pelo companheirismo e ajuda prestados. Em especial a Rogério e a Rogerinho (Jr.) pela atenção, paciência e boa vontade em ensinar, tendo sido guias para o desenvolvimento deste trabalho, sua ajuda foi fundamental.

À CAPES, ao CNPq e ao TeleMídia pelo apoio financeiro.

Resumo

Guimarães, Rodrigo Laiola; Costa, Romualdo Monteiro de Resende **Interatividade e Sincronismo em TV Digital.** Rio de Janeiro, 2005. 66p. Monografia da Disciplina de Tópicos de Hipertexto e Multimídia II - "Seminários sobre TV Digital Interativa" - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os atuais padrões para TV digital definem um conjunto de linguagens para a especificação de aplicações. Através dessas linguagens são definidas as relações entre os eventos de uma aplicação, onde incluem-se os eventos de interatividade com o usuário e de sincronismo entre os objetos que compõem a apresentação. No contexto geral, pode-se afirmar que as linguagens atuais, baseadas em Java e baseadas em HTML+Scripts, definem as aplicações como programas, exigindo que o autor especifique uma sucessão de comandos para obter o resultado desejado. Para permitir a especificação das aplicações em um nível mais alto de abstração, linguagens declarativas, como SMIL e NCL podem ser utilizadas. Nessas linguagens o autor deve especificar o conjunto de tarefas a serem realizadas, sem que seja necessário mencionar a implementação dessas tarefas. Além do uso de linguagens voltadas para o domínio da aplicação, este trabalho aborda o uso de ferramentas destinadas ao desenvolvimento de aplicações para TV digital, com o objetivo de utilizar a autoria gráfica em substituição a autoria textual.

Palavras-chave

Interatividade; TV Digital; Sincronismo; Ferramentas de autoria

Sumário

Lista de tabelas

Lista de figuras

1 Introdução

Um sistema hipermídia normalmente é composto por três ambientes: o ambiente de autoria, no qual os autores criam os seus hiperdocumentos; o ambiente de armazenamento, onde o hiperdocumento criado na fase de autoria é armazenado e mantido em servidores; e o ambiente de execução, onde o documento recém-editado (ou já armazenado) é apresentado ao usuário [Coelho04]. A Figura 1 ilustra a modelagem desses três ambientes de um sistema hipermídia.

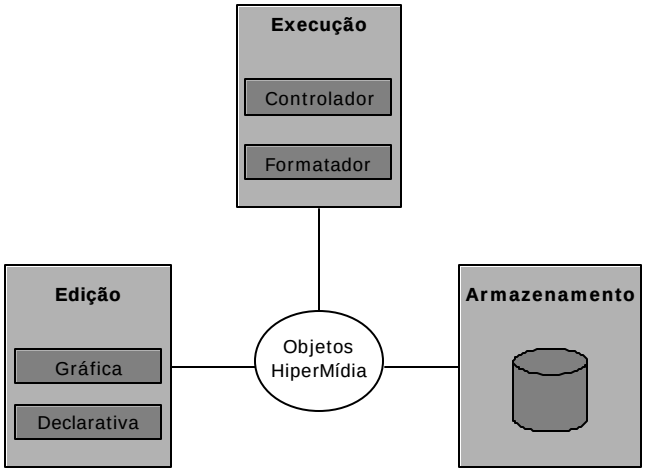

Figura 1 - Subsistemas de um sistema hipermídia [Coelho04].

A autoria em sistemas hipermídia costuma ser feita através do uso de interfaces gráficas, com editores especializados que oferecem facilidades para criação e edição de hiperdocumentos, ou então pela forma textual, na qual o autor utiliza um editor de texto e uma linguagem para especificar seus documentos.

No ambiente da TV digital, a capacidade de difusão de informação em várias mídias diferentes, tais como áudio e vídeo principal e dados (como texto, imagens e outros vídeos e áudios), faz com que esse ambiente possa oferecer diversos tipos de serviços, desde a simples distribuição de conteúdo áudio-visual da programação atual da TV analógica por difusão, até serviços hipermídia antes não disponíveis.

Através de diferentes mecanismos de sincronização e interatividade, é

possível que programas de TV sejam formados por aplicações hipermídia complexas, onde a interação do usuário com o conteúdo disponibilizado pelas emissoras permitirá o acesso a serviços e a personalização do conteúdo.

A seguir, a motivação para o desenvolvimento deste trabalho é apresentada.

1.1. Motivação

É preciso ter em mente que a sincronização em um sistema de TV digital está presente até mesmo em aplicações simples, como a exibição temporalmente sincronizada do fluxo de vídeo e áudio principal de um programa. Aplicações para TV digital são casos particulares de aplicações multimídia/hipermídia e, muitas vezes, devem lidar com a sincronização, espacial e temporal, de objetos de diferentes tipos de mídia, além dos objetos de vídeo e áudio que compõem o fluxo principal. Os vários objetos sincronizados irão compor um documento multimídia/hipermídia.

Muitas vezes, o sincronismo existente em documentos multimídia pode não ser apenas baseado em eventos onde é possível prever o tempo e local de suas ocorrências relativos a uma referência qualquer. A interatividade nada mais é que um exemplo de evento imprevisível, onde a interação do usuário dará início a um outro evento. Documentos multimídia onde é possível estabelecer o sincronismo entre a interação do usuário e outros objetos que compõem o documento são usualmente denominados documentos hipermídia.

Nesse mesmo contexto, o uso de ferramentas de autoria para a criação de programas multimídia/hipermídia pode agilizar e facilitar o processo de criação, uma vez que essas ferramentas procuram abstrair do autor toda a complexidade de se programar em uma linguagem qualquer, como as linguagens de programação tradicionais ou mesmo as linguagens baseadas em modelos hipermídia.

Dada essas motivações, cabe a análise criteriosa dos mecanismos de sincronização e interatividade nos principais padrões de TV digital existentes hoje, destacando suas características mais importantes. Além disso, é importante a análise de ambientes de autoria que explorem tornar mais ágil e prático o desenvolvimento de aplicações para o ambiente de TV digital interativa.

1.2. Objetivos

Esta monografia tem como objetivo analisar os mecanismos de sincronismo e interatividade presentes nos principais padrões de TV digital existentes. Em uma segunda frente, não menos importante, é feita uma análise criteriosa das principais características de algumas ferramentas de autoria disponíveis para sistemas multimídia/hipermídia.

O estudo do suporte ao sincronismo e à interatividade definidos nos padrões ATSC, DVB e ISDB são discutidos. Para isso, são citadas as suas arquiteturas e principais características.

Em relação à necessidade de conhecer o modelo ou a linguagem para a criação de programas multimídia/hipermídia, em cada uma das ferramentas de autoria analisadas, são levantadas suas principais características funcionais, voltadas tanto para usuários leigos quanto para usuários avançados.

1.3. Estrutura da Monografia

Esta monografia encontra-se organizada como a seguir. O Capítulo 2 insere o leitor no ambiente através da apresentação resumida dos conceitos básicos relativos aos padrões de TV digital ATSC, DVB e ISDB.

O Capítulo 3 faz uma discussão dos mecanismos de sincronização e interatividade no mundo de TV digital, com destaque para as linguagens utilizadas na especificação desses eventos. Uma parte específica trata de linguagens declarativas e da possibilidade de uso dessas linguagens pelos padrões.

No Capítulo 4 é feita uma análise de algumas ferramentas de autoria voltadas para a produção de conteúdo multimídia/hipermídia para TV digital interativa, destacando-se os pontos positivos e negativos de cada uma delas.

Por fim, o Capítulo 5 tece as conclusões e descreve trabalhos futuros.

2 Padrões para TV Digital

A padronização da codificação e do transporte de dados estabelece regras para a introdução de informação no fluxo de transporte das emissoras de TV. Essas regras abrangem também a possibilidade de relacionamento entre os dados e o conteúdo audiovisual da TV.

No terminal de acesso para o sistema de TV digital é desejável que exista um elemento capaz de fornecer uma abstração do sistema para as aplicações e os usuários, escondendo toda a complexidade dos mecanismos definidos pelos padrões, protocolos de comunicação e até mesmo do sistema operacional instalado no equipamento.

Para que aplicativos desenvolvidos por diferentes emissoras possam ser executados em diferentes terminais de acesso, é necessário ainda a definição de uma plataforma única de execução/apresentação. A essa abstração dá-se o nome de *middleware*. A padronização desse elemento permite a construção de aplicações independentes do hardware e do sistema operacional, e executáveis em qualquer plataforma de qualquer fabricante. Além do *middleware*, também há a necessidade da padronização de outros componentes, que vão desde a linguagem utilizada nos programas desenvolvidos, até as classes de serviços oferecidas na execução dos programas em um terminal de acesso.

Neste capítulo são apresentadas as características dos padrões ATSC, DVB e ISDB que são pertinentes a este trabalho. Ao final deste capítulo é feita uma breve discussão sobre os esforços em torno do GEM.

2.1. ATSC

O ATSC (*North-America/US Advanced Television Systems Committee*) é a organização responsável por especificar o padrão de TV digital dos Estados Unidos. Esse padrão recebeu o mesmo nome do seu comitê responsável O

objetivo inicial do ATSC era definir o serviço de TV digital voltado principalmente para a transmissão de alta qualidade de vídeo (HDTV – *High Definition Television*) e de áudio (*Surround* 5.1). Para isso, foram adotados os padrões MPEG-2 e Dolby AC-3 para a codificação de vídeo e de áudio, respectivamente. Em virtude, principalmente, da evolução do ATSC e a sua adoção por outros países, esse padrão foi estendido para comportar serviços de dados e interatividade.

O *middleware* DASE (*DTV Application Software Environment*) [ARIB04a] vinha sendo adotado pelo padrão ATSC. Entretanto, frente ao movimento de convergência dos padrões em relação ao *middleware*, o ATSC propôs um novo padrão denominado ACAP (*Advanced Common Application Platform* - CS/101 [ATSC04c]), que, entre outras vantagens, é compatível com a especificação GEM (*Globally Executable MHP* - TS 102 819 [ETSI04a]). Em relação ao GEM, abordado ao final deste capítulo, existe uma forte tendência da sua adoção por vários padrões internacionais de TV digital. Entre outras vantagens, essa tendência permite que aplicações criadas em um padrão possam ser executadas nos terminais de acesso projetados para outros padrões de TV digital.

Nas próximas subseções o DASE e o ACAP são apresentados.

2.1.1. DASE

Quando concebido, o padrão DASE nível 1 era restrito ao nível de interatividade local, pois não contemplava a existência de um canal de retorno. Atualmente, a versão DASE nível 3, além de contemplar interatividade, tem como finalidade a integração da Internet com a televisão, fornecendo programas como correio eletrônico e navegadores Web no terminal de acesso.

As aplicações DASE são classificadas como declarativas e procedurais. As aplicações declarativas são representadas por documentos hipermídia/multimídia escritos por meio de uma linguagem de marcação baseadas, por exemplo, em XHTML (e*Xtensible HyperText Markup Language*) [PAAÇ02]. Já as aplicações procedurais são programas que utilizam a tecnologia Java para a execução de instruções, agregando maior poder de processamento. A máquina virtual Java

recebeu extensões que permitem um desenvolvimento mais adequado ao mundo da TV Digital[1].

Deve-se ressaltar que uma aplicação DASE não é apenas declarativa ou procedural. De forma geral, as aplicações declarativas nesse *middleware*, freqüentemente, fazem uso de *scripts*. Complementarmente, uma aplicação procedural pode referenciar um conteúdo declarativo.

A Figura 2 ilustra a arquitetura geral do DASE. A camada superior dessa arquitetura representa as aplicações, que podem ser declarativas ou procedurais.

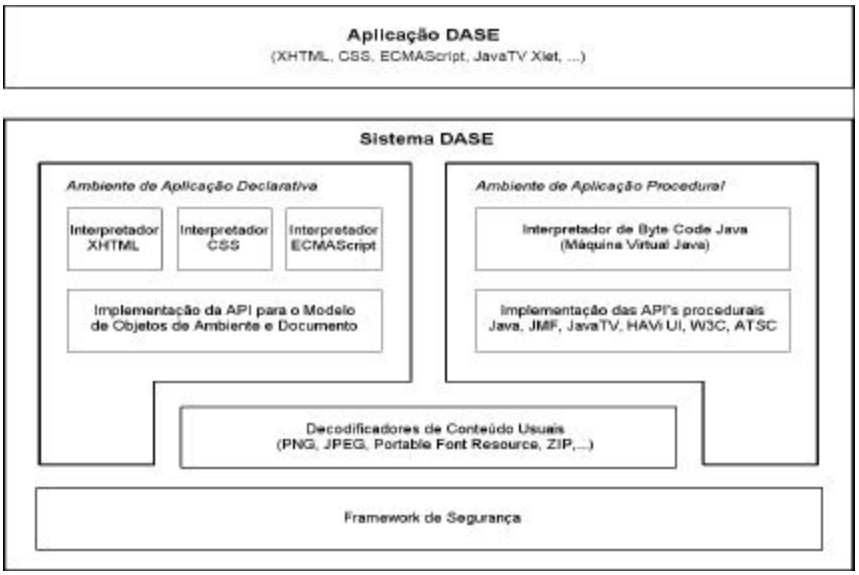

Figura 2 - Arquitetura DASE [SoCoRo04].

O sistema DASE é formado pelos módulos "Ambiente de Aplicação Declarativa" (*Declarative Applicantion Environment* ou DAE) e "Ambiente de Aplicação Procedural" (*Procedural Applicantion Environment* ou PAE). O DAE é um subsistema lógico que processa linguagens de marcação, folhas de estilos e scripts. Seu principal componente é o DCDE (*Declarative Content Decoding Engine*), o qual tem por finalidade realizar a análise sintática do documento. Por sua vez, o PAE é o módulo responsável por processar conteúdo de objetos ativos, ou executáveis. Seu principal componente é o PCDE (*Procedural Content Execution Engine*), que, por exemplo, contém a máquina virtual Java.

[1] No Capítulo 3 os tipos de linguagens adotados pelos padrões são abordados em maiores detalhes. No entanto, é importante ressaltar que os termos procedural e declarativa, não correspondem à classificação das linguagens utilizadas na autoria (Java e HTML).

2.1.2. ACAP

Com o intuito de definir um padrão de *middleware* único, e que fosse comum a todos os terminais de acesso de TV digital terrestre e a cabo nos Estados Unidos, um grupo de pesquisadores do ATSC e de empresas ligadas à padronização da TV digital americana reuniram-se para harmonizar os seus ambientes de aplicação originais: DASE-1 e OCAP (*Open Cable Application Platform*) [CABL02].

O padrão OCAP, desenvolvido principalmente pela *CableLabs*, é derivado do MHP (*Multimedia Home Platform*), e ajustado às características técnicas e de negócio do ambiente de difusão via cabo dos Estados Unidos.

O ACAP segue as especificações definidas no padrão GEM e utiliza o mesmo conjunto de APIs Java e o modelo de aplicação utilizado no MHP (será discutido mais adiante). A obrigatoriedade de um canal de retorno, o suporte a aplicações armazenadas e a versão de carrossel de objetos do DSM-CC, caracterizam algumas diferenças entre o ACAP e o MHP.

As aplicações no ACAP também são classificadas como declarativas e procedurais. Aquelas contendo apenas conteúdo procedural, escritas em Java e que podem combinar gráficos, vídeos e imagens, são denominadas aplicações ACAP-J. As aplicações declarativas são denominadas ACAP-X, e são representadas por documentos multimídia compostos através de uma linguagem de marcação, como XHTML, regras de estilo, scripts, vídeos e áudios. Vale ressaltar que uma aplicação ACAP não necessita ser inteiramente procedural ou declarativa. Uma aplicação ACAP-J pode referenciar um conteúdo declarativo, assim como uma aplicação ACAP-X pode, por exemplo, fazer uso de scripts.

