2
Trabalhos Relacionados

Ferramentas de autoria podem ser empregadas a fim de abstrair do autor toda, ou pelo menos parte da complexidade de se utilizar uma linguagem de programação na criação de aplicações voltadas para TV digital. Através dessas ferramentas, o autor pode se concentrar, principalmente, no processo de criação.

Para dar suporte a usuários de todos os níveis de *expertise*, as ferramentas de autoria devem oferecer os recursos existentes nas linguagens e, simultaneamente, criar abstrações para facilitar e agilizar o desenvolvimento das aplicações. Nesse aspecto, nota-se uma tendência em definir estruturas auxiliares. É importante ressaltar que essas estruturas auxiliares não são necessárias nas linguagens dos padrões para TV digital, sendo apenas estruturas de apoio ao desenvolvimento. No caso das ferramentas de autoria para linguagens baseadas em modelos hipermídia, geralmente, as abstrações são definidas nos diversos tipos de visões, que permitem simular um tipo específico de edição (temporal, leiaute, estrutural etc.).

Existem diversas ferramentas de autoria para TV digital, e inúmeras outras para Web (*World-Wide Web* ou *WWW*) (Berners-Lee et al., 1994) e linguagens declarativas. Este capítulo faz uma análise de algumas dessas ferramentas, levantando suas principais características. Como complemento, o Apêndice A apresenta informações sobre as linguagens subjacentes às ferramentas de autoria apresentadas neste capítulo.

2.1.
JAME Author

O JAME Author (Fraunhofer, 2007a) é um ambiente de autoria para a criação de aplicativos voltados para TV digital interativa no *middleware* MHP/OCAP (*OpenCable Application Platform*) (ETSI, 2003; CableLabs, 2006), e faz parte de um grupo de soluções para iTV (*interactive TV*) do Fraunhofer Institute for Media Communication IMK.

Essa ferramenta 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 é possível se navegar de maneira muito semelhante à navegação na Web. Essas páginas são descritas através de uma linguagem chamada JAME PDL (*JAME Page Description Language*) (Fraunhofer, 2007c), a qual é baseada em XML.

O JAME Author lida com dois tipos básicos de página. No primeiro tipo, o funcionamento é similar ao das páginas Web, onde a ativação de um *link* (elo) carrega uma nova página, e esta substitui a página corrente. No segundo, 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 tratamento de erro. Durante a elaboração de um projeto (aplicativo) outras páginas podem ser criadas a fim de caracterizar o tipo de aplicação que está sendo desenvolvida. A Figura 1 apresenta a interface gráfica do JAME Author.

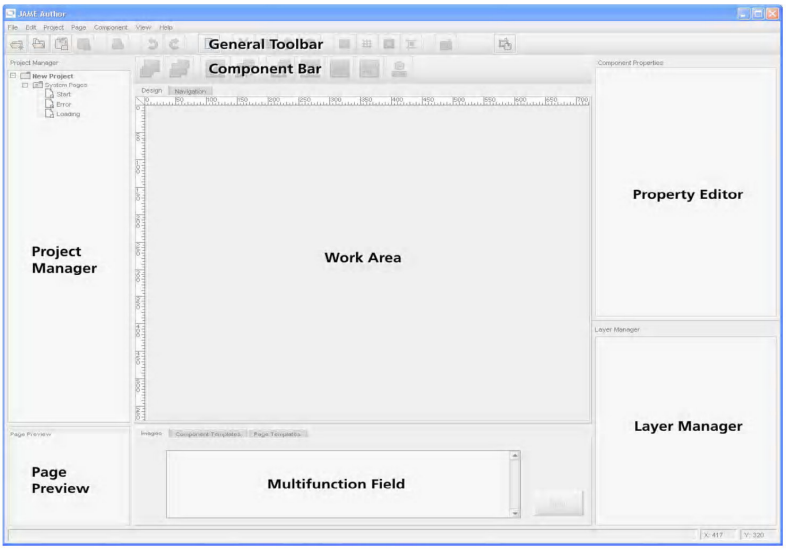

Figura 1 - Interface gráfica do JAME Author (Fraunhofer, 2007c)

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 selecionada no *Project Manager*. Quando um componente de uma página aberta na *Work Area* é selecionado, suas propriedades são exibidas na região *Property Editor*. Além de serem exibidas, essas propriedades também podem ser editadas. Já 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.

O processo de edição se dá quando uma página listada no *Project Manager* é aberta na *Work Area*. Existem dois modos de edição: 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 padrão de componentes já vem especificado na ferramenta de autoria, mas outros componentes 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 no projeto em tempo de criação, quanto como uma região informativa onde mensagens de validação são apresentadas ao usuário.