A Figura 3 ilustra a arquitetura do ACAP para sistemas que suportam aplicações procedurais e declarativas. Como se pode notar há uma grande semelhança com a arquitetura DASE. Enquanto o ambiente ACAP-X é responsável por interpretar o conteúdo declarativo de uma aplicação ACAP, o ambiente ACAP-J é responsável pelo processamento do conteúdo procedural, através da máquina virtual Java e das APIs de extensão.

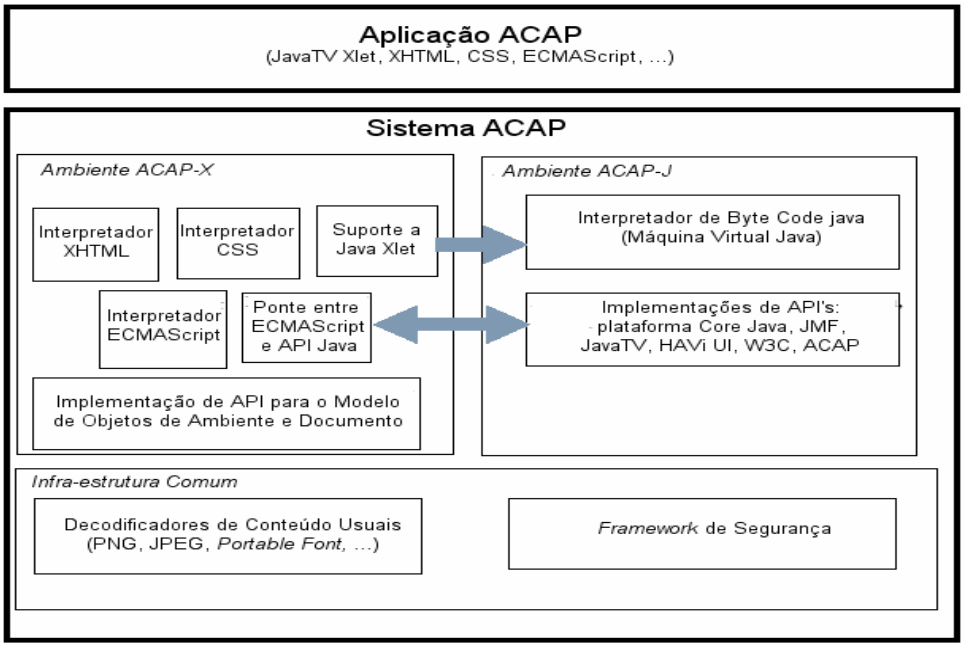

Figura 3 - Arquitetura do sistema ACAP [SoCoRo04]

As definições de uma aplicação procedural e do ambiente ACAP-J são baseadas nos padrões GEM e OCAP. Uma aplicação e o ambiente ACAP-J suportam os mesmos arquivos de classe Java definidos pelo MHP, além de algumas API's de extensão.

As definições de uma aplicação declarativa e do ambiente ACAP-X são baseadas no *middleware* DASE. Algumas alterações são necessárias, de forma a manter a compatibilidade com o padrão GEM.

2.2. DVB

O padrão DVB (*Digital Video Broadcast*) foi desenvolvido na Europa por um consórcio de empresas do ramo de TV. O nome de cada padrão é formado pelo nome do consórcio (DVB) adicionado de um sufixo, conforme a forma de transmissão. Por exemplo, DVB-S padroniza a TV digital via satélite; DVB-T define a TV digital terrestre, por difusão; e DVB-H é a especificação para a TV digital móvel (por exemplo, em celulares e computadores portáteis). Na sua concepção, o padrão DVB não foi formulado com o objetivo de transmissão em HDTV, mas devido às extensões que recebeu, passou a dispor de tal funcionalidade. São adotados como padrões a codificação de vídeo MPEG-2 e de

áudio MPEG-2 *Layer* 2. Entretanto, há uma tendência que no futuro o formato de áudio seja o AAC.

O MHP (*Multimedia Home Platform* - ES 201 812 [ETSI03a]), *middleware* do DVB, define uma interface genérica para que aplicações utilizem os recursos do terminal de acesso. Essa interface permite que softwares desenvolvidos por diferentes fabricantes possam ser executados em terminais distintos (portabilidade). O MHP permite que fornecedores de conteúdo digital possam endereçar suas aplicações para diferentes terminais de acesso, integrando TV digital e aplicações multimídia.

Assim como os *middlewares* do ATSC, o MHP também suporta dois tipos de aplicações: as declarativas (DVB-HTML) e procedurais (DVB-J). Uma aplicação MHP pode ser especificada em DVB-HTML, DVB-J ou de forma mista, como ilustrado na Figura 4.

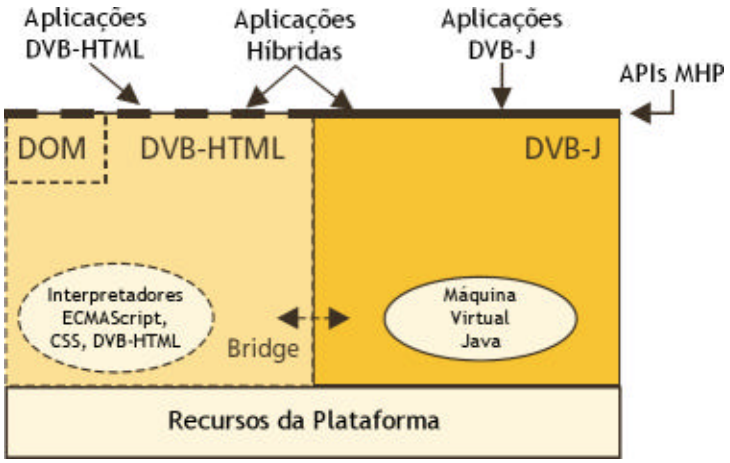

Figura 4 - Arquitetura básica do MHP [SoCoRo04].

As aplicações DVB-HTML são baseadas em documentos declarativos hipermídia que reúnem um conjunto de padrões para Internet, com destaque para XHTML. Complementarmente, as aplicações DVB-J, da mesma forma que as aplicações procedurais DASE, são Xlets. Mais detalhes sobre essas APIs podem ser encontradas em [FLEX04].

2.3. ISDB

O ISDB (*Integrated Services Digital Broadcasting*) é o padrão de TV digital desenvolvido pelo ARIB (*Association of Radio Industries and Business*), e utilizado no Japão. Em parte, o ISDB é derivado do padrão europeu DVB, mas

possui diferenças principalmente na codificação de áudio e no *middleware*. Um ponto de grande avanço na tecnologia ISDB é a recepção móvel de TV digital.

No ISDB, a codificação de vídeo segue o padrão MPEG-2 e o áudio é o AAC. Esse padrão é utilizado apenas no Japão, e sua documentação não é completamente aberta, nem gratuita. O *middleware* é comumente chamado de ARIB (ARIB STD-B23 [ARIB04a]), e atualmente possui compatibilidade com a especificação GEM.

A arquitetura ARIB é composta por duas partes: apresentação e execução. O modelo de execução utiliza a máquina virtual Java para a interpretação de *byte codes* que representam procedimentos/funções relacionadas ao conteúdo sendo transmitido. Por sua vez, o modelo de apresentação especifica a sintaxe da linguagem de marcação BML (*Broadcast Markup Language*).

A Figura 5 ilustra a arquitetura do *middleware* japonês e destaca a divisão entre a parte do protocolo responsável pela apresentação e a responsável pela execução.

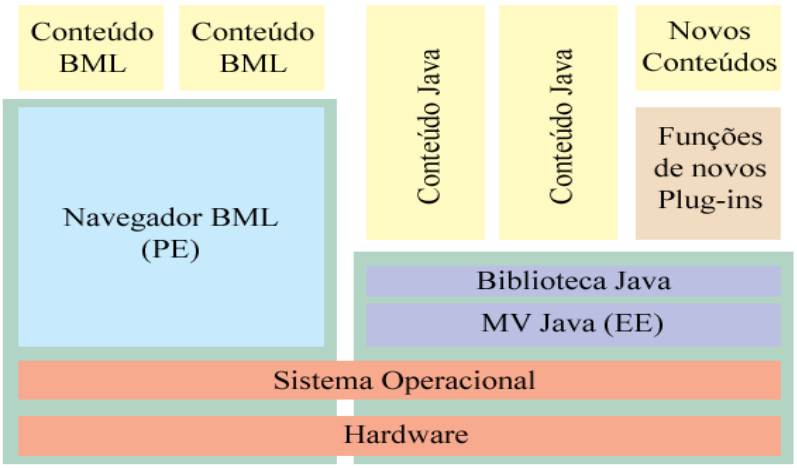

Figura 5 - Arquitetura do *middleware* do padrão japonês [SoCoRo04].

O modelo de apresentação corresponde ao subsistema responsável por manipular aplicações declarativas. A linguagem BML inclui marcadores e atributos utilizados para a autoria de conteúdo declarativo de TV Digital. O escopo de sua aplicação corresponde ao definido conforme os requisitos dos serviços multimídia. A codificação BML é definida como uma linguagem baseada em XML (*eXtensible Markup Language*) [W3C04a]. Complementarmente, o padrão permite a especificação de aplicativos através da linguagem Java, com bibliotecas próprias, que recebe o nome de ARIB-B23.

Uma das principais funções da linguagem BML é permitir o controle da exibição espacial e temporal, definindo onde, quando e o que será exibido na tela. Outras funções importantes incluem a manutenção do relacionamento entre informação e mídia (exibição de programas e URLs relacionados ao fluxo de áudio/vídeo), estruturação de envio de informações (títulos, artigos, legendas etc.) e a exibição de interfaces gráficas para a requisição de uma informação por meio da seleção de ícones ou outros objetos.

Já o modelo de execução foi concebido baseado na utilização de uma máquina virtual Java (instalada no terminal de acesso) para permitir a interpretação de *byte codes* Java. Esse modelo foi criado em conformidade com as especificações do GEM, a fim de se obter uma maior interoperabilidade.

2.4. GEM

Devido a rápida evolução e popularização do *middleware* MHP entre diversos países que utilizam o padrão DVB, surgiram iniciativas para sua implementação sobre outras plataformas internacionais. Entidades responsáveis pela padronização de sistemas de TV Digital fora da Europa manifestaram interesse em participar desse esforço, desejando implementar partes do *middleware* MHP em suas especificações, como forma de se aproveitar o desenvolvimento tecnológico e manter uma compatibilidade que permitisse que as aplicações pudessem ser executadas/apresentadas em terminais de acesso de outros padrões.

Para que aplicações MHP pudessem ser utilizadas sobre outras plataformas, conforme requisitado por entidades tais como *CableLabs* (EUA) e ARIB (Japão), o grupo DVB tomou a decisão de propor uma especificação única chamada GEM *(Globally Executable MHP*) [ETSI04a]. Além de capturar as interfaces e toda a semântica definidas pelo MHP, o GEM também inclui necessidades impostas por outros padrões internacionais.

Formalmente, a especificação GEM não pode ser considerada como uma especificação completa para terminais de acesso. O correto é dizer que GEM é um *framework* a partir do qual uma implementação do *middleware* de um terminal de acesso pode ser instanciada. O padrão define, o conjunto de APIs, garantias semânticas e formatos de conteúdo com os quais as aplicações (agora globalmente

interoperáveis) podem contar para a constituição de serviços interativos, oferecidos sobre qualquer padrão internacional de TV digital. Para a criação de *middlewares* compatíveis, torna-se obrigatório referenciar de forma completa a especificação GEM, de forma a preencher todos os seus requisitos.

Como mencionado anteriormente, o GEM é um esforço que visa maximizar a interoperabilidade entre os diferentes padrões. Como ganhos dessa investida podemos destacar a economia de escala para toda a cadeia de difusão interativa.

A Figura 6 ilustra como o GEM se enquadra dentro dos padrões de TV digital existentes.

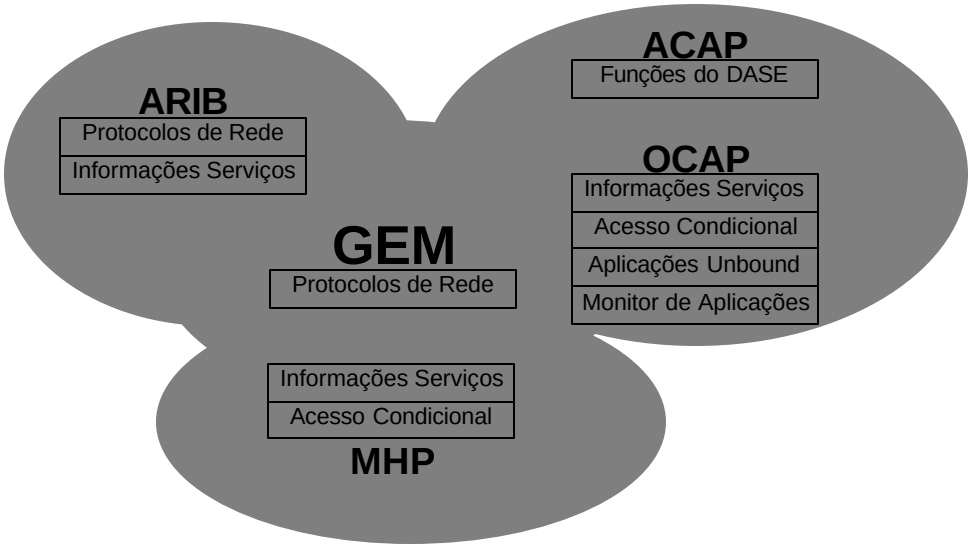

Figura 6 - Relação entre a especificação GEM e outros padrões [SoCoRo04].

3 Linguagens Declarativas, Sincronismo e Interatividade

Os padrões atuais para TV digital, conforme apresentado no Capítulo 2, oferecem linguagens para a especificação da interatividade e do sincronismo nas aplicações. Essas aplicações, usualmente classificadas pelos padrões em procedurais e declarativas, são concebidas, em sua maioria, por linguagens baseadas em Java e HTML, respectivamente[2].

As aplicações desenvolvidas em Java são criadas a partir de um conjunto de classes (pacotes) oferecidas pelo middleware. Para desenvolver essas aplicações, é necessário que o autor conheça as funcionalidades das classes do middleware. Nesse contexto, também é necessário que o autor possua conhecimentos de programação e orientação a objeto, como encapsulamento, herança, polimorfismo etc.

Complementarmente, as aplicações desenvolvidas em HTML, a princípio, favorecem a autoria ao permitir a especificação das aplicações através de marcações XML (XHTML). No entanto, a aplicabilidade do HTML restringe-se à formatação para apresentação e a definição de relações de referência entre documentos. Para aplicações mais elaboradas, incluindo, usualmente, sincronismo e outros tipos de interatividade, são adicionalmente utilizadas linguagens baseadas em script. Essas linguagens definem estruturas de controle, variáveis e procedimentos, que podem, eventualmente, acessar funções oferecidas pelo *middleware.*

Em comum, as aplicações classificadas como procedurais, especificadas em Java, e declarativas, especificadas através de XHTML e linguagens de scripts, como ECMAScript, apresentam restrições à autoria, pois requerem conhecimentos de programação para serem utilizadas no desenvolvimento de aplicações. A Seção

[2] Nesse caso, as nomenclaturas das aplicações não correspondem às classes das linguagens utilizadas para a concepção. A linguagem Java é destinada a criação de aplicações orientadas a objeto, enquanto HTML é voltada para a formatação de documentos. Nos padrões de TV digital existentes, o sincronismo e interatividade em HTML são definidos, primariamente, através de linguagens de scripts.

3.1 apresenta como as linguagens Java e XHTML são adotadas no desenvolvimento de aplicações voltadas para TV digital, apresentando suas vantagens e desvantagens.

Além das linguagens Java e HTML, as especificações de eventos entre objetos de mídia, incluindo-se eventos de sincronismo e interatividade, podem ser especificadas por linguagens que representam modelos hipermídia. A Seção 3.2 apresenta algumas dessas linguagens, conhecidas como linguagens declarativas, pois, além de possuírem uma estrutura de marcação (XML), cada marcação possui um significado semântico que permite representar, inclusive, eventos de sincronismo e interatividade.

3.1.Linguagens nos Padrões de TV digital

3.1.1.Linguagens baseadas em Java

Nos padrões para TV digital, citados no Capítulo 2, a especificação das relações de sincronismo e interatividade podem ser realizadas através da linguagem Java. As linguagens DVB-J, ACAP-J e ARIB-B23, relativas, respectivamente, aos padrões DVB, ATSC e ISDB têm como principal diferença os conjuntos de classes que cada padrão implementa no *middleware*. Em todas essas linguagens, sem exceção, o programador deve modelar o conjunto de objetos da aplicação e a relação entre esses objetos.

Na especificação dos objetos através de uma linguagem orientada a objetos, é função do programador definir a estrutura de dados interna de cada objeto e as interfaces que serão utilizadas na comunicação com os outros objetos da aplicação (encapsulamento). Nesse paradigma de linguagem, objetos podem ser instanciados a partir de um conjunto hierárquico de classes definidas pelo *middleware*, herdando a estrutura de dados dessas classes e suas respectivas interfaces (herança). Além disso, durante a execução, um objeto pode ser instanciado a partir de uma das diversas classes dentro da hierarquia na qual o ele foi definido, adquirindo um comportamento distinto de acordo com a instância escolhida (polimorfismo).

Nas linguagens DVB-J, ACAP-J e ARIB-B23, as relações entre os objetos são especificadas através de métodos, onde o programador deve especificar uma

sucessão de passos que devem ser seguidos, utilizando estruturas de controle, variáveis e chamadas a outros métodos. As linguagens desses padrões são, portanto, baseadas em uma linguagem de programação de propósito geral, onde o programador dispõe de muitos recursos para a especificação da aplicação. Em contrapartida, existe uma razoável complexidade na especificação dessas aplicações.

Por serem baseadas em uma linguagem de propósito geral, as linguagens DVB-J, ACAP-J e ARIB-B23, não apresentam mecanismos diretos para a especificação de sincronismo e interatividade na construção de aplicações hipermídia. Essas especificações devem ser realizadas através programas, onde o programador deve especificar os recursos de implementação, de acordo com os métodos existentes nos conjuntos de classes (pacotes) oferecidas pelo *middleware*.

Entre os conjuntos de classes que podem ser encontrados nos diversos middlewares dos padrões DVB, ATSC e ISDB, podem ser destacadas as classes para o desenvolvimento de interfaces gráficas AWT (*Abstract Window Toolkit*) e JMF (*Java Media Framework*), além das classes para transmissão de dados DAVIC (*Digital Audio Video Interactive Consortium*) e para interconexão de dispositivos HAVi (*Home Audio/Video Interoperability*).

A Tabela 1 apresenta alguns dos principais pacotes definidos pelo DAVIC e pelo HAVi.

Tabela 1 - Pacotes DAVIC e HAVi.

Pacote	Descrição
org.davic.awt	Estensão da API AWT. Entre outras funcionalidades, define a classe Color, que permite estabelecer transparência para imagens e vídeos.
org.davic.media	Estensão da API JMF. Entre outras funcionalidades, define uma maior granularidade para o controle da exibição das mídias, aumentando o conjunto de eventos de controle.
org.davic.mpeg	Permite acessar os fluxos MPEG (*transport stream*, *elementary stream*).
org.davic.mpeg.sections	Permite acessar as seções privadas DSM-CC.
org.davic.net	Permite acesso geral ao conteúdo, onde o acesso pode ser realizado através de fluxos de transporte (*org.davic.net.tuning*) e também de forma controlada (*org.davic.net.ca*).
org.davic.resources	Informa os recursos disponíveis na estação onde o middleware está em execução.
org.havi.ui	Interface para acesso às classes org.havi.
org.havi.ui.event	Conjunto de classes que permite substituir as funcionalidades do pacote AWT. Inclui suporte a componentes para TV, suporte a sobreposição de camadas e captura de eventos gerados pelo controle remoto, entre outras funcionalidades.

Além dos pacotes citados no parágrafo anterior, os pacotes da implementação JavaTV também são usualmente encontrados no *middleware* dos padrões para TV digital. As classes dos pacotes JavaTV são apresentadas na Tabela 2.

Tabela 2 - Pacotes JavaTV.

Pacote	Descrição
javax.tx.xlet	Oferece interfaces para o gerenciamento do ciclo de vida de uma aplicação.
javax.tv.locator	Define referências para objetos transportados via carrossel de objetos.
javax.tv.net	Define interfaces para acesso a redes IP.
javax.tv.graphics	Define o container no qual as aplicações serão apresentadas.
javax.tv.util	Define classes com métodos úteis para o controle das aplicações, como a classe Timer, utilizada no sincronismo da aplicação.
javax.tv.media	Define controles para eventos JMF e acesso aos fluxos de dados MPEG (javax.tv.protocol).
javax.tv.service	Fornece acesso ao conteúdo da tabela *Service Information - SI* do fluxo de transporte. Define diversas classes que permitem acessar itens da tabela SI. Fornece acesso ao *Eletronic Program Guide – EPG* (javax.tv.service.guide) e às aplicações em uso (javax.tv.service.navigation).
javax.tv.carousel	Fornece acesso ao conteúdo transmitido via carrossel (diretórios, arquivos etc.).

Na implementação JavaTV, as classes definidas no pacote *javax.tx.xlet* merecem destaque por implementarem o ciclo de vida das aplicações. As aplicações criadas a partir das classes definidas nesse pacote são genericamente denominadas Xlets. Os Xlets são controlados por um gerenciador de aplicações (*application manager*) que faz parte do middleware e reside no set-top box [PeVu01]. As principais funções do gerente de aplicações são: sinalizar as mudanças de estados dos Xlets e controlar os recursos do middleware a partir da programação do Xlet.

Os Xlets podem estar residentes no set-top box ou serem recebidos por carrossel de objetos ou de dados. Todo o controle dos Xlets é realizado pelo gerenciador de aplicações, incluindo a inicialização, que pode ocorrer de duas formas: o Xlet pode estar temporalmente sincronizado com o fluxo de transporte MPEG-2, correspondente ao fluxo de áudio e vídeo principal, ou pode ser inicializado pela ação do usuário. Em ambas as formas, é importante que os Xlets possuam uma baixa latência de inicialização [PeCV01], a fim de preservar a sincronização da aplicação com o conteúdo audiovisual principal.

A Figura 7 apresenta os quatro estados de um Xlet: carregado (loaded), pausado (paused), iniciado (started) e destruído (destroyed). O gerenciador de aplicações é capaz de controlar múltiplas aplicações simultâneas, no entanto,

somente uma aplicação é visível em um dado instante de tempo. No contexto dos estados, somente um Xlet, em um dado instante de tempo, encontra-se no estado iniciado.

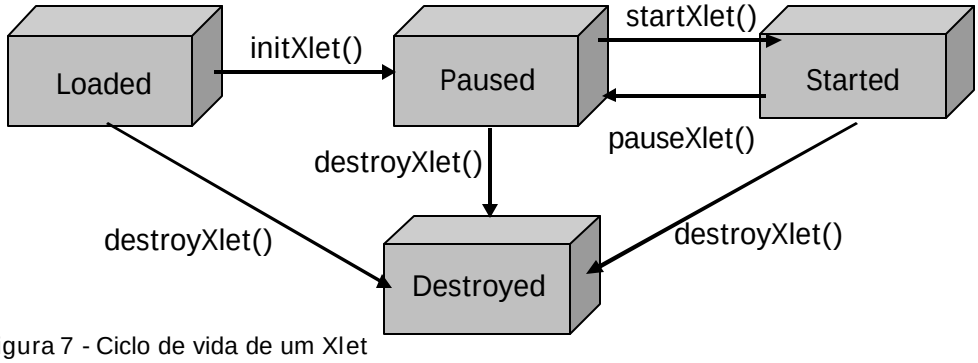

Figura 7 - Ciclo de vida de um Xlet

Quando o gerenciador de aplicações recebe um novo Xlet, o seu construtor é executado. Durante a execução do construtor, o Xlet encontra-se no estado carregado. Posteriormente, quando uma das condições de inicialização do Xlet ocorre, o método *initXlet()* da aplicação é executado, tendo como parâmetro o contexto[3] do Xlet. Nesse momento, o Xlet passa para o estado pausado. Logo em seguida ao método *initXlet()*, o método *startXlet()* é executado, passando o Xlet para o estado iniciado. A qualquer momento, durante o estado iniciado de um Xlet, o gerenciador de aplicações pode solicitar a execução do método *pauseXlet()* e retornar a aplicação para o estado pausado. Quando um Xlet termina sua execução, o seu método *destroyXlet()* é executado e os recursos alocados são liberados. As condições de término de um Xlet são idênticas às condições de inicialização.

3.1.2. Sincronismo e Interatividade das linguagens baseadas em Java

Conforme mencionado no início deste capítulo, para definir o sincronismo e interatividade de aplicações através da linguagem Java, é necessário que o autor tenha conhecimentos de programação. Nessa linguagem, considerada de propósito geral, os eventos de sincronismo e interatividade, bem como os demais eventos que podem ocorrer em uma aplicação multimídia, são, em sua maioria,

[3] As informações do contexto incluem, principalmente, os recursos alocados por um Xlet, como arquivos ou acesso à dispositivos de entrada e saída. Esses recursos somente são liberados quando o Xlet é destruído.

implementados através de observadores (*listener*). Esses observadores são instanciados nos objetos Xlets que constituem as aplicações.

Os observadores nas aplicações Java para TV digital podem ser utilizados desde o início da apresentação. Para receber o conteúdo dos objetos, preservando o controle sobre o conteúdo recebido, os Xlets podem utilizar *threads* Java para realizar o carregamento do conteúdo dos objetos de mídia de forma assíncrona.

Como exemplo de acesso assíncrono aos objetos de mídia, a linguagem DVB-J, através da classe *org.dvb.dsmcc.DSMCCObject*, implementa o acesso aos objetos transmitidos por carrossel. A classe *DSMCCObject* deve receber, no seu construtor, o objeto Java que instancia o carrossel e o caminho do objeto de mídia solicitado pela aplicação (string). O acesso ao objeto de mídia pode ser feito de forma assíncrona utilizando o método *asynchronousLoad()*, do objeto da classe *DSMCCObject*. Esse método recebe como parâmetro um observador, cuja classe é definida a partir da especificação da classe *java.util.EventListener*. Esta classe deve ser projetada para avisar a aplicação (Xlet), quando o objeto de mídia estiver carregado.

Para implementar os pontos de sincronização pertencentes a um fluxo (*stream events*), as linguagens baseadas em Java também utilizam observadores. Como exemplo, na linguagen DVB-J, o controle dos *stream events* é realizado pela classe *org.dvb.dsmcc.DSMCCStreamEvent*. Para instanciar um objeto da classe *DSMCCStreamEvent*, deve ser passado como parâmetro, ao construtor da classe, um objeto da classe *DSMCCObject*. Nesse caso, a aplicação irá monitorar os pontos de sincronização que forem recebidos e que estejam relacionados ao objeto passado como parâmetro ao construtor da classe.

Quando um objeto da classe *DSMCCStreamEvent* é instanciado, para cada tipo de *stream event* definido pela aplicação, deve ser definido um observador correspondente. Os observadores, definidos a partir da especificação da classe *java.util.EventListener*, devem ser associados aos tipos de *stream events* através do método *subscribe()* do objeto da classe *DSMCCStreamEvent*. Quando um tipo de *stream event* é considerado irrelevante pela aplicação, o observador associado a esse ponto deve ser desativado através da chamada ao método *unsubscribe()*.

Além dos eventos de sincronização do DSM-CC, as aplicações para TV digital, especificadas através das linguagens DVB-J, ACAP-J e ARIB-B23 podem capturar eventos de interatividade originados pelo usuário. Para definir esses

eventos, podem ser utilizadas as classes definidas no pacote HAVi, como a classe *org.havi.ui.event.HRcEvent*, que especifica os eventos das teclas encontrados em controles remotos.

Para exemplificar o controle de eventos relacionados à interatividade, considere a linguagem DVB-J, que define um repósitorio, relativo aos tipos de teclas cujo pressionamento deva ser monitorado pela aplicação. Esse repositório é instanciado através da classe *org.dvb.event.UserEventRepository*. O objeto repositório, instanciado a partir dessa classe, pode, por exemplo, conter os eventos relativos às teclas coloridas, através do método *addAllColorKeys()*, ou os eventos relativos às teclas numéricas, através do método *addAllNumericKeys()*, bem como outros eventos, relativos às demais teclas encontradas em controles de televisores.

Em um Xlet da linguagem DVB-J, a partir do definição do repositório, deve ser definido um gerenciador de eventos (*event manager*) para controlar, através de um observador, a ocorrência dos eventos relativos às teclas definidas no repositório. O gerenciador de eventos é instanciado através do método estático *getInstance()*, definido na classe *org.dvb.EventManager*. O observador, que implementa a classe *org.dvb.UserEventListener*, deve ser passado como parâmetro do objeto da classe *EventManager*, através do método *addUserEventListener()*. Os eventos retornados através do observador são definidos na classe *org.dvb.event.UserEvent*. A Figura 8 apresenta, como exemplo, um trecho de uma classe que especifica o controle de eventos no DVB-J.

```
class Example implements UserEventListener{

  public Example () {
    EventManager em ;
    UserEventRepository repository ;
    em = EventManager.getInstance () ;
    repository = new UserEventRepository ("R1") ;
    repository.addKey (UserEvent.VK_ENTER) ;
    em.addUserListener ((UserEventListener)this, repository) ;
    em.addResourceStatusEventListener (this) ;
  }

  /**
   * método definido pela interface UserEventListener.
   */
  public void UserEventReceived (UserEvent e) {

  }
```

Figura 8 - Exemplo do controle de eventos no DVB-J.

Na Figura 8, a classe denominada *Example* implementa o observador de eventos *UserEventListener*. No construtor da classe *Example* são instanciados o

gerenciador de eventos, através do objeto *em*, e o repositório, denominado "R1", através do objeto *repository*. O repositório contém apenas o evento da tecla de confirmação (*VK_ENTER*). Quando essa tecla é pressionada, o método *UserEventReceived()* do observador e acionado, recebendo como parâmetro o objeto *e* da classe *UserEvent*. A classe *UserEvent* define uma métodos para o controle dos eventos, como o método *getCode()*, que retorna o código da tecla pressionada.

3.1.3. Linguagens baseadas em HTML

Nos padrões de TV digital, adicionalmente às linguagens baseadas em Java, são definidas linguagens baseadas em HTML para a especificação de aplicações. Conforme mencionado no início deste capítulo, as linguagens baseadas em HTML favorecem, a princípio, a autoria, em função da simplicidade e da difusão do HTML como linguagem para formatação de documentos na WWW. É importante ressaltar, no entanto, que as funcionalidades do HTML são restritas quando é necessário realizar especificações de sincronismo e de interatividade. Para superar essa limitação é realizada a adoção do HTML em conjunto com outras linguagens de script. Porém, nesse caso, a complexidade da especificação torna-se próxima a complexidade da programação procedural ou orientada a objetos [4].

De forma similar as linguagens dos padrões de TV digital baseadas em Java, as linguagens baseadas em HTML utilizam um conjunto de recursos oferecidos pelos middlewares. Entre esses recursos, podem ser destacados os interpretadores de tecnologias relacionadas ao HTML, como o XHTML, DOM (*Document Object Model*), CSS e XML. Em relação as linguagens DVB-HTML, ACAP-X e BML/B-XML, definidas, respectivamente, pelos padrões DVB, ATSC e ISDB podem haver variações em relação à versão ou a disponibilidade dos recursos no middleware, porém, no contexto geral, as linguagens suportam os recursos citados e, adicionalmente, uma linguagem de script.