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

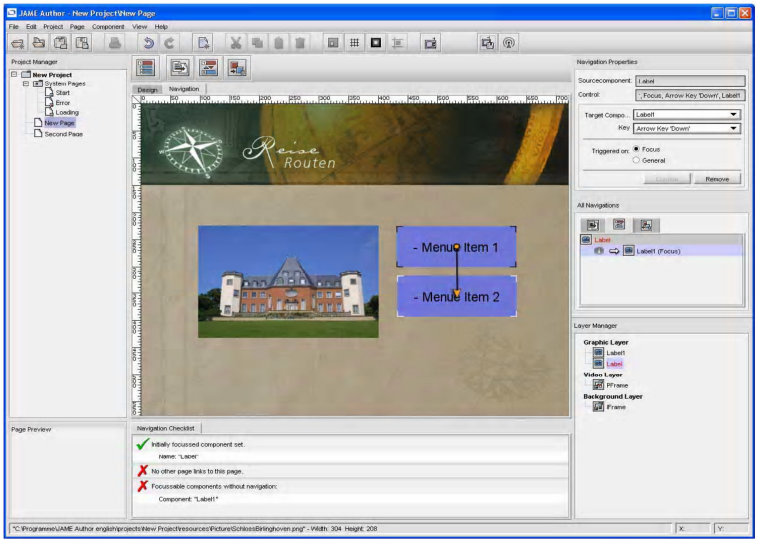

Figura 2 - Transferência de foco no JAME Author (Fraunhofer, 2007b)

Esse ambiente de autoria possui um emulador (simulador) MHP integrado. Isso permite que o autor execute e valide seu trabalho à medida que esse é desenvolvido.

Pode ser observado que a criação de documentos hipermídia nessa ferramenta é similar à criação de documentos HTML, o que vem a facilitar bastante a sua utilização por usuários não programadores em Java. Entretanto, o JAME Author não oferece suporte nem à especificação de sincronismo espacial/temporal entre objetos de mídia, nem a qualquer outra computação em Java, além das pré-definidas. Basicamente, a interatividade se resume à definição de relacionamentos de referência entre documentos e à mudança de foco entre componentes. Ademais, essa ferramenta de autoria não fornece mecanismos para que seja feita a edição ao vivo de documentos, nem a edição temporal.

2.2. Cardinal Studio

O Cardinal Studio (Cardinal, 2007) é uma ferramenta de autoria da Cardinal Systems 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*). 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 os componentes de um conjunto possuem um componente equivalente no outro. A diferença é que componentes focáveis podem ser usados na navegação com o controle remoto, enquanto os componentes de *background* não.

Componentes invisíveis, como o próprio nome sugere, não são desenhados na tela. Esses componentes usualmente são utilizados como estruturas auxiliares que ajudam a aplicação (documento) a funcionar. Alguns exemplos de componentes invisíveis são o canal de retorno, uma conexão HTTP (*Hypertext Transfer Protocol*) (W3C, 1999b), um s*tream event* (detalhes no Apêndice A), um botão do controle remoto, entre outros.

Dentro dos atos, os componentes são estruturados em camadas, sendo que uma camada pode ser compartilhada entre atos. A camada invisível é definida como um contêiner de componentes invisíveis. Já a camada gráfica é definida como o local onde ficam os componentes visíveis. A camada de vídeo é definida como o local para exibição do vídeo especificado 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. 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.

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

A Figura 3 apresenta a interface gráfica do Cardinal Studio. Os números dentro dos círculos, todos em vermelho, não fazem parte da interface gráfica, servindo apenas para destacar uma região ou elemento de interesse. Essa notação será utilizada ao longo de toda a dissertação.

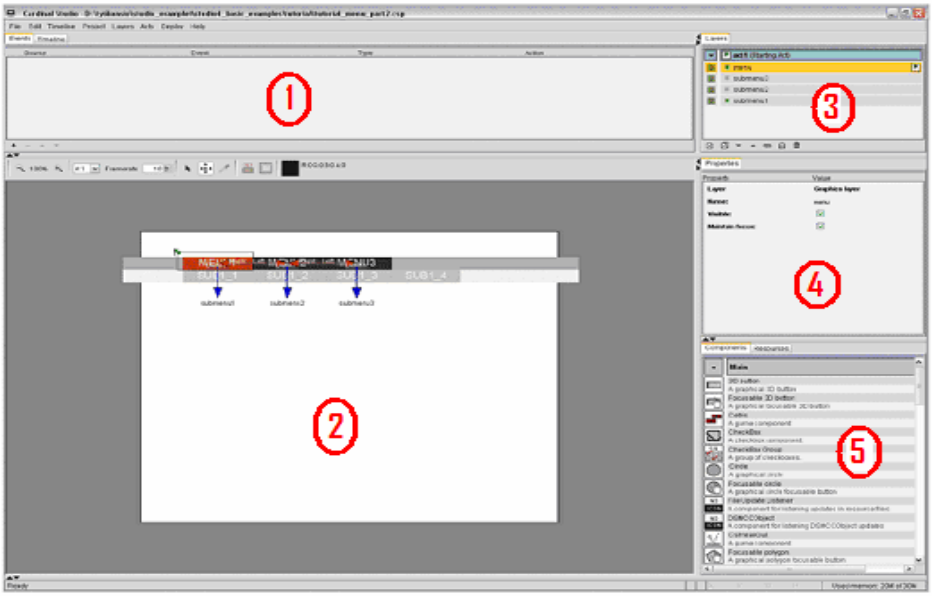

Figura 3 - Interface gráfica do Cardinal Studio

Na figura acima, a região 1 (um) é formada pelo editor de eventos e pelo editor de *timeline* (linha do tempo), os quais são usados para monitorar eventos e propriedades de componentes baseados no tempo, respectivamente. Cabe destacar que o editor de *timeline* é muito simples, não dando uma noção clara de como a apresentação decorrerá ao longo do tempo, muito menos permitindo a simulação de eventos interativos.