Normalmente, as linguagens baseadas em HTML possuem um ciclo de vida, similar ao ciclo das linguagens baseadas em Java. No DVB-HTML, por exemplo,

[4] As linguagens de script, como o ECMAScript, usualmente podem ser definidas através de construções típicas do paradigma procedural, ou, em alguns casos, utilizando objetos e as características do paradigma orientado a objetos.

podem haver várias aplicações baseadas em HTML simultâneas, controladas por uma entidade denominada agente do usuário, residente no middleware. A Figura 9 apresenta os cinco estados de uma aplicação DVB-HTML: carregando (loading), pausado (paused), ativo (active), desativado (killed) e destruído (destroyed).

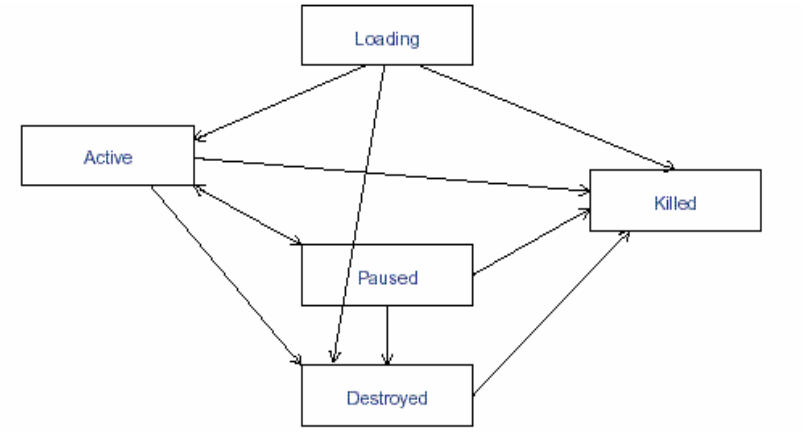

Figura 9 - Ciclo de vida de uma aplicação DVB-HTML.

Quando uma aplicação DVB-HTML é recebida, ela encontra-se no estado carregando. Nesse estado, a aplicação deve realizar o acesso ao conteúdo dos objetos de mídia referenciados. Caso todos os conteúdos iniciais estejam disponíveis, a aplicação é alterada para o estado ativo. Uma vez no estado ativo, uma aplicação pode ser transferida para o estado pausado, em virtude de alguma restrição nos recursos necessários à aplicação ou ainda, devido ao fato da aplicação ter provocado alguma violação das permissões de acesso aos recursos. Uma aplicação no estado pausado pode ser reativada e voltar para o estado ativo quando terminam as restrições que levaram a aplicação a este estado. Quando a aplicação termina sua execução ela muda para o estado desativado. Nesse estado todos os recursos utilizados pela aplicação são liberados. Finalmente, quando a aplicação é removida do sistema, define-se que a aplicação encontra-se no estado destruído.

As linguagens baseadas em HTML, utilizadas nos padrões para TV digital, são, no geral, integradas às linguagens baseadas em Java do mesmo padrão. Nos padrões DVB e ATSC, todos os recursos do middleware, disponíveis para as aplicações Java, estão disponíveis para as aplicações baseadas em HTML. O acesso às API's dos middlewares é oferecido através de construções da linguagem ECMAScript.

Além do acesso às API's Java do middleware, nos padrões DVB e ATSC, as linguagens baseadas em HTML podem exibir aplicações construídas nas linguagens DVB-J e ACAP-J, respectivamente, como se fossem mais um conteúdo sincronizado ao audiovisual principal. Como exemplo, a Figura 10 apresenta um trecho de programa definido em DVB-HTML, onde o elemento *object*, responsável pela exibição de aplicações especificadas em DVB-J, é destacado. No exemplo da Figura 10, o Xlet *MyApplication.class* será exibido na região delimitada pelo elemento *object* e receberá como parâmetros os valores passados por *arg_0* e *arg_1*.

```
<object type = "application/dvbj" id = "myApplication"
        codetype = "application/javatv-xlet"
        classid = "MyApplication.class"
        codebase = "myApplication/"
        height = "200" width = "150">

   <param name = "arg_0" value = "param1" />
   <param name = "arg_1" value = "param2" />

   <embed height = "200" width = "150"
          arg_0 = "param1" arg_1 = "param2">
   </embed>
</object>
```

Figura 10 - Exemplo de chamada DVB-J no DVB-HTML.

3.1.4. Sincronismo e Interatividade nas linguagens baseadas em HTML

No contexto geral, as linguagens baseadas em HTML, DVB-HTML, ACAP-X e BML/B-XML, tratam os eventos de sincronismo e interatividade de forma similar às linguagens DVB-J, ACAP-J e ARIB-B23, baseadas em Java. Essa característica deve-se ao uso do HTML em conjunto com o ECMAScript, que implementa características do paradigma orientado a objetos.

Para implementar os pontos de sincronização pertencentes a um fluxo (*stream events*), as linguagens baseadas em HTML, no geral, também utilizam observadores (*listener*). Em ECMAScript os observadores podem ser definidos em relação a eventos DOM. Dessa forma, para permitir aplicações especificadas em HTML, os middlewares realizam a conversão de *stream events* para eventos DOM.

Como exemplo, no middleware do DVB, a especificação da conversão de *stream events* para eventos DOM é realizada através das especificações contidas

em um arquivo XML denominado *event factory*. Esse arquivo deve ser colocado no mesmo diretório DSM-CC do arquivo DVB-HTML, com o mesmo nome, porém com a extensão *lnk*. A Figura 11 apresenta um arquivo DVB-HTML que define uma função a ser executada quando o evento denominado *myTriggerEvent* for recebido.

```
<?xml version =”1.0”?>
   <!DOCTYPE html PUBLIC “//DVB//DTD XHTML DVB-HTML 1.0//EN”
      “http://www.dvb.org/mhp/dtd/dvbhtml-1-0.dtd” >
   <html xmlns = “http://www.w3.org/1999/xhtml”
         xmlns:dvbhtml = “http://www.dvb.org/mhp” >

      <head>
         <script type = ”text/ecmascript”>
            function handleEvent(evt) {
              /*código do evento*/
            }
            function setupEventListeners() {
              var htmlNode = document.documentElement;
              htmlNode.addEventListener(“myTriggerEvent”, handleEvent,
true);
            }
         </script>
      </head>

      <body dvbhtml:onload = “setupEventListeners()”>
      </body>

   </html>
```

Figura 11 - Manipulação de *stream events* no DVB-HTML

No exemplo da Figura 11, quando o arquivo DVB-HTML é carregado, a função *setupEventListeners()* é executada. Essa função apenas informa ao observador do documento, através do método *addEventListener*(), que a função *handleEvent()* deve ser executada quando o evento denominado *myTriggerEvent* for recebido. Conforme mencionado anteriormente, a associação entre o nome *myTriggerEvent* e os eventos *stream events* que são recebidos pela aplicação é definida no arquivo *event factory*.

Nas linguagens baseadas em HTML, os eventos de interatividade podem, a princípio, ser definidos através de relacionamentos de referência, utilizando os elos HTML. Os elos são disparados por eventos de interatividade, como o pressionamento de uma tecla específica do controle remoto e podem ter como ação a exibição de um novo documento. Além de exibir um novo documento, outras ações podem ser executadas, como, por exempo, a execução de funções ECMAScript.

A Figura 12 apresenta, como exemplo, um trecho de um documento DVB-HTML contendo um elo que define uma relacionamento de referência para outro documento. No exemplo, o elo, representado pelo elemento *a*, define o atributo *accesskey* com valor *&VK_GUIDE;* que equivale a ação de pressionar a tecla do guia de programação no controle remoto. No DVB-HTML os mneumônicos relativos as teclas do controle remoto são definidos no DTD (*Document Type Definition*) dessa linguagem.

```
<?xml version =”1.0”?>
   <!DOCTYPE html PUBLIC “//DVB//DTD XHTML DVB-HTML 1.0//EN”
      “http://www.dvb.org/mhp/dtd/dvbhtml-1-0.dtd” >
   <html xmlns = “http://www.w3.org/1999/xhtml”
         xmlns:dvbhtml = “http://www.dvb.org/mhp” >

      <head> </head>

      <body>
         <a accesskey=”&VK_GUIDE;” href=”guide/contents.html”>
           informações do guia de services
         </a>
      </body>

   </html>
```

Figura 12 - Elo de interatividade no DVB-HTML

3.2. Linguagens Declarativas

Para superar a complexidade no desenvolvimento de aplicações para TV digital utilizando linguagens baseadas em Java ou em HTML com scripts, linguagens declarativas podem ser utilizadas. As linguagens declarativas favorecem a autoria, pois permitem representar as relações entre eventos em documentos multimídia/hipermídia sem que seja necessário especificar estruturas complexas, como estruturas de controle, repetição, variáveis, observadores etc.

Atualmente, as linguagens declarativas não são encontradas, de forma efetiva, nos padrões para TV digital. No entanto, essas linguagens são amplamente difundidas para a modelagem de aplicações multimídia/hipermídia, com destaque para as linguagens SMIL [BuRu04] e NCL [MuSS03][5].

[5] Além dessas linguagens, outras poderiam ser citadas. Entre essas outras linguagens, a linguagem declarativa do padrão MPEG-4 (...,...), XMT-O (...,...) poderia ser utilizada para especificar eventoss nas aplicações para TV digital. No caso específico de XMT-O, os módulos de

A linguagem SMIL (*Synchronized Multimidia Integration Language*) é uma linguagem baseada em XML, e especificada pelo W3C [W3C01b], que permite a descrição de apresentações multimídia interativas. Essa linguagem é bastante intuitiva para o desenvolvimento de apresentações, pois é possível se definir de forma simples como os objetos de mídia estarão dispostos no tempo através de um conjunto básico de composições que possuem semântica de sincronização. São três os tipos de composições de sincronização temporal:

- Seqüencial (*seq*): os objetos de mídia são executados seqüencialmente, onde um objeto nunca começa antes que seu predecessor termine;
- Paralela (*par*): os objetos de mídia são executados em paralelo. Isso não significa que eles terminarão juntos, mas sim que esses objetos são iniciados simultaneamente;
- Exclusiva (*excl*): somente um objeto de mídia que compõe esse tipo de composição pode estar ativo por vez. A ativação desses objetos é geralmente feita sob demanda, como por exemplo, ao ocorrer um evento de clique (*begin="buttom1.activateEvent"*).

Vale destacar que as composições SMIL podem ser aninhadas para elaboração de uma apresentação, ou seja, uma composição pode ser composta por objetos de mídia e/ou outras composições, e assim recursivamente.

A NCL (*Nested Context Language*) é uma linguagem para autoria de documentos hipermídia baseados no modelo conceitual NCM (*Nested Context Model*) [SoRM03]. Para compreender as características dessa linguagem, é importante compreender as características do modelo no qual ela é baseada.

No modelo NCM um documento hipermídia é representado por um nó de composição, podendo conter um conjunto de nós, que podem ser objetos de mídia ou outros nós de composição, recursivamente, e ainda elos relacionando esses nós.

Nós de composição no modelo NCM não têm nenhuma semântica embutida, diferente das composições SMIL que, por exemplo, têm semântica temporal. A semântica dos relacionamentos entre os componentes de um documento é feita através dos elos NCM, que podem especificar relações de

---

sincronismo e interatividade são praticamente os mesmos da linguagem SMIL. Dessa forma, as definições de SMIL para TV digital também podem ser aplicadas no caso de XMT-O.

referência, relações de sincronização, relações de derivação, relações entre tarefas de um trabalho cooperativo etc. Elos NCM são definidos fazendo referência a um conector hipermídia, que pode representar qualquer tipo de relação. Além da referência a um conector, um elo define um conjunto de *binds* associando papéis do conector a nós do documento. Assim sendo, elos podem representar relacionamentos multiponto entre vários nós de um documento.

Elos são agrupados em bases de elos, logo, o conjunto de elos de uma composição é dado pela união de suas bases de elos. Além do reuso de nós, o NCM também permite o reuso de elos e de bases de elos, tornando mais flexível a definição de elos no modelo.

3.2.1. Sincronismo e Interatividade nas linguagens declarativas

Linguagens como SMIL e NCL são compostas por um conjunto de módulos, que agrupam elementos com funcionalidades semelhantes. Ambas as linguagens definem um conjunto de módulos destinados a suportar eventos de sincronismo e de interatividade.

Na linguagem SMIL, as composições são definidas nos módulos *BasicTimeContainers* (*par* e *seq*) e *ExclTimeContainers* (*excl*). Em SMIL, os relacionamentos, além das composições com semântica de sincronização, podem ser definidos pela igualdade de eventos. Utilizando os eventos de início da apresentação do nó (*begin*), término da apresentação do nó (*end*), seleção espacial (*click*) e alguns outros [BuRu04], é possível estabelecer relacionamentos de sincronização entre os nós (objetos de mídia, por exemplo) de um documento. O módulo *BasicInlineTiming* define os atributos *begin* e *end* que podem ser utilizados nos elementos que representam os objetos de mídia e as composições. O módulo *BasicInlineTiming* define também o atributo *dur*, que especifica a duração explicita de um objeto de mídia ou de uma composição.

Em relação às ações de interatividade, SMIL define eventos relativos asx ações do mouse (*mouse up*, *down*, *out* e *over*), no módulo *SyncbaseTiming*, e do teclado, no módulo *AccessKeyTime*[6].

Na linguagem NCL, os relacionamentos entre eventos são definidos através do perfil *XConnector* [MuSS03] da linguagem, que permite especificar instâncias de conectores, definindo relações entre eventos. Existem diversos tipos de eventos podem ser relacionados através de conectores, incluindo eventos de sincronismo e de interatividade.

Um conector especifica a relação, sem mencionar os nós que irão participar do relacionamento. Elos definem um relacionamento fazendo referência a um conector hipermídia e a um conjunto de associações, que definem os participantes do relacionamento.

A princípio, nas linguagens como SMIL, onde as composições possuem semântica de sincronização, a especificação de relacionamentos de sincronização torna-se mais fácil. No entanto, esse benefício é limitado pela existência de um conjunto simples de composições (paralela, seqüencial e exclusiva), o que dificulta a definição de relacionamentos complexos, podendo ser necessário, para esses casos, estabelecer composições com vários níveis de aninhamento. Além disso, essas composições obrigam o autor a estruturar o documento de acordo com a especificação para apresentação [MuSS03].

Na linguagem NCL, as composições podem herdar a semântica definida em um outro perfil da linguagem denominado *XTemplate* [MuSS03]. Os templates de composição especificam tipos de componentes, tipos de relações, componentes e relacionamentos que uma composição possui ou pode possuir, sem identificar todos os componentes e relacionamentos, pois essa especificação fica sob responsabilidade da composição que utilizar o template. Como caso particular pode ser definida a semântica das composições SMIL.

Em relação aos *stream events*, que podem ser recebidos pelo *middleware*, estabelecendo pontos de sincronização em relação a um fluxo, a linguagem SMIL não define um perfil que manipule esse tipo de evento através da linguagem. De

[6] Em XMT-O, adicionalmente a SMIL, é definido o módulo *XMTEvents* com outros eventos relativos à apresentação, tais como: colisão e aproximação de objetos em animações, visibilidade etc.

forma similar, na linguagem NCL não existem elementos nos módulos que possam formar um perfil voltado para o tratamento de *stream events*. No entanto, o uso da linguagem NCL para aplicações de TV digital prevê a recepção dos *stream events* pelo *middleware*, onde esses metadados serão interpretados para modificação da especificação da linguagem durante a exibição.

Em [MCRS05] é proposta uma arquitetura onde os *stream events* são tratados como metadados para edição do modelo, que a especificação em NCL de uma aplicação qualquer representa. É relativamente comum a interpretação dos *stream events* pelo *middleware*, nas linguagens dos padrões para TV digital, descritas na Seção 3.1. Nessas linguagens, os *stream events* são convertidos em eventos detectáveis aos observadores das linguagens, que, por sua vez, podem definir métodos para implementar as ações correspondentes aos eventos recebidos.

Na linguagem NCL, os *stream events* podem ser criados a partir de operações de edição realizadas durante a apresentação das aplicações. Entre outras vantagens, essa proposta preserva a estrutura original do documento no ambiente de execução. Essa proposta permite, por exemplo, que o exibidor torne-se autor de outras exibições posteriores, o que é relativamente comum nos sistemas de TV, onde podem ocorrer retransmissões realizadas por emissoras afiliadas.

Além das vantagens citadas, a proposta da linguagem NCL evita a necessidade de operações complexas de programação para definir métodos ou funções condicionais em relação aos *stream events* recebidos. Essa proposta integra os ambientes de edição e de execução, sendo transparente para o autor a criação ou a semântica dos *stream events*, cujo controle fica completamente a cargo do editor e do exibidor. O autor deve apenas se preocupar com a semântica do documento e não com a forma com que o sincronismo será implementado.

Nas propostas de *middleware* para SMIL, como em [PICV02], [LCHV03] e [ICECRE] não são citados como os *stream events* podem ser tratados. Nas propostas mencionadas, a linguagem SMIL é utilizada em conjunto com um *middleware* procedural, definido nos padrões para TV digital. A Figura 13 apresenta a arquitetura proposta em [ICECRE].

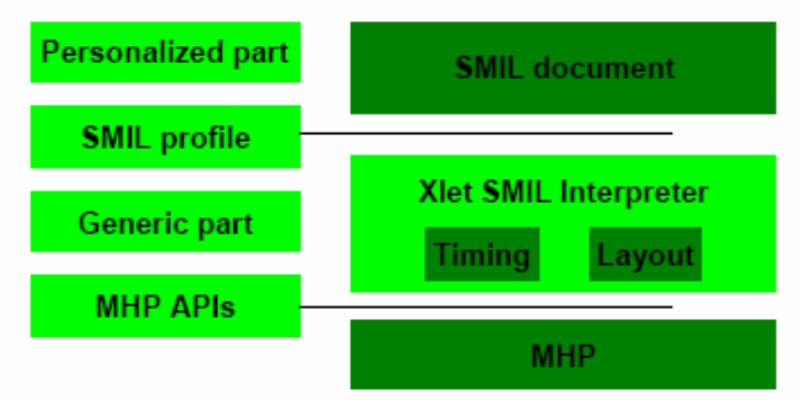

Figura 13 – Proposta de uma arquitetura SMIL para TV digital

A proposta apresentada na Figura 13 define um Xlet capaz de interpretar especificações de um documento SMIL. O Xlet, por sua vez, é implementado através da linguagem Java, sobre um *middleware* MHP. O Xlet acessa as funções do *middleware*, através da sua API, com o objetivo de interpretar e obedecer as especificações de um perfil SMIL voltado para TV digital. O perfil em questão é similar aos perfis SMIL que podem ser normalmente definidos através da linguagem. Nesse caso, a diferença consiste em alguns módulos, como o módulo *AccessKeyTime*, da área funcional Timing, que tem alguns elementos estendidos a fim de capturar eventos gerados pelas teclas do controle remoto.

A Figura 14 ilustra uma outra arquitetura proposta em [PICV02] e [LCHV03]. Na realidade, as características da arquitetura apresentada nesses trabalho é similar as características apresentadas em [ICECRE]. Na Figura 14, o gerenciador de aplicações controla Xlets que podem ser interpretadores SMIL ou outras aplicações qualquer.

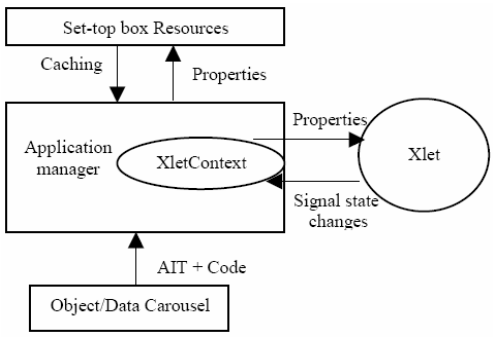

Figura 14 – Gerenciamento de Xlets contendo interpretadores SMIL.

4 Ferramentas de Autoria

Para facilitar e agilizar a criação de aplicações voltadas para TV digital, ferramentas de autoria podem ser empregadas a fim de abstrair do autor toda ou pelo menos parte da complexidade de se programar, muitas vezes através de uma linguagem de programação. Através dessas ferramentas, o autor pode se concentrar, principalmente, no processo de criação.

Existem diversas ferramentas de autoria para o ambiente de TV digital. Contudo, para realização deste trabalho, somente um número limitado delas pôde ser obtida em uma versão para demonstração. Este capítulo faz uma análise de algumas dessas ferramentas tanto para aplicações procedurais, quanto declarativas, levantando suas principais características. Algumas ferramentas para linguagens baseadas em modelos hipermídia, como o editor Maestro e o Grins, também foram analisadas.

4.1. JAME Author

JAME Author é um ambiente de autoria para a criação de aplicativos voltados para TV digital interativa no middleware MHP/OCAP, e faz parte de um grupo de soluções para iTV do *Fraunhofer Institute for Media Communication IMK* (http://www.imk.fraunhofer.de).

Devido a sua API Java (Xlet), o MHP possibilita a criação de aplicativos para TV digital interativa com várias funcionalidades. Em contra partida, todo esse poder pode tornar complexo o processo de criação, principalmente, quando um aplicativo precisa ser criado sem que existam aplicativos anteriores. O JAME Author visa facilitar a criação de programas atendendo a uma classe especial de aplicativos, chamados de *page-based services* (Serviços Baseados em Páginas). Esses aplicativos são compostos por páginas, nas quais se pode navegar de maneira muito semelhante a navegação Web. As páginas de um aplicativo são descritas através de uma linguagem chamada JAME PDL (*JAME Page*

Description Language) [UMJAME], a qual é baseada em XML. Na documentação analisada não são dados maiores detalhes sobre essa linguagem.

A ferramenta lida com dois tipos básicos de página. No primeiro, o funcionamento é similar com os das páginas da Web, onde a ativação de um link carrega uma nova página, e esta substitui a página corrente. No segundo caso, é utilizada sobreposição com transparência, e efeitos mais interessantes podem ser obtidos.

Todo projeto no JAME Author possui páginas de sistema pré-definidas para, por exemplo, lidar com mensagens de erro. Durante a elaboração de um projeto (aplicativo) novas páginas podem ser criadas para caracterizar o tipo de aplicação que está sendo desenvolvida. A Figura 15 apresenta a interface gráfica do JAME Author.

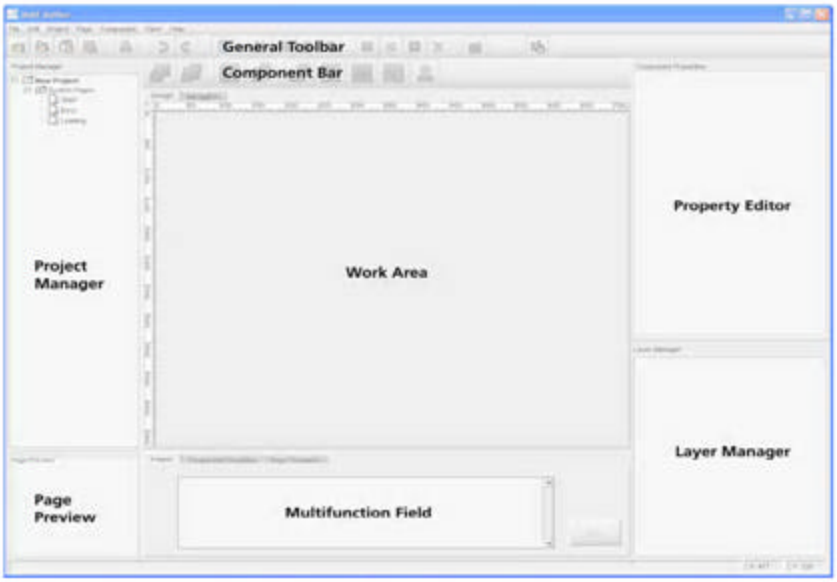

Figura 15 - Interface gráfica da ferramenta JAME Author [UMJAME].

O *Project Manager* é a região onde são listadas as páginas de um aplicativo e o *Page Preview* é a região de pré-visualização de uma página quando ela é selecionada no *Project Manager*.

O processo de edição se dá quando uma página listada no *Project Manager* é aberta na *Work Área*. Existem dois tipos principais de modos de edição (visões) suportados nessa região: o modo de design e o modo de navegação.

No modo de design, componentes da barra de componentes (*Component Bar*) podem ser selecionados e inseridos na página em edição. Um conjunto de

componentes padrão já vem definido na ferramenta de autoria, mas outros componentes também podem ser definidos pelo autor como modelos (*templates*). Tanto os modelos de componentes quanto os de página podem ser inseridos em um documento a partir da região *Multifunction Field*. Essa região funciona tanto como um repositório de páginas, componentes e figuras que podem ser inseridos ao projeto em tempo de criação, quanto como uma região informativa onde mensagens de validação são apresentadas ao usuário. Essas funcionalidades podem ser receber o foco através de tabulações do teclado.

No modo de navegação, é possível definir a estrutura de navegação intra e entre páginas. No primeiro caso essa navegação é definida através de regras de transferência do foco entre componentes. Essas regras definem como a mudança de foco deve ocorrer quando as teclas do controle remoto são pressionadas. Através da ação de arrasto do mouse do componente de origem até o componente de destino é criado um link de navegação. Além dos componentes envolvidos (origem e destino), é preciso definir qual a tecla do controle remoto que disparará o evento de mudança de foco. A Figura 16 exibe um link entre os componentes "Menu Item 1" e "Menu Item 2". O processo para se definir a navegação entre páginas é semelhante. Nesse caso, a diferença é que o destino deixa de ser um componente e passa a ser uma página.

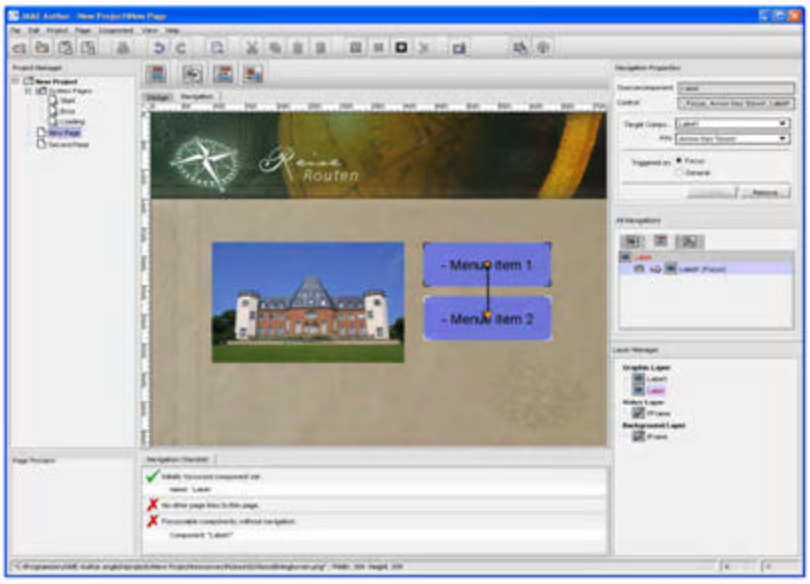

Figura 16 - Exemplo de definição de transferência de foco no JAME Author [QSJAME].

A região *Property Editor* (vide Figura 15) exibe as propriedades do componente selecionado quando uma página está aberta na *Work Area*, e permite

que elas sejam editadas. Por sua vez, a região *Layer Manager* lista as camadas suportadas, *Graphic*, *Vídeo* e *Background Layer*, em concordância com o modelo de referência do MHP, e em qual delas cada componente está definido.

O ambiente de autoria possui um emulador MHP integrado. Isso permite que o autor execute e valide seu trabalho à medida que ele é desenvolvido.

Pode ser observado que a criação de documentos hipermídia nessa ferramenta é similar a criação de documentos HTML (*HyperText Markup Language*) [W3C99a], o que vem a facilitar bastante a sua utilização por usuários não programadores em Java. Entretanto, na sua documentação não é mencionado nada que facilite ou possibilite a utilização de recursos mais sofisticados para sincronização temporal e/ou espacial com o áudio e vídeo principal, por exemplo. Por outro lado, assim como no HTML, é oferecido suporte a interatividade com o usuário.

4.2. Cardinal Studio

Cardinal Studio é uma ferramenta de autoria da Cardinal *Systems* (www.cardinal.fi) para a criação de aplicativos de TV digital interativa voltados para o *middleware* MHP.

O modelo de autoria do Cardinal Studio tem como abstração de mais alto nível os "Atos" (Acts) [CARDINAL]. Essas entidades servem para fazer a estruturação de um aplicativo e podem ser entendidas como cenas.

Os atos são compostos por componentes que podem ser de dois tipos: visíveis e invisíveis. O primeiro grupo contempla elementos visuais básicos como painéis, áreas de texto e botões. Esse grupo ainda é classificado em componentes de *background* e componentes focáveis. Praticamente todos componentes de um conjunto possuem um componente equivalente no outro conjunto. A diferença principal é que componentes focáveis podem ser usados na navegação com o controle remoto, enquanto que componentes de *background* não. Já os componentes invisíveis, como o próprio nome indica, não são desenhados na tela. Esses componentes usualmente comandam componentes visuais, o processamento de dados ou a transferência de dados entre componentes. Alguns exemplos de componentes invisíveis são o canal de retorno, uma conexão HTTP, um *Stream Event*, um botão do controle remoto, entre outros.

Dentro dos atos, os componentes são estruturados em camadas, sendo que cada camada pode ser compartilhada entre atos. A camada invisível é definida como um contêiner de componentes invisíveis. A camada gráfica é definida como o local onde ficam os componentes visíveis que são apresentados ao usuário. A camada de vídeo é definida como o local para exibição do vídeo selecionado pela propriedade *"Video source"*. Caso essa propriedade seja definida como nula, o vídeo principal do canal corrente é exibido. Finalmente, a camada *background* exibe um I-frame como plano de fundo.

No Cardinal Studio, toda interatividade é definida através de eventos disparados pelos componentes, como, por exemplo, o evento *actionPerformed* de um componente focável. Neste caso, esse evento é disparado quando o componente tem o foco e o um botão do controle remoto é pressionado. A ação executada pode ser personalizada de acordo com as intenções do autor.

A Figura 17 apresenta a interface gráfica do Cardinal Studio.

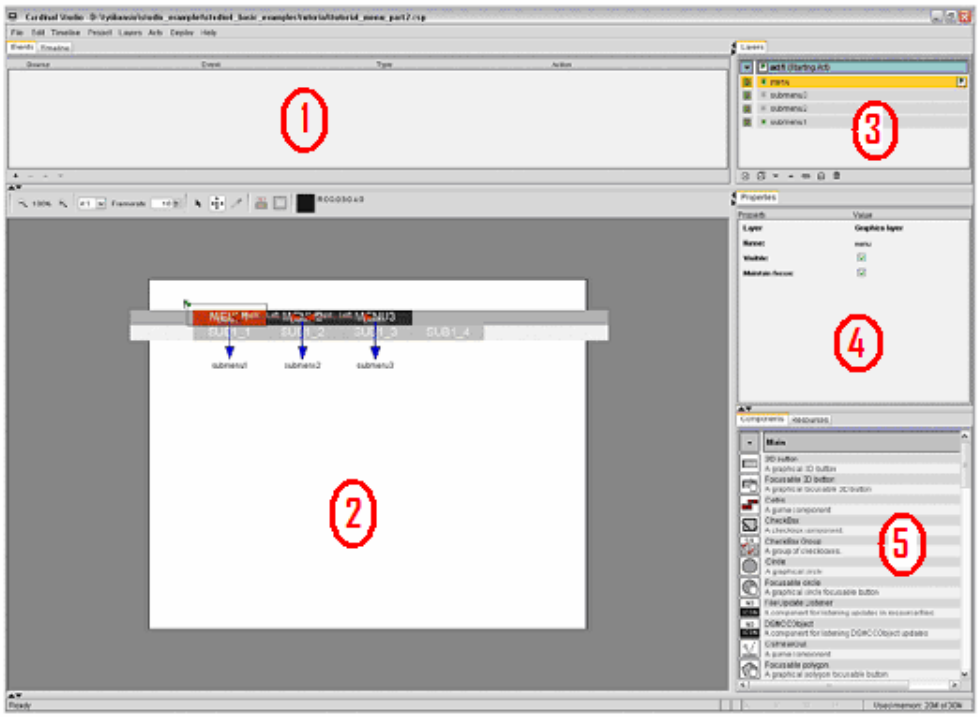

Figura 17 - Interface gráfica do Cardinal Studio.