A região 2 (dois) da Figura 3, chamada de *Canvas*, é utilizada para definir como os componentes estarão espacialmente dispostos na tela. Já a região 3 (três) exibe a lista de camadas de um ato. Nela é possível adicionar e remover atos e camadas. A região 4 (quatro) é o local onde são exibidas as propriedades de um componente selecionado (tanto componentes visíveis quanto invisíveis possuem propriedades). Por fim, a região 5 (cinco) é um repositório de componentes que podem ser adicionados ao documento durante o processo de criação.

Esse ambiente de autoria configura automaticamente a navegação entre componentes focáveis de acordo com a ordem que esses componentes são criados. Para modificar essa ordem, a ferramenta oferece o modo de navegação. Esse modo exibe setas de um componente para outro, mostrando como será realizada a transferência de foco através do controle remoto. Se desejar, o autor tem a liberdade para definir qual será a seqüência a ser utilizada no seu documento.

O Cardinal Studio dá suporte tanto a usuários avançados quanto a iniciantes. 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, usuários avançados podem desenvolver componentes e integrá-los à ferramenta a fim de estender suas funcionalidades.

Esse ambiente foca principalmente na especificação de eventos interativos. Contudo, a definição de relacionamentos de sincronismo espacial/temporal entre objetos de mídia é suportada através da personalização dos eventos dos componentes, e isso demanda codificação.

O Cardinal Studio não fornece recursos diretos para a edição ao vivo de documentos, mas permite a especificação de *stream events*, que podem vir a ser usados na edição ao vivo (Apêndice A). Por fim, é importante destacar que esse ambiente de autoria possui um emulador para testar e validar as funcionalidades dos aplicativos desenvolvidos.

2.3. AltiComposer

O AltiComposer (Alticast, 2007a) é um ambiente de autoria para criação de aplicativos em conformidade com o DVB-MHP (*Digital Video Broadcast*) (Soares et al., 2004). 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. Usuários mais avançados, por sua vez, podem criar componentes personalizados, estendendo a API (*Application Programming Interface*) *Component Development Kit* (CDK) da ferramenta.

Visando tornar o processo de autoria mais intuitivo e atrativo para os profissionais de TV digital, o ambiente AltiComposer utiliza um modelo definido em concordância com a industria de TV e cinema. Os componentes existentes e suas relações de dependência são apresentados na Figura 4.

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

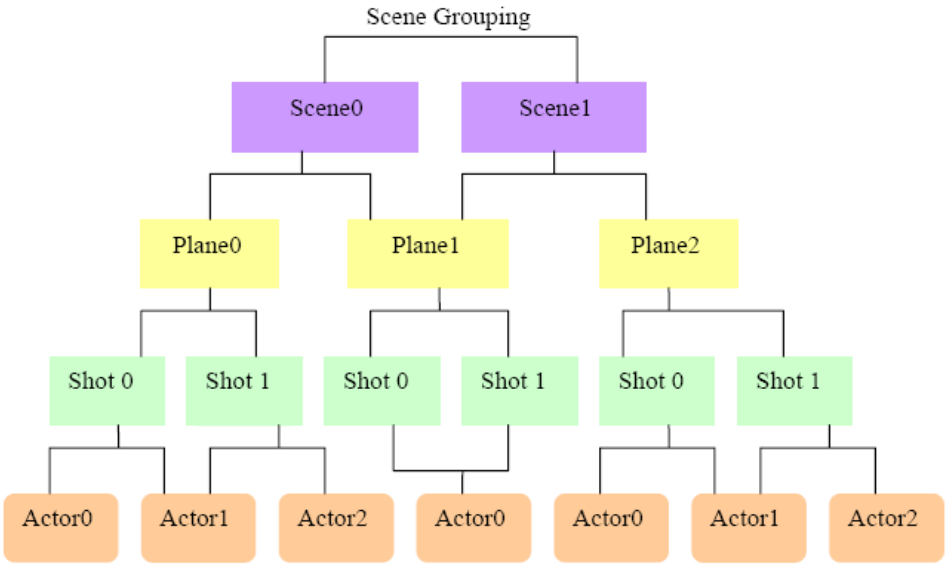

Figura 4 - Arquitetura de componentes do AltiComposer (Alticast, 2007c)

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 telespectador, 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 entendê-lo como sendo um contêiner de atores. Em um dado instante de tempo, um *shot* é visto como um *snapshot* (uma foto) da aplicação.