A região 1 é formada pelo editor de eventos e pelo editor de *timeline* (a seleção entre esses dois editores é feita através da ação de tabulação do teclado). Como mencionado anteriormente, os eventos são disparados pela interação do usuário com o controle remoto e executam operações definidas dentro dos

componentes. Já o editor de *timeline* é usado para monitorar eventos e propriedades (de componentes) baseados no tempo.

A região 2 é chamada de *Canvas*, e é utilizada para dispor espacialmente os componentes na tela. Além disso, nessa região é possível adicionar e remover componentes. A região 3 exibe a lista de camadas de um ato. Nela é possível adicionar e remover atos e camadas. A região 4 é o local onde são exibidas as propriedades dos componentes selecionados. Tanto componentes visíveis quanto invisíveis possuem propriedades. Por fim, a região 5 lista um repositório de componentes que podem ser adicionados ao documento durante a fase de criação. Cabe ressaltar que dentro de um ato os componentes só podem ser adicionados de acordo com as características da camada corrente, como exemplo, somente componentes invisíveis podem ser inseridos na camada invisível.

A navegação especifica como será realizada a mudança de foco entre os componentes focáveis. O ambiente de autoria configura automaticamente essa navegação de acordo com os componentes inseridos. Para modificá-la a ferramenta oferece o modo de navegação. Esse modo exibe setas de um componente para outro, representando como será realizada a transferência de foco através do controle remoto. O autor tem a liberdade para definir qual será a seqüência utilizada no seu documento.

O Cardinal Studio foi desenvolvido para atender tanto a usuários programadores quanto a não-programadores em relação a linguagem Java. Para os não-programadores, a ferramenta oferece uma interface de mais alto nível de abstração, que permite aos autores investirem a maior parte do seu tempo no processo de criação. Por sua vez, os usuários programadores, familiarizados, principalmente, com os conceitos do *JavaBeans*, podem desenvolver componentes através do SDK, e integrá-los a ferramenta a fim de estender suas funcionalidades.

Esse ambiente de autoria também possui um emulador para testar e validar as funcionalidades dos aplicativos desenvolvidos. Além da apresentação do documento, um controle remoto é exibido para interação do usuário.

4.3. AltiComposer

Assim como as duas ferramentas descritas anteriormente, o AltiComposer é um ambiente de autoria para se criar aplicativos em conformidade com o DVB-MHP.

Essa ferramenta foi desenvolvida para dar suporte tanto a usuários não-programadores quanto aos que possuem conhecimento de programação em Java. Para o primeiro grupo, a ferramenta oferece uma interface gráfica intuitiva e funcional, onde vários recursos podem ser utilizados através do mouse. Por sua vez, usuários mais avançados podem extender a API *Component Development Kit* (CDK) da ferramenta para criar componentes personalizados.

Os nomes de componentes utilizados no AltiComposer foram definidos de acordo com os termos utilizados na industria de TV e cinema [UGALTI]. Os componentes existentes e suas relações de dependência são apresentados na Figura 18.

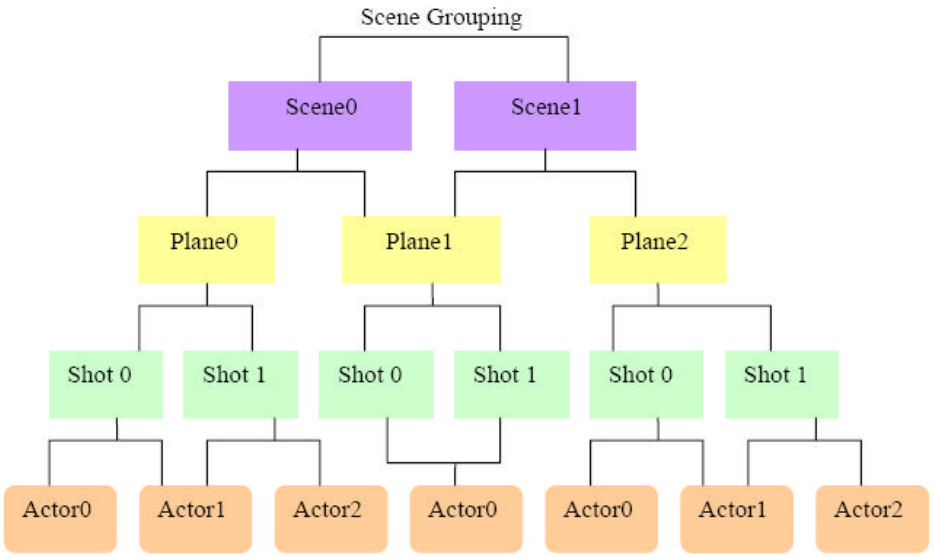

Figura 18 - Arquitetura de componentes do AltiComposer [UGALTI].

Uma cena (*Scene*) contém planos e que podem ser compartilhados com outras cenas. Um plano (*Plane*) contém vários *shots* e atores (*Actors*), mas não pode compartilhá-los com outros planos.

Na implementação da ferramenta, um ator é um objeto Java que pode ter um comportamento próprio. Esse comportamento define como o objeto responde, por exemplo, a uma interação do usuário através do controle remoto. São dois os tipos de atores: visual e não-visual. Um ator visual atua como um elemento gráfico que é apresentado ao usuário, enquanto que um ator não-visual serve para ajudar os aplicativos a funcionarem devidamente. Caixas de texto e figuras são exemplos de atores visuais, enquanto que a abstração de um botão do controle remoto é um exemplo de ator não-visual.

O *Shot* é a unidade básica na qual os usuários podem navegar no conteúdo interativo. Pode-se entender um *shot* como sendo um contêiner de atores. Em um dado instante de tempo, um *shot* pode ser visto como um *snapshot* (uma foto) da aplicação.

O plano é a unidade básica para salvar e carregar conteúdo dentro para apresentação. Já a cena é a unidade básica para se fazer a pré-carga de conteúdo para apresentação. Dessa forma, todos os atores, *shots* e planos são carregados junto à cena, antes do aplicativo ser iniciado.

O modelo de autoria dessa ferramenta define, além da arquitetura de componentes apresentada, um comportamento (*Behavior*). O nome desse *behavior* é definido pelo elemento ao qual ele está associado, como por exemplo, o "*Actor Behavior*" que está associado a um ator. Esse comportamento é um script que diz como cada componente deve responder a um determinado evento. A Figura 19 exibe o modelo de autoria do AltiComposer.

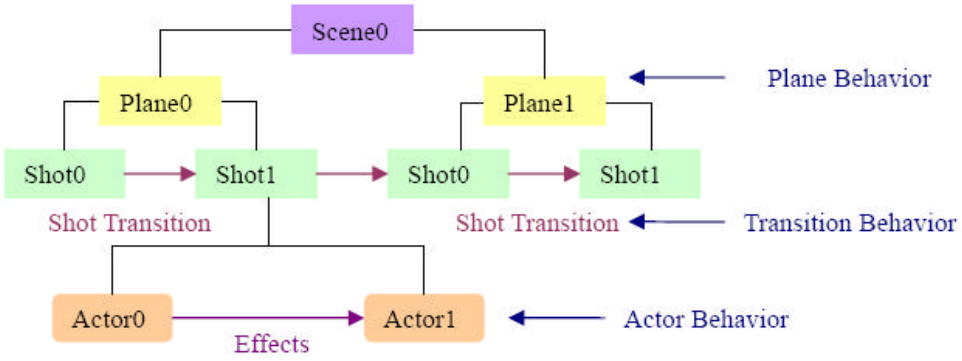

Figura 19 - Modelo de autoria do AltiComposer [UGALTI].

Um efeito é um objeto Java que leva um ator de um estado para outro. A restrição é que somente um ator compartilhado entre *shots* pode ter um efeito. Efeitos são usados para mudar propriedades ou a aparência visual de um ator. Uma transição de *shot* (*shot transition*) é um recurso geralmente usado para dar um efeito visual mais interessante a um aplicativo. É possível definir qual o tipo e quanto tempo durará a transição.

O que pode ser notado, é que os efeitos existem como recursos que estão prontos para serem usados. Já o *behavior* pode ser editado (codificando) em tempo de criação para realizar a operação desejada. Essa codificação do *behavior* é facilitada pelo uso de uma janela (*Script Editor*) que exibe, através de uma interface gráfica, as opções que podem ser selecionadas, e o código é

automaticamente gerado. A linguagem de programação utilizada é o AltiComposer Script Language [SGALTI], um subconjunto da ECMA-262 [ECMA262]. Usuários avançados podem programar de forma mais específica, tirando o máximo de proveito do modelo da ferramenta.

A Figura 20 exibe a interface gráfica do ambiente AltiComposer.

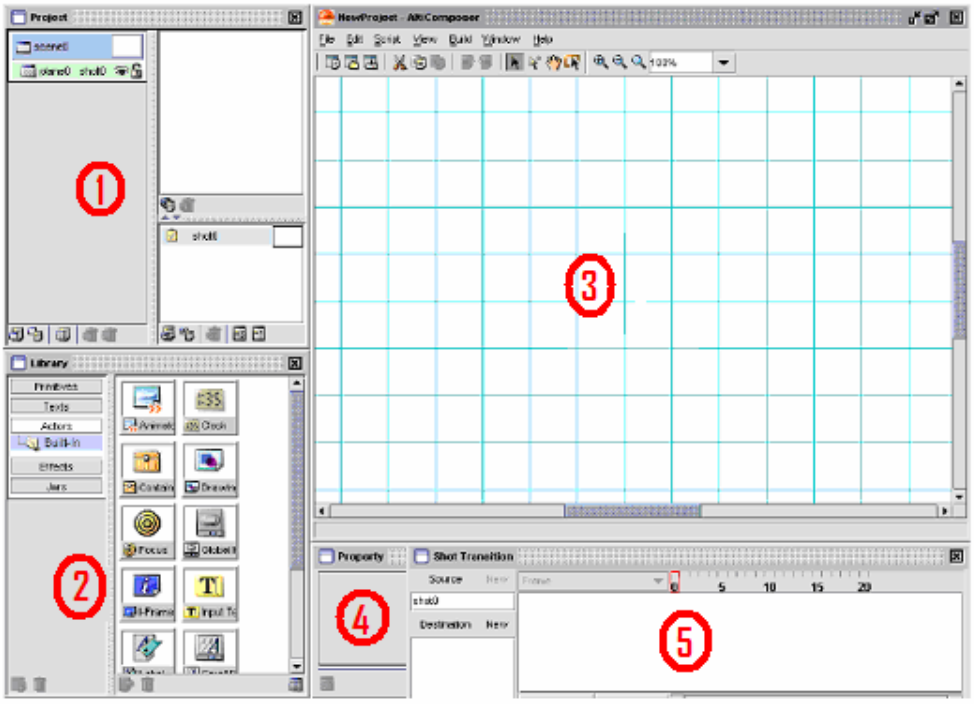

Figura 20 - Interface gráfica do AltiComposer.

A região 1 é conhecida como *Project Window* (Janela de Projeto). Na sua parte esquerda são listadas hierarquicamente as cenas e planos. Já na sua parte inferior direita são listados os *shots* de um plano, enquanto que acima são mostrados os atores pertencentes a um *shot*. O número 2 representa a *Library Window*, um repositório de recursos que podem ser utilizados durante a elaboração de um aplicativo. A região 3, conhecida como *Stage*, funciona como a área de trabalho para criação de cenas, planos e *shots*. A região 4 é a janela de propriedades, onde um elemento selecionado na área de trabalho pode ter suas propriedades editadas. Por fim, a região 5 representa a janela de transição, na qual pode ser definida como será a transição de *shots*.

Nessa ferramenta, a sincronização temporal e espacial são contempladas através da codificação do *behavior* de um determinado componente. A interatividade, assim como nas ferramentas anteriores, também é contemplada.

Por fim, vale ressaltar que esse ambiente de autoria também possui um emulador MHP que permite que o autor possa depurar seu trabalho à medida que esse é desenvolvido.

4.4. GR*i*NS

Uma forma de eximir do autor a necessidade de conhecer profundamente a linguagem SMIL é a utilização de ferramentas de autoria durante o processo de criação. O GR*i*NS Pro Editor for SMIL 2.0 é um ambiente de autoria para SMIL que possui várias visões integradas (temporal, espacial e textual). Contudo, essa ferramenta destaca-se por sua visão temporal para concepção de documentos.

Na sua visão temporal é possível montar e manipular a apresentação no decorrer do tempo de forma bastante intuitiva. Diferentemente da edição na linha do tempo normal, o paradigma de linha do tempo estruturado (*structured timeline*) exibe composições estruturadas que definem as relações lógicas entre os objetos de mídia. Outro detalhe é que todas as composições mencionadas anteriormente são exibidas em cores diferentes na visão temporal, e isso permite ao autor identificar facilmente de qual tipo se trata (vide Figura 21). Além das características mencionadas, o autor ainda pode ter uma estimativa do tempo da apresentação do documento.

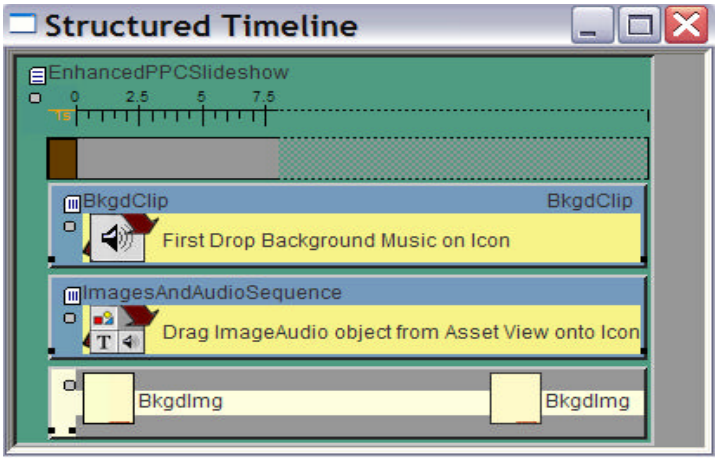

Figura 21 - Visão temporal do GR*i*NS.

A visão espacial tem como objetivo fornecer ao autor um modo prático e ágil de criar e manipular regiões nas quais os objetos de mídia serão apresentados no espaço. A Figura 22 apresenta a visão espacial do ambiente de autoria GR*i*NS. A região 1 exibe uma árvore que lista hierarquicamente as regiões definidas e os

objetos de mídia que serão exibidos em cada uma delas. Por sua vez, a região 2 representa o espaço onde o autor pode definir e manipular graficamente as regiões. Como propriedades importantes que precisam ser definidas nessa visão, pode-se citar o "*z-index*", as coordenadas espaciais e o tipo de ajuste da mídia a região. No caso desta última propriedade, os possíveis valores são "*meet*" e "*fill*". No primeiro caso, a mídia mantém sua resolução (tamanho), e se esta for maior que a região somente parte da mídia aparece. Com o valor "*fill*", a mídia se ajusta à área da região.