O plano é a unidade básica para salvar e carregar conteúdo 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. O comportamento é um script que diz como cada componente deve responder a um determinado evento. A Figura 5 exibe o modelo de autoria do AltiComposer.

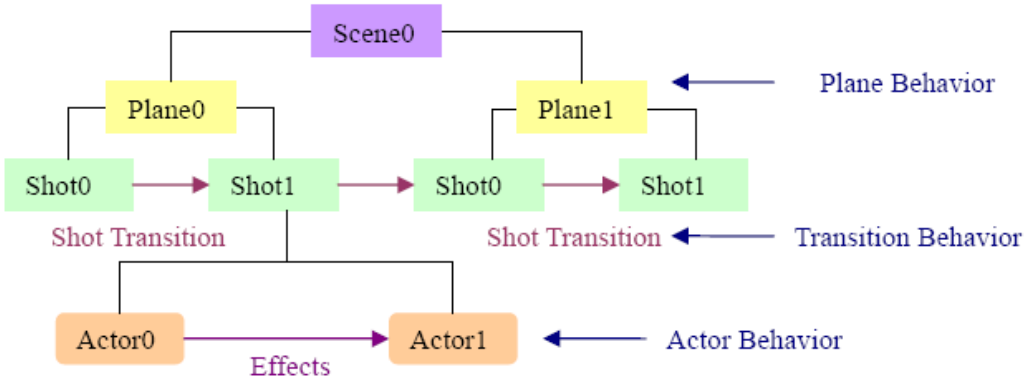

Figura 5 - Modelo de autoria do AltiComposer (Alticast, 2007c)

Um efeito (*effect*) é um objeto Java que leva um ator de um estado para outro. Basicamente, 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.

Efeitos e transições podem ser vistos como recursos, ou *behaviors*, que estão prontos para serem usados. Não obstante, *behaviors* podem ser editados (codificados) em tempo de criação para realizar uma operação desejada. Essa codificação é facilitada pelo uso de uma janela (*Script Editor*) que exibe através de sua interface gráfica as opções que podem ser utilizadas, permitindo a geração de código em mais alto nível. A linguagem de programação utilizada é a *AltiComposer Script Language* (Alticast, 2007b), um subconjunto da ECMA-262 (ECMA, 1999). Assim, usuários avançados podem programar de forma mais específica, tirando o máximo de proveito do modelo da ferramenta.

A Figura 6 exibe a interface gráfica do ambiente AltiComposer. A região 1 (um) é conhecida como *Project Window* (Janela de Projeto). Do seu lado esquerdo são listadas hierarquicamente as cenas e planos. Na parte inferior direita são listados os *shots* de um plano, e se um deles for selecionado, seus atores são mostrados na região acima (superior à direita da região 1). A região 2 (dois) representa a *Library Window*, um repositório de recursos que podem ser utilizados durante a elaboração de um aplicativo.