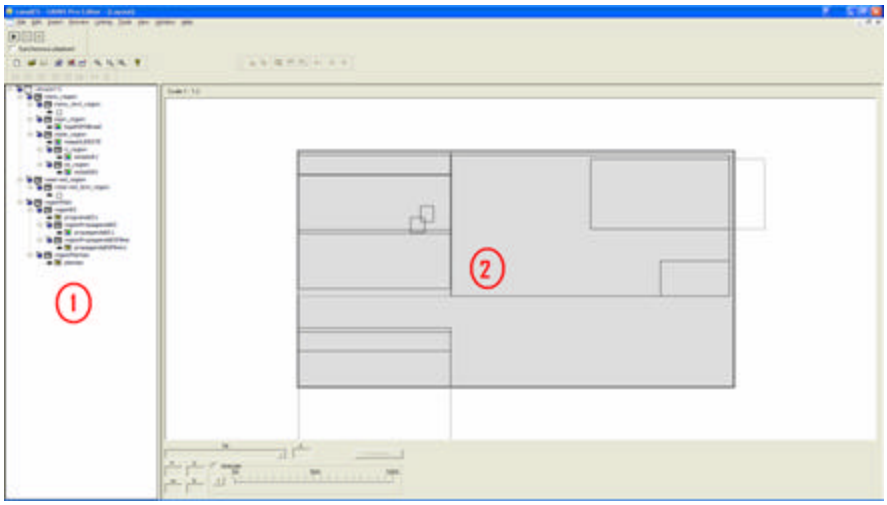

Figura 22 - Visão espacial do GRiNS.

Mesmo que na maior parte do tempo seja mais fácil se trabalhar nas visões gráficas, editar o código fonte em SMIL também pode ser bastante útil. É com essa intenção que a visão textual do GRiNS permite que o código seja não só visualizado como também editado, de forma que qualquer alteração nessa visão é devidamente refletida nas demais visões.

O editor da visão textual é bem simples, e oferece somente os recursos básicos encontrados na maioria dos editores textuais disponíveis no mercado. A Figura 23 exibe o código fonte de um documento nessa visão. Para que as modificações feitas possam ter efeito nas outras visões é necessário que o usuário clique no botão "*Apply*". Caso haja algum erro, o usuário é prontamente informado. Caso opte por desfazer as alterações, o usuário deve clicar no botão "*Revert*".

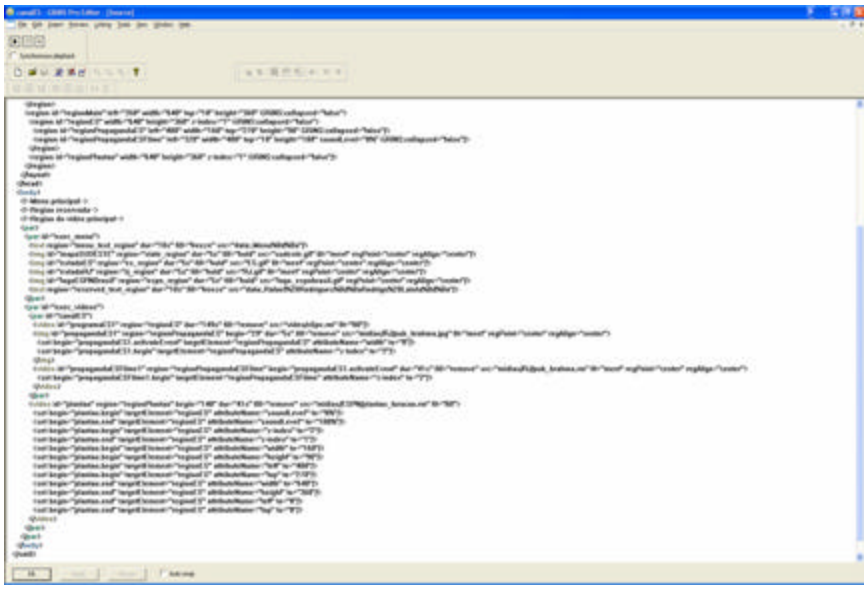

Figura 23 - Visão textual do GR*i*NS.

Um ponto fraco da visão textual é que somente as composições com semântica (*seq, par, excl* e *switch*) recebem cores de destaque. Se por um lado o destaque de um grande número de componentes deixaria o código poluído visualmente, a falta de destaque faz com que o autor tenha que despender de um maior esforço para identificar o que está procurando. Uma alternativa interessante a ser trabalhada seria o uso de filtros de cores que permitissem destacar o foco de interesse do autor de maneira mais conveniente.

A existência de várias visões permite ao autor lançar mão lançar das suas habilidades da forma que mais lhe convém. Vale destacar que se usadas complementarmente, pode ser explorado o que de melhor existe em cada uma das visões.

A existência de um *player* associado à ferramenta de autoria é de grande valia, pois em tempo de criação é possível ter noção de como está ficando a apresentação. Isso agiliza o processo de desenvolvimento e diminui consideravelmente o trabalho.

Vale destacar que o GR*i*NS privilegia à sincronização, diferentemente do que acontece nas ferramentas de autoria vistas anteriormente. Por outro lado, quando o assunto é interatividade, é muito mais fácil de se implementar graficamente naquelas ferramentas. No GR*i*NS, a implementação de eventos interativos é mais fácil na visão textual. Já que a interatividade é uma funcionalidade tão importante de ser provida em sistemas de TV digital interativa,

uma alternativa seria incorporar uma nova visão que permitisse definir eventos interativos de forma mais intuitiva.

4.5. Maestro Editor

A NCL é uma linguagem bastante atraente para o desenvolvimento de apresentações hipermídia interativas, pois nela é possível se definir de forma simples e intuitiva a sincronização temporal e espacial, bem como se definir eventos interativos. Isso tudo é possível através da utilização de elos causais/restrição que dão toda a semântica de sincronização necessária.

O Maestro Editor é uma ferramenta para auxiliar na construção de documentos NCL. Esse editor é dividido em quatro visões: uma visão gráfica estrutural, uma visão gráfica temporal, uma visão gráfica espacial e uma visão textual. As quatro visões funcionam de maneira sincronizada, a fim de oferecer um ambiente integrado de autoria. Além disso, as visões são providas de mecanismos de filtragem para auxiliar na especificação de arquiteturas mais complexas [Coelho04]. Além disso, assim como todas outras analisadas, essa ferramenta possui um *player* (formatador) associado.

A Figura 24 apresenta a interface gráfica do Maestro Editor. A região 1 exibe a base de documentos NCL na forma de uma árvore. A região 2 é a área de trabalho onde um documento pode ser editado em qualquer uma das visões. Na figura abaixo um documento está aberto e as quatro visões são mostradas.

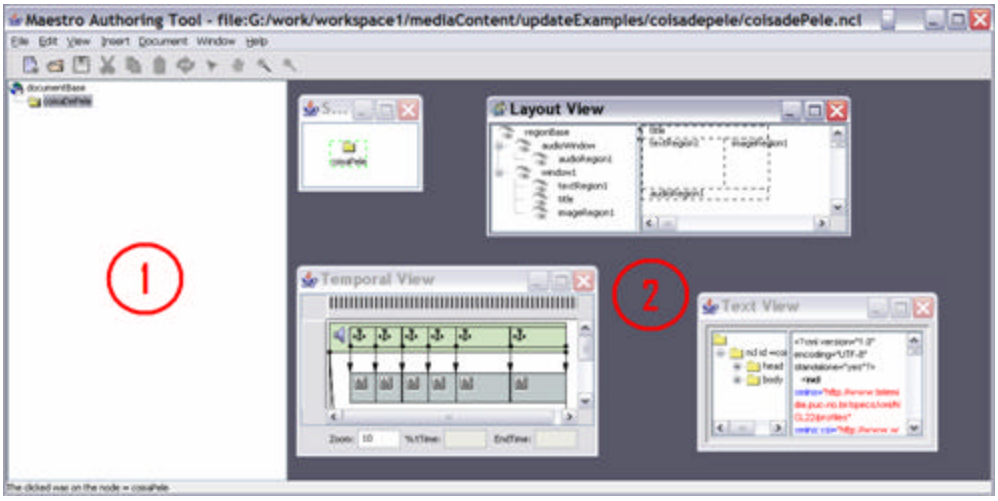

Figura 24 - Interface gráfica do Editor Maestro.

A visão textual (declarativa) permite ao autor editar o código NCL do documento que está sendo gerado. A Figura 25 ilustra o editor textual exibindo um documento NCL.

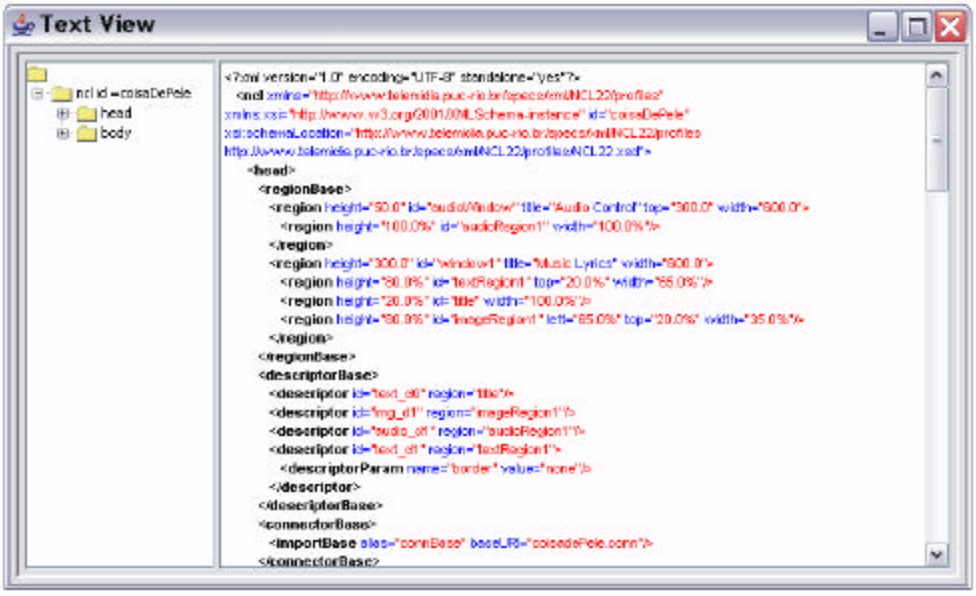

Figura 25 - Visão textual do Editor Maestro.

A interface do editor textual é dividida em duas partes. A área da esquerda apresenta uma árvore XML do documento, na qual os vértices compostos podem ser expandidos ou colapsados. Já a área direita apresenta todo o documento na forma textual.

O código apresentado no lado direito do editor pode ser exibido com o recurso de *highlight*, no qual o nome dos elementos fica em negrito, enquanto os atributos do elemento e seus valores aparecem em cores diferentes. O uso do *highlight* é opcional. O interessante é que as alterações feitas em um dos lados do editor são refletidas no outro lado e nas demais visões. Essa sincronização é feita através do botão *Refresh* ou através da opção *Refresh* do menu *Edit*.

Já a visão estrutural permite ao autor criar a estrutura lógica do documento NCL, ou seja, nela o autor pode criar, editar e apagar composições, objetos de mídia e elos. A Figura 26 exibe a interface gráfica da visão estrutural do ambiente de autoria Maestro Editor.

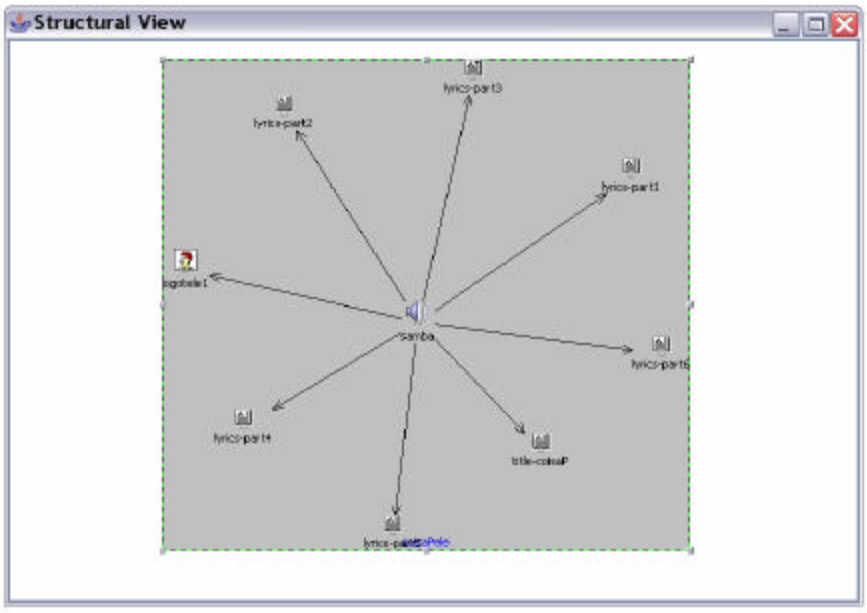

Figura 26 - Visão estrutural do Editor Maestro.

A visão temporal é responsável pela especificação dos relacionamentos temporais entre vértices de um grafo composto, definindo suas posições relativas no tempo. Os objetos de mídia na visão temporal são representados por retângulos, cujos comprimentos indicam suas durações de exibição/execução, conforme ilustrado na Figura 27. O eixo horizontal representa a escala temporal na qual os objetos de mídia se encontram. Conforme pode ser notado na figura abaixo, âncoras e elos também são representados graficamente nessa visão. Atualmente essa ferramenta está passando por um processo de reformulação que permitirá que modificações feitas nos objetos de mídia dessa visão sejam refletidas nas demais visões.

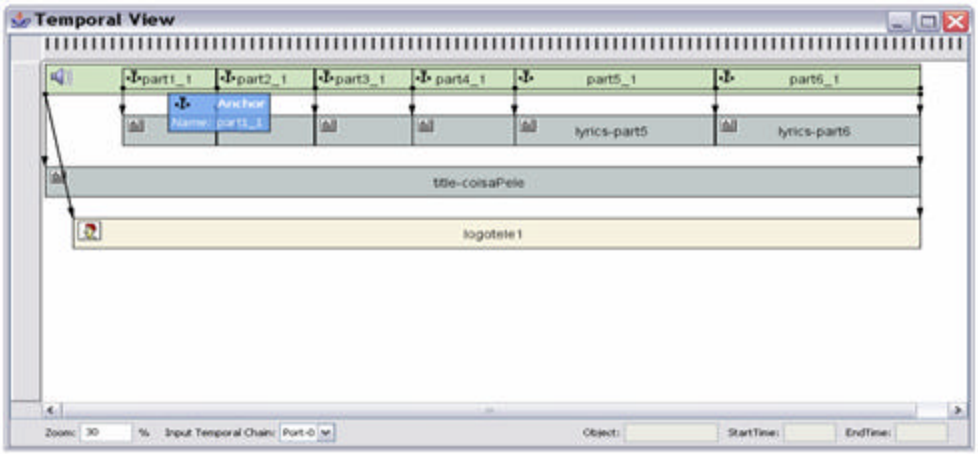

Figura 27 - Visão temporal do Editor Maestro.

Já a visão espacial possibilita ao autor definir graficamente como as regiões, onde os objetos de mídia serão apresentados, estarão dispostas espacialmente. Quando uma região é movimentada, todas as suas sub-regiões são deslocadas, mantendo suas posições relativas. A Figura 28 apresenta a visão espacial do Maestro Editor.

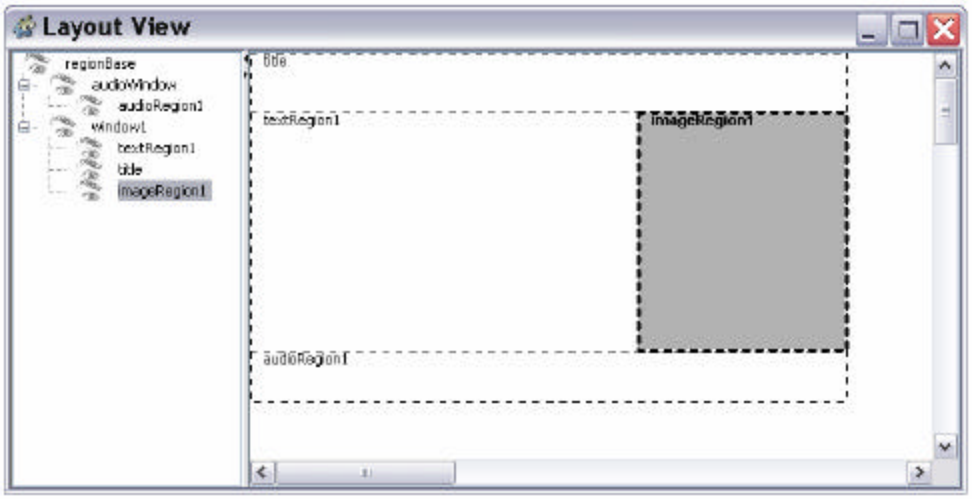

Figura 28 - Visão espacial do Editor Maestro.