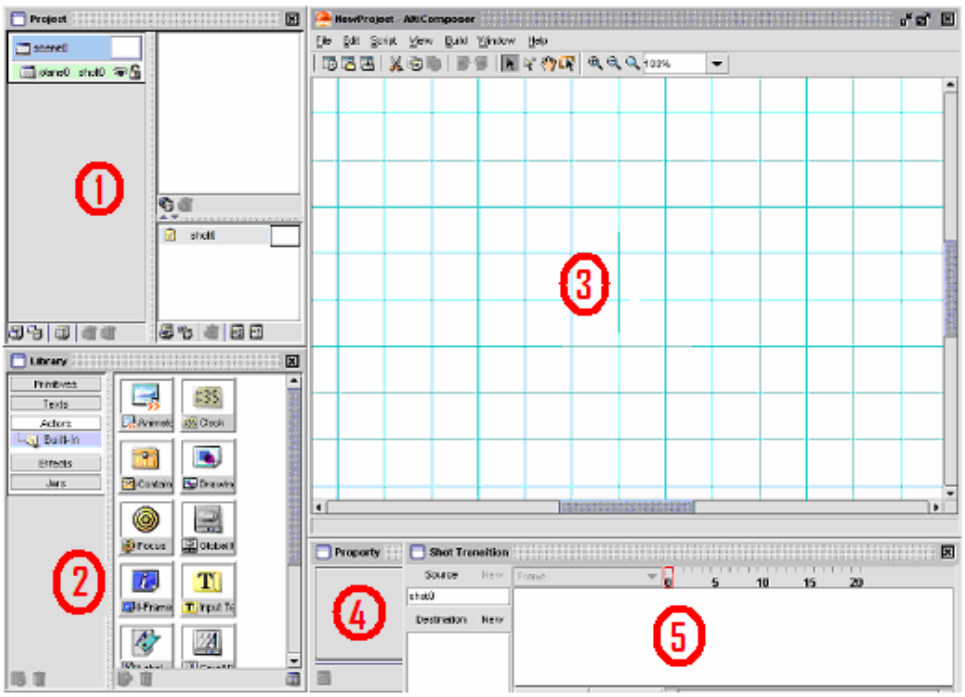

Figura 6 - Interface gráfica do AltiComposer

Ainda com relação à Figura 6, a região 3 (três), conhecida como *Stage*, serve como área de trabalho para criação do aplicativo. A região 4 (quatro), por sua vez, é a janela de propriedades, onde um elemento selecionado na área de trabalho pode ter suas propriedades editadas. Por fim, a região 5 (cinco) representa a janela de transição, na qual pode ser definida e visualizada a transição de *shots*. O AltiComposer não fornece uma edição na linha do tempo, na qual seja possível especificar e verificar a disposição de objetos de mídia no tempo.

Nessa ferramenta, a sincronização temporal e espacial são contempladas através da codificação do *behavior* de um determinado componente. Já a interatividade é definida através da utilização de componentes que representam os botões do controle remoto, e da especificação de seus *behaviors*.

Assim como no Cardinal Studio, no AltiComposer não são oferecidos mecanismos para a edição ao vivo de documentos, mas esse ambiente prevê a especificação de *stream events*, os quais podem vir a ser utilizados para edição ao vivo (Apêndice A). Vale ainda ressaltar que esse ambiente de autoria possui um emulador MHP que permite que o autor depure seu trabalho à medida que esse é desenvolvido.

2.4. GR*i*NS

Uma forma de eximir o autor da necessidade de conhecer profundamente a linguagem SMIL é a utilização de uma ferramenta de autoria durante o processo de criação. O GR*i*NS (Bulterman et al., 1998) é um ambiente de autoria para SMIL que possui várias visões integradas (temporal, espacial e textual), se destacando por sua visão temporal para concepção de documentos. A abordagem de visões será discutida no próximo capítulo, na apresentação da arquitetura do *Composer*. Neste ponto, pode-se considerar uma visão como uma abstração que permite visualizar um programa hipermídia sob uma determinada perpectiva, como por exemplo, a linha do tempo.

Na visão temporal é possível montar e manipular a apresentação de um documento multimídia 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. Todas as composições com semântica temporal do SMIL são exibidas em cores diferentes na visão temporal, e isso permite ao autor identificar facilmente de qual tipo se trata. A Figura 7 apresenta a visão temporal do GR*i*NS.

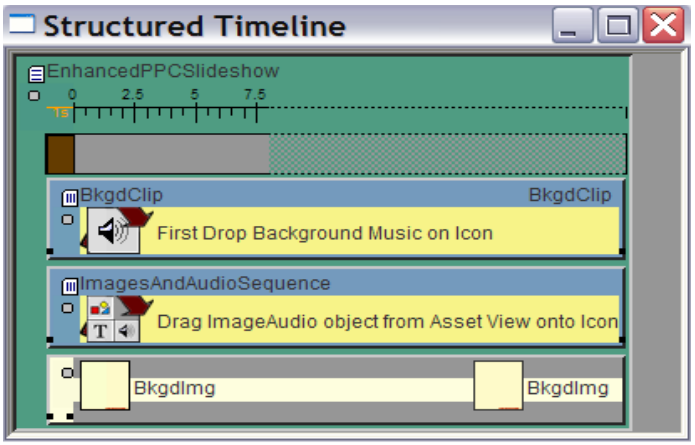

Figura 7 - Visão temporal do GR*i*NS

A visão espacial tem por objetivo fornecer ao autor um modo prático e ágil de criar e manipular as regiões nas quais os objetos de mídia serão apresentados no espaço. Por permitir apenas a especificação inicial das regiões, essa visão

poderia ser chamada de visão de leiaute[2]. A Figura 8 apresenta a visão espacial do GR*i*NS.

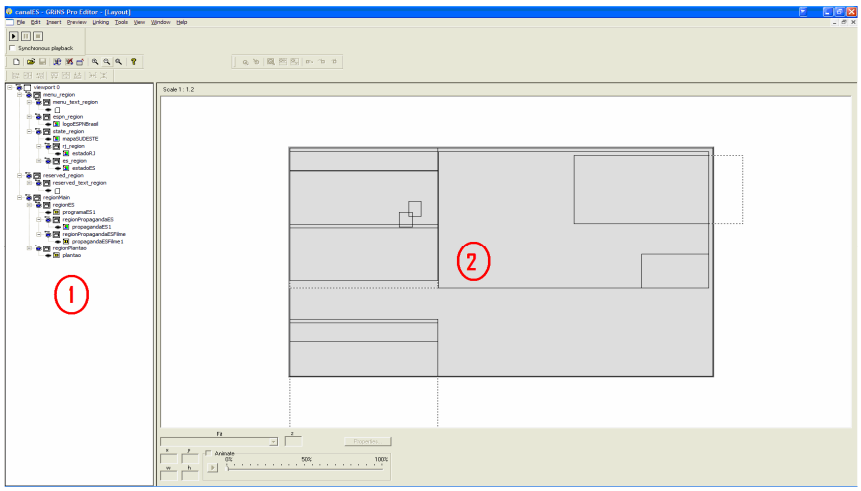

Figura 8 - Visão espacial do GR*i*NS

Na figura acima, a região 1 (um) exibe uma árvore que lista hierarquicamente as regiões definidas, e os objetos de mídia que serão exibidos em cada uma delas. Já a região 2 (dois) 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 à região. Nessa última propriedade, os possíveis valores são *meet* e *fill*. No primeiro caso, a mídia mantém sua resolução (tamanho), e se ela for maior que a região somente parte da mídia aparece. Com o valor "*fill*", a mídia se ajusta à área da região.

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

O editor textual é bem simples, e oferece somente recursos básicos. A Figura 9 exibe o código fonte de um documento nessa visão. Para que as modificações feitas possam ser aplicadas nas outras visões, é necessário que o usuário clique no botão *Apply*. Caso haja algum erro, o usuário é prontamente informado, e se optar por desfazer as alterações, deve clicar no botão *Revert*.

[2] Geralmente uma visão espacial permite ao autor visualizar graficamente como os objetos de mídia estarão dispostos no espaço em um dado instante da apresentação do documento.

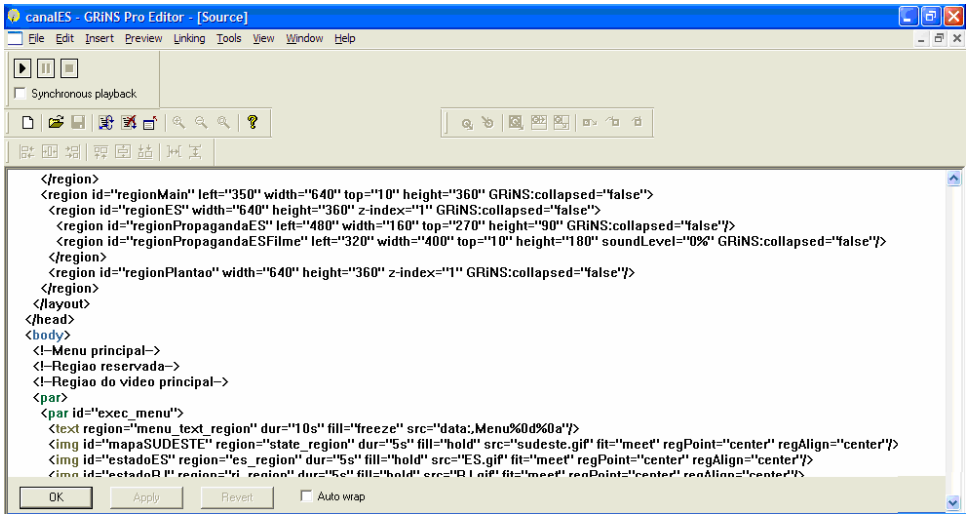

Figura 9 - Visão textual do GRiNS

Um ponto fraco da visão textual é que somente as composições com semântica temporal (*seq*, *par* e *excl*) recebem cores de destaque, e isso pode fazer 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 usuário lançar mão das suas características da forma que mais lhe convém, e se usadas complementarmente, pode ser explorado o que de melhor existe em cada uma delas.

A existência de um *player* (exibidor) 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 retrabalho.

Vale destacar que o GRiNS privilegia fortemente a 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 GRiNS, a especificação de relacionamentos interativos é mais fácil na visão textual. Dado que a interatividade é um requisito importante a ser atendido em sistemas de TV digital interativa, uma alternativa seria incorporar uma nova visão que permitisse definir relacionamentos interativos de forma mais intuitiva.

É importante ressaltar ainda que a linguagem SMIL, e conseqüentemente o GRiNS, não fornece suporte à edição ao vivo (não foi previsto o seu uso para TV

digital). Além disso, sua visão temporal não permite a simulação de eventos interativos em tempo de criação.

2.5. LimSee2

O LimSee2 (WAM, 2007) é uma ferramenta de autoria para documentos SMIL desenvolvida pelo grupo de pesquisa WAM (*Web Adaptation and Multimedia*) do Institut National de Recherche en Informatique (INRIA).

Assim como o GR*i*NS, o LimSee2 é um ambiente de autoria com múltiplas visões integradas, de modo que uma modificação em uma visão é refletida nas demais. A Figura 10 apresenta a interface gráfica do ambiente LimSee2.