Na figura acima, a região à esquerda exibe uma árvore hierárquica com as regiões definidas pelo autor. Ao selecionar um nó dessa árvore, a região correspondente é imediatamente selecionada do lado direito. Outro recurso interessante é que visando despoluir o processo de edição, o autor pode optar por exibir ou não regiões. Para isso, basta clicar sobre o olho correspondente a região de interesse no lado esquerdo da visão espacial. Se o olho referente à região estiver aberto, após o clique com o mouse esse olho se fechará e a região deixará de ser exibida do lado direito. Para reverter o processo basta clicar novamente sobre o olho fechado, e esse se abrirá na árvore à esquerda e a região correspondente voltará a ser exibida do lado direito.

O ambiente de autoria Maestro ainda está aquém de atender a demanda dos mais variados tipos de usuários. Atualmente, somente usuários que conhecem bem a linguagem tiram maior proveito dessa ferramenta. Isso se deve em parte ao fato de oambiente não priorizar o que o usuário quer fazer, mas sim o que a linguagem tem a oferecer.

Apresentações um pouco mais complexas não são triviais de se elaborar. Elas levam à criação de conectores, o que requer do autor um conhecimento mais

profundo da linguagem NCL. Na ferramenta de autoria ainda não é dado um suporte a esse tipo de problema.

Outro problema sério diz respeito as mensagens de erros de compilação, que são pouco descritivas no nível de usuário leigo. Exemplo disso é que não há uma separação de erros de sintaxe (na estrutura do documento) ou de execução (*binds* inválidos, arquivos não encontrados).

4.6. Análise comparativa

Das ferramentas analisadas, pode-se concluir que as ferramentas JAME Author, Cardinal Studio e AltiComposer dão um suporte muito maior ao autor quando o assunto é interatividade. Através de seus modelos de autoria e interfaces gráficas é mais simples pensar tanto em interatividade quanto nas abstrações de controle remoto, botão, etc.

Já quando o assunto de interesse é sincronismo, os ambientes GR*i*NS e Maestro Editor levam uma vantagem enorme frente aos outros ambientes. Isso se deve em grande parte as linguagens declarativas SMIL e NCL, nas quais o GR*i*NS e o Maestro Editor se apóiam, respectivamente.

Uma característica positiva é que todos os ambientes analisados possuem um emulador ou *player* associado à ferramenta de autoria. Isso permite que o autor teste sua aplicação sem a necessidade de um *setop-box*, evitando que problemas sejam percebidos tardiamente.

Quanto ao suporte a usuários de diferentes níveis de conhecimento, a ferramenta JAME Author dá um melhor suporte a usuários não programadores. Tanto o Cardinal Studio quanto o AltiComposer dão suporte a usuários programadores e não-programadores em Java. Já o GR*i*NS e o Maestro Editor podem ser consideradas ferramentas que dão grande suporte a usuários que conhecem bem as suas respectivas linguagens, mas deixam a desejar no suporte a usuários inexperientes.

Um passo inicial para deixar o Maestro Editor mais amigável aos usuários é começar a abstrair do usuário a necessidade de se conhecer profundamente a linguagem NCL. Uma medida para isso é fornecer uma maneira mais intuitiva de se definir os conectores dos elos. Outro ponto que também pode ser contemplado é a incorporação de *templates* de documentos a essa ferramenta. Além disso, seria

interessante a existência um repositório com algumas composições com semântica pré-definidas que são comumente usadas na criação de programas para TV digital interativa. Tudo isso de fácil acesso através do mouse.

A seguir, a Tabela 3 exibe um resumo de algumas características dos ambientes de autoria analisados.

Tabela 3 - Análise comparativa entre ferramentas de autoria.

. **Ferramenta**	**Paradigma/ Modelo**	**Emulador integrado**	**Foco**	**Suporte a usuários leigos**	**Suporte a usuários experientes**
JAME Author	Procedural (DVB-J) /Page-based	SIM	Interatividade	ALTO	BAIXO
Cardinal Studio	Procedural (DVB-J) / Baseado em Atos, camadas e componentes	SIM	Interatividade	ALTO	ALTO
Alti Composer	Procedural (DVB-J) / Baseado em cenas, planos, *shots* e atores	SIM	Interatividade	ALTO	ALTO
GR*i*NS	Declarativo (SMIL) / Modelo baseado no modelo CMIF	SIM	Sincronização	MÉDIO	ALTO
Maestro Editor	Declarativo (NCL)/ NCM	SIM	Sincronização	MÉDIO	ALTO

5 Conclusões

Neste trabalho foram apresentadas as linguagens utilizadas nos principais padrões para TV digital interativa. Nesses padrões, tanto as linguagens baseadas em HTML, quanto as linguagens baseadas em Java são, no geral, similares, em relação a forma de implementação dos eventos de sincronismo e interatividade. Ambos os tipos de linguagens exigem que o autor especifique, em detalhes, os passos que a aplicação deve executar para atingir seu objetivo.

O uso da linguagem Java exige que o autor conheça técnicas de programação orientadas a objetos e, também, o conjunto de pacotes oferecidos pelo *middleware*. Entre esses pacotes, destaca-se a implementação JavaTV. Para desenvolver aplicações nos padrões que adotam esse pacote, geralmente, o autor deve conhecer em detalhes o ciclo de vida das aplicações e como controlar esse ciclo, de acordo com a aplicação que ele deseja construir.

O uso da linguagem HTML, a princípio, poderia sugerir uma maior facilidade para a autoria, em função da simplicidade e da difusão do HTML como linguagem para formatação de documentos na WWW. No entanto, na maioria das vezes, para a especificação de eventos de interatividade e sincronismo utilizando HTML, faz-se necessário o uso de outra linguagem, baseada em scripts. Na linguagem ECMAScript são encontradas características do paradigma orientado a objetos, que tornam a implementação tão complexa quanto através do uso de outras linguagens desse paradigma, como Java.

Os padrões de TV digital oferecem, geralmente, multiplas possibilidades de autoria, usualmente denominadas de procedural, para linguagens baseadas em Java, e declarativas, para linguagens baseadas em HTML. No caso dos padrões DVB e ATSC, pode-se afirmar que as funcionalidades de ambas as classes de linguagens são similares, pois é possível utilizar as API's do *middleware* projetadas para o DVB-J e ACAP-J através de comandos ECMAScript.

Outro ponto de similaridade entre as linguagens dos padrões de TV digital é a forma como os *stream events* são tratados. Em ambas as classes de linguagens (procedural e declarativa), normalmente, os *stream events* são tratados através de

observadores (*listener*) associados a objetos e a um método responsável por definir a semântica do tipo de *stream event* recebido no contexto da aplicação. Da mesma forma que os *stream events*, os eventos de interatividade, disparados pela interação do usuário com o controle remoto possuem representações similares nas linguagens procedurais e declarativas.

Ao contrário das linguagens baseadas em Java e em HTML+Script, as linguagens declarativas são projetadas sobre modelos hipermídia que tem como uma de suas principais funções representar os relacionamentos entre objetos de mídia em uma apresentação. Nessas linguagens, as relações entre eventos em aplicações para TV digital podem ser especificadas sem que seja necessário definir estruturas complexas, como estruturas de controle, de repetição, variáveis, observadores etc., favorecendo a autoria.

Entre as linguagens declarativas, podem ser destacadas as linguagens SMIL e NCL. Para a linguagem SMIL existem propostas de arquiteturas que utilizam essa linguagem em conjunto com linguagens baseadas em Java, através de um interpretador implementado em Xlet. No entanto, uma das principais restrições apresentadas foi a falta de um perfil da linguagem SMIL que trate os *stream events*.

A linguagem NCL, por outro lado, possui uma proposta de *middleware* cuja implementação é voltada para a linguagem, que corresponde a proposta do sistema brasileiro de TV digital. No entanto, no momento atual, não possui uma arquitetura definida para a implementação através de interpretadores Xlets. Uma das principais vantagens da NCL, além das vantagens inerentes a sua natureza declarativa, é o tratamento dispensado aos *stream events*. Nessa linguagem, os *stream events* podem alterar o documento em execução, preservando as especificações no ambiente de execução. Além disso, a manipulação dos *stream events* é totalmente transparente para os autores.

Considerando-se a complexidade das linguagens baseadas em Java e em HTML+Script para a especificação de aplicações para TV digital, o uso de ferramentas para autoria gráfica torna-se ainda mais importante. Como a linguagem Java e as linguagens de Script possuem várias características de linguagens de propósito geral, as ferramentas de autoria para essas linguagens devem suportar os recursos oferecidos pelas linguages e, simultanemente, criar abstrações para o desenvolvimento de aplicações voltadas para TV digital. Nesse

aspecto nota-se uma tendência em definir estruturas auxiliares para a criação, como cenas, notas, shots e atores, por exemplo, definidas pelo Cardinal Studio. É importante ressaltar que essas estruturas auxiliares não são necessárias nas linguagens dos padrões para TV digital, elas são apenas estruturas para auxiliar o desenvolvimento mental do autor na criação, abstraindo as estruturas originais das linguagens.

Outras ferramentas de autoria, voltadas para linguagens declarativas, são caracterizadas pelas múltiplas visões. Nessas ferramentas, as abstrações são definidas nos diversos tipos de visões, que permitem simular um tipo específico de edição (temporal, leiaute, estrutura etc.).

6 Referências Bibliográficas

[ARIB04a] ARIB. **ARIB STD-B23: Application Execution Engine Platform for Digital Broadcasting**. ARIB Standard, fevereiro, 2004.

[ATSC03a] ATSC. **A/100: DTV Application Software Environment - Level 1 (DASE-1)**. ATSC Standard, março, 2003. Disponível em *http://www.atsc.org/standards/a_100.zip*. Acesso em 10/10/2004.

[ATSC04c] ATSC. **CS/101A: Advanced Common Application Platform (ACAP)**. ATSC Candidate Standard, fevereiro, 2004. Disponível em *http://www.atsc.org/standards/cs_documents/cs_101a.pdf*. Acesso em 10/10/2004.

[BuRu04] BULTERMAN, D.; RUTLEDGE, L. **SMIL 2.0: Interactive Multimedia for Web and Mobile Devices**. Springer, Abril de 2004.

[CABL02] CableLabs. **OpenCable Application Platform Specification**. Issued Specification, abril, 2002. Disponível em *http://www.opencable.com/downloads/specs/OC-SP-OCAP2.0-I01-020419.pdf*. Acesso em 10/11/2004.

[CARDINAL] Cardinal Information Systems Ltd, **Cardinal Studio 4.0 User's Guide** – Finland - www.cardinal.fi

[Coelho04] Coelho, R. M. **Integração de Ferramentas Gráficas e Declarativas na Autoria de Arquiteturas Modeladas através de Grafos Compostos** - Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio – 2004

[ECMA262] ECMA Standardizing Information and Communication Systems. **ECMAScript Language Specification**, Standard ECMA 262, 3rd Edition, Dezembro de 1999. Disponível em http://www.ecma-international.org/publications/standards/Ecma-262.htm. Acesso em 06 nov. 04.

[ETSI03a] ETSI. ES 201 812: **Multimedia Home Platform (MHP) Specification 1.0.3**. ETSI Standard, dezembro, 2003. Disponível em http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=18800. Acesso em 10/10/2004.

[ETSI04a] ETSI. **TS 102 819: Globally Executable MHP (GEM)**. ETSI Standard, maio, 2004. Disponível em *http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=19737*. Acesso em 10/10/2004.

[FLEX04] FLEXTV. **TV Digital: Análise das Alternativas Tecnológicas**. Relatório em atendimento ao Requisito 4.1.3 – RFP04/2004 – MCT/FINEP, produto 1A, dezembro, 2004.

[ICECRE] ICE-CREAM Consortium – Interactive Consumption of Entertainment in Consumer Responsive, Engaging & Active Media – http://www.hitech-projects.com/euprojects/icecream.

[LCHV03] LAMDON, J.L; CESAR, P.; HERRERO, C.; VUORIMAA, P. **Usages of a SMIL player in digital television**, Proceedings of the VII International Conference on Internet and Multimedia Systems and Applications – IASTED, USA, Agosto de 2003.

[MCRS05] MORENO, M.F.; COSTA, R.M.R.; RODRIGUES, R.F.; SOARES, L.F.G. **Edição de Documentos Hipermídia em Tempo de Exibição**, X Simpósio Brasileiro de Sistemas Multimídia e Hipermídia, Poços de Caldas, Brasil, dezembro de 2005.

[MuSS03] MUCHALUAT-SAADE, D.C.; SILVA, H.V.O; SOARES, L.F.G. **Linguagem NCL versão 2.0 para Autoria Declarativa de Documentos Hipermídia**, IX Simpósio Brasileiro de Sistemas Multimídia e WEB - WebMídia 2003, Salvador, Brasil, Novembro de 2003.

[PeVu01] PENG, C.; VUORIMAA, P. **Digital Television Application Manager**, Proceedings of the IEEE International Conference on Multimedia and Expo, 2001, Japão, Agosto de 2001.

[PeCV01] PENG, C.; CESAR, P.; VUORIMAA, P. **Integration of Applications into Digital Television Environment**, Proceedings of the VII International Conference on Distributed Multimedia Systems, Taiwan, Setembro de 2001.

[PAAÇ02] PEMBERTON, S.; AUSTIN, D.; AXELSSON, J.; et al. **XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)**, 2002. Disponível em http://www.microsoft.com/windows/ie/. Acesso em 16 nov. 04. Disponível em http://www.w3.org/TR/xhtml1/. Acesso em 16 nov. 04.

[PICV02] PIHKALA, K.; CESAR, P.; VUORIMAA, P. **A Cross-Platform SMIL Player**, Procedings of the International Conference on Communication, Internet and Information - IASTED, USA, Novembro de 2002.

[QSJAME] Fraunhofer Institute for Media Communication IMK, **Quick Start - JAME Author 1.0** – Schloss Birlinghoven – Germany – www.jame.tv

[RMGRINS] GRiNS Pro Editor for SMIL 2.0 - **Reference Manual** - http://oratrix.oratrix.com/Download/ReferenceManual.pdf

[SGALTI] Alticast Inc, **Script Guide – AltiComposer 2.0** – USA – www.alticast.com

[SoCoRo04] Soares, L. F. S., Colcher, S., Rodrigues, R. **Relatório TV Digital - Identificação dos Cenários Tecnológicos de Interesse** - Relatório apresentado como parte dos requisitos de avaliação da disciplina "Seminários de Sistemas Multimídia", 2º semestre de 2004.

[SoRM03] SOARES, L.F.G., RODRIGUES, R.F., MUCHALUAT-SAADE, D.C. **Modelo de Contextos Aninhados – versão 3.0**, Relatório Técnico, Laboratório TeleMídia, Departamento de Informática, PUC-Rio, 2003.

[UGALTI] Alticast Inc, **User Guide – AltiComposer 2.0** – USA – www.alticast.com

[UMJAME] Fraunhofer Institute for Media Communication IMK, **User Manual - JAME Author 1.0** – Schloss Birlinghoven – Germany – www.jame.tv

[W3C01b] W3C - World-Wide Web Consortium. **Synchronized Multimedia Integration Language (SMIL 2.0) Specification**, W3C Recommendation, Agosto de 2001. Disponível em http://www.w3.org/TR/smil20/. Acesso em 28 out.. 04.

[W3C04a] W3C - World-Wide Web Consortium. **Extensible Markup Language (XML) 1.1**, fevereiro de 2004. Disponível em http://www.w3.org/TR/2004/REC-xml11-20040204/. Acesso em 05 nov. 04. [W3C04b] W3C - World-Wide Web Consortium. Cascading Style Sheets Home Page, 2004. Disponível em http://www.w3.org/Style/CSS/. Acesso em 08 nov. 04.