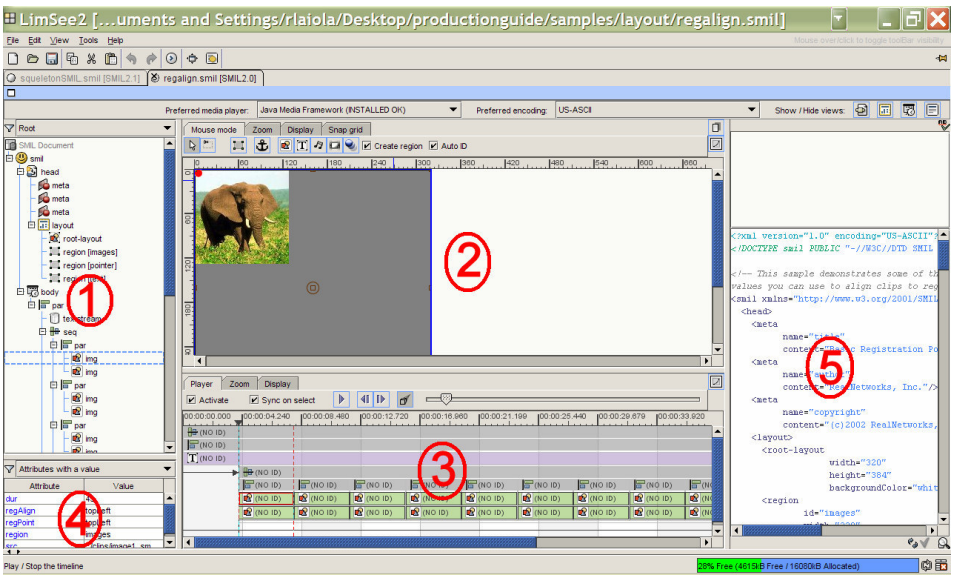

Figura 10 - Interface gráfica do LimSee2

Na figura acima, a região 1 (um) representa a visão estrutural, na qual o documento SMIL é organizado na forma de uma árvore. A região 2 (dois) é a visão espacial, utilizada para criação e manipulação de regiões. Nela, os objetos de mídia também podem ser pré-visualizados em um dado instante de tempo (somente imagens e textos).

Ainda com relação à Figura 10, a região 3 (três) representa a visão temporal. Nessa visão os elementos são representados por retângulos cujo comprimento está relacionado à sua duração. Esses objetos temporais podem ser movidos e redimensionados na linha do tempo. Além disso, nessa visão é possível especificar um determinado instante de tempo para visualizar na visão espacial o estado da apresentação. Esse recurso é de grande valia, pois permite verificar

como está ficando a apresentação durante a criação. Cabe ressaltar que na versão do LimSee2 analisada, a visão espacial não estava funcionando completamente.

A região 4 (quatro) da Figura 10, chamada de visão de atributos, exibe e permite a edição das propriedades de um elemento selecionado na visão temporal, estrutural ou espacial. Por fim, a região 5 (cinco) representa a visão textual, na qual o código fonte SMIL pode ser visualizado e alterado.

Esse ambiente de autoria não possui um exibidor integrado, mas reconhece os exibidores nativos instalados no sistema, e permite que o autor escolha qual utilizar.

Da mesma forma que o GR*i*NS, o LimSee2 privilegia a especificação da sincronização temporal à interatividade, sendo essa última mais fácil de definir através da visão textual. É importante destacar que pela linguagem SMIL não ser voltada para TV digital, o LimSee2 não fornece nenhum suporte à edição ao vivo de documentos SMIL. Além disso, a visão temporal dessa ferramenta não permite que novos objetos de mídia sejam inseridos no tempo. Para isso, é necessário utilizar ou a visão estrutural ou a visão espacial. Outra carência fica por conta da falta de suporte à simulação de eventos interativos.

2.6. Macromedia Flash

O Macromedia Flash (Adobe, 2007b), atualmente chamado de Adobe Flash, ou simplesmente Flash, é uma ferramenta de autoria gráfica vetorial utilizada para criação de conteúdo para Web, jogos, filmes, entre outros. Estima-se que a tecnologia Flash atinja mais de 98% dos usuários da Internet[3].

Uma das principais características desse ambiente é que, além de oferecer uma interface gráfica bastante intuitiva para criação de animações e aplicações interativas, também permite ao desenvolvedor lançar mão de recursos mais interessantes através da linguagem de programação ActionScript (Williams, 2002). Essa linguagem, que é uma variação da ECMAScript (ECMA, 1999), é usada, por exemplo, para manipulação de dados e para especificação da interatividade em aplicações Flash.

[3] Fonte: http://www.adobe.com/products/player_census/flashplayer/. Acesso em 23 de abril de 2007.

A ferramenta de autoria Macromedia Flash é composta pela IDE (*Integrated Development Environment*) e por um exibidor. Esse exibidor (*player*) é usado para depurar e apresentar as aplicações criadas, e possui, dentre outras coisas, uma máquina virtual para processar scripts, a *ActionScript Virtual Machine* (AVM). A Figura 11 apresenta a interface gráfica do Macromedia Flash.

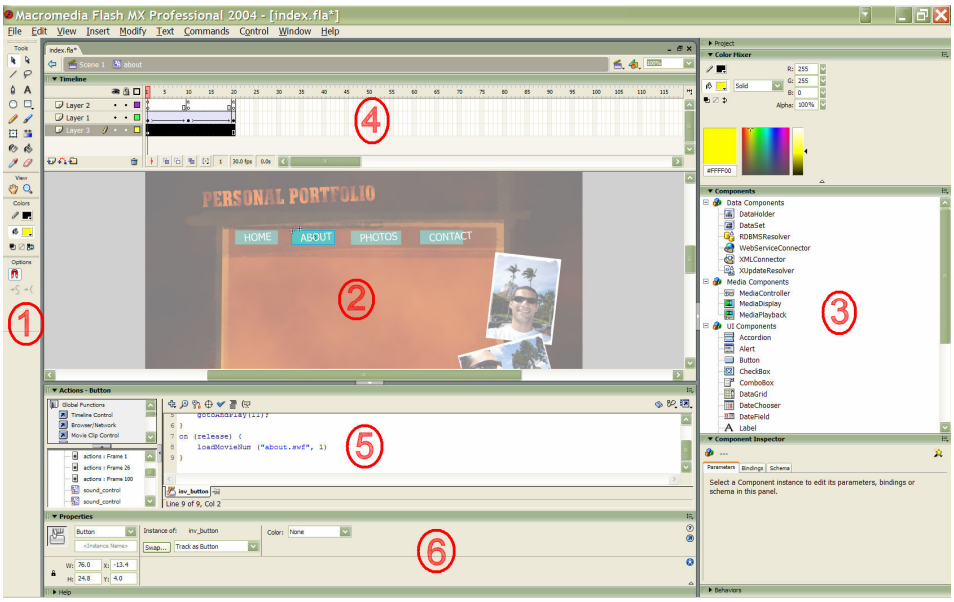

Figura 11 - Interface gráfica do Macromedia Flash

Na figura acima, a região 1 (um) representa a barra de ferramentas. Nela estão compreendidas funcionalidades básicas que são bastante utilizadas, como por exemplo, recursos de desenho e zoom. A região 2 (dois), conhecida como *Stage*, é utilizada para definir como os elementos estarão dispostos visualmente em um dado instante (visão espacial). É nessa região que os recursos da barra de ferramentas são aplicados.

Já a região 3 (três) da Figura 11 destaca componentes que podem ser usados para criação de uma aplicação. São definidos, entre outros, componentes gráficos, como botões e campos de texto. A região 4 (quatro) apresenta como as entidades de uma aplicação estão dispostas na linha do tempo. Em conjunto com a região *Stage* funcionam como uma poderosa ferramenta de edição espaço/temporal. Os pontos negativos ficam por conta de não ser dado suporte à simulação da interação do usuário durante o processo de criação, e de ser possível somente a especificação do instante absoluto da apresentação de uma mídia na linha do tempo. Além disso, a especificação de relacionamentos espaço/temporais e interatividade só é possível através da codificação de scripts.

A região 5 (cinco) é um editor de ActionScript. Nele, por exemplo, pode ser especificada a ação a ser desempenhada por um determinado componente em resposta a um evento interativo. Para ajudar na codificação, são oferecidos dicas e recursos de destaque do código (*highlight*). Por fim, a região 6 (seis) exibe e permite a edição das propriedades de um componente selecionado.

O produto da autoria nesse ambiente é um ou mais arquivos executáveis, chamados de "SWF" (*Shockwave Flash File*). Esses arquivos podem ser visualizados em um navegador Web, ou utilizando-se o Flash Player.

Por fim, vale destacar que o Macromedia Flash, por não ser voltado para TV digital, não fornece nenhum suporte à edição ao vivo de documentos.

2.7. Análise Comparativa

A seguir, a Tabela 1 sumariza as principais características das ferramentas de autoria discutidas neste capítulo.

Tabela 1 - Análise comparativa entre ferramentas de autoria

Ferramenta	**Paradigma / Modelo**	**Foco**	**Visão temporal / simulação interatividade**	**Suporte a usuários**	**Edição ao vivo**
JAME Author	Procedural (DVB-J) / *page-based services*	Disposição espacial de componentes / interatividade	Não tem	Não suporta usuários avançados	Não
Cardinal Studio	Procedural (DVB-J) / Baseado em atos, camadas e componentes	Disposição espacial de componentes / interatividade	*timeline* simples / Não	Todos	Não
Alti Composer	Procedural (DVB-J) / Baseado em cenas, planos, *shots* e atores	Disposição espacial de componentes / interatividade	*Não tem*	Todos	Não
GRiNS	Declarativo (SMIL) / AHM (Hardman et al., 1994)	Estruturação / Sincronismo de mídias	*structured timeline* / Não	Médio	Não

LimSee2	Declarativo (SMIL) / AHM	Estruturação / Sincronismo de mídias	*structured timeline* / Não	Médio	Não
Macromedia Flash	Procedural / ActionScript	Disposição espacial/ interatividade	*timeline* sincronizada com espacial / Não	Todos	Não