5
Formatador HyperProp

Este capítulo descreve a implementação do ambiente de execução do sistema hipermídia HyperProp (Soares et al., 2000), feita com base no modelo de execução definido no Capítulo 2 e na arquitetura proposta nos Capítulos 3 e 4. Com efeito, o trabalho de implementação serviu para refinar e validar as definições do framework para construção de formatadores hipermídia. Java foi a linguagem de programação escolhida para o desenvolvimento do ambiente de execução, buscando uma solução independente de plataforma.

5.1
Conversor NCM

Esta seção comenta os principais aspectos da tradução de documentos NCM (Casanova et al., 1991; Soares et al., 1995; Soares et al., 2000; Soares et al., 2003) para o modelo de execução do formatador HyperProp, cuja implementação baseia-se no modelo descrito na Seção 2.3. A razão dessa conversão deve-se ao fato do NCM ser o modelo conceitual utilizado pelos ambientes de autoria e armazenamento do sistema HyperProp. Uma vez que o modelo de execução descrito no Capítulo 2 foi diretamente influenciado pelas propostas do modelo NCM, muitas das entidades são facilmente convertidas, conforme tornar-se-á mais claro ao longo desta seção.

O NCM é um modelo orientado a objetos, baseado nos conceitos usuais de nós e elos. Nós representam fragmentos de informação, enquanto elos permitem estabelecer relacionamentos de sincronização espacial e temporal entre os nós.

Um nó NCM pode ser simples ou composto. O nó simples, denominado *nó de conteúdo* (ou *nó terminal*), representa um nó de mídia usual, podendo ser especializado em classes como texto, imagem, vídeo, áudio etc. O nó composto, denominado *nó de composição*, é utilizado para organizar logicamente os

componentes do documento, sendo útil também para oferecer suporte ao reuso de partes de um documento[17].

A Figura 28 ilustra um exemplo de documento estruturado com o uso de composições. O nó de composição *Coisa de Pele* representa o documento, sendo constituído por duas outras composições, *Beth Carvalho* e *Jorge Aragão*, que organizam a interpretação de uma mesma música ("Coisa de Pele") por dois diferentes cantores. A composição *Beth Carvalho* contém o nó composto *Letra*, além de um nó de conteúdo texto *música-Beth* e um nó de conteúdo vídeo *clip-Beth*; enquanto a composição *Jorge Aragão* contém a mesma composição *Letra* e outros dois nós de conteúdo: o texto *música-Jorge* e o vídeo *clip-Jorge*. A composição *Letra* possui nós de conteúdo que contêm os versos da música em fragmentos separados. Relações diferentes das relações de estruturação são representadas no NCM por elos. Por exemplo, um relacionamento de sincronização temporal entre os objetos *música-Beth* e *versos-01* é definido através do elo lb_1. O elo NCM é muito similar ao elo do modelo de execução e seu tratamento será comentado mais adiante.

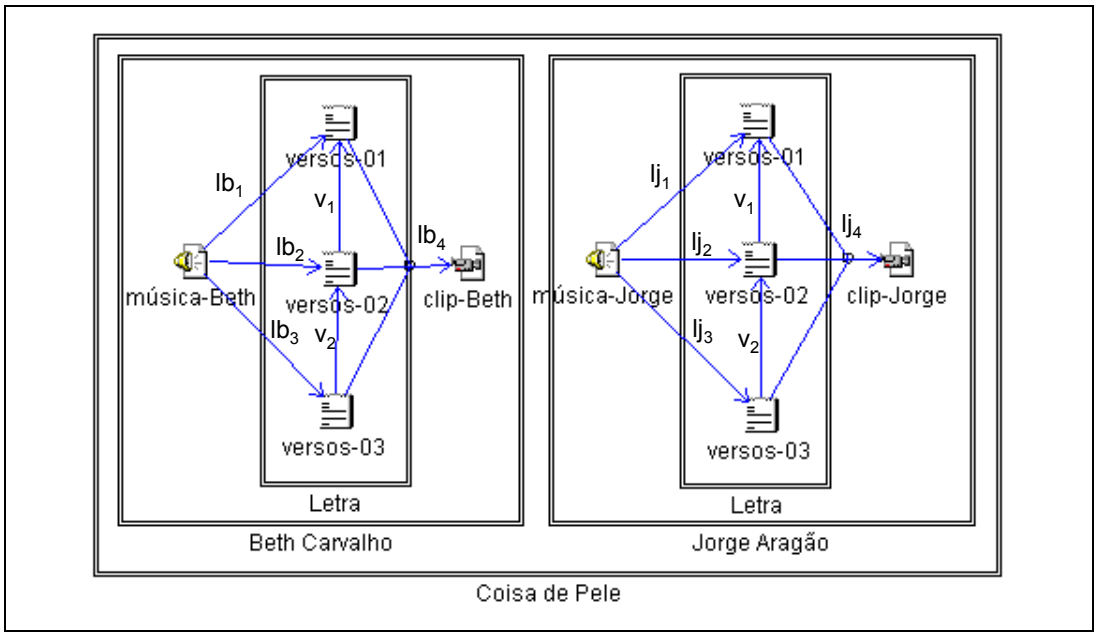


Figura 28 – Exemplo de documento estruturado em composições.

[17] O NCM define composição como uma classe abstrata, de onde derivam diversas subclasses (*contexto do usuário*, *contexto de versões*, *base privada* etc.), que definem diferentes semânticas para a estruturação dos nós (Soares, 2000). Por simplificação, esta seção tratará os nós compostos genericamente como composições, a não ser que seja necessário referir-se a uma determinada especialização.

Como é possível perceber pelo exemplo, o modelo possui um importante recurso de reuso que permite que um mesmo nó (de conteúdo ou de composição) esteja contido em mais de uma composição diferente (por exemplo, o nó *Letra* contido nas composições *Beth Carvalho* e *Jorge Aragão*). Uma vez que o modelo também permite que nós de composição estejam aninhados em qualquer profundidade (composição *Letra* contida na composição *Beth Carvalho*, que por sua vez está contida na composição *Coisa de Pele*), existe definido no NCM o conceito de perspectiva do nó. Intuitivamente, a *perspectiva* de um nó identifica através de que seqüência de nós de composição aninhados uma dada instância de um nó *N* é observada, podendo haver diferentes perspectivas para um mesmo nó *N*, se esse nó estiver contido em mais de uma composição. Por exemplo, na figura, o nó *versos-01* possui duas perspectivas distintas: P_1=*/Coisa de Pele/Beth Carvalho/Letra/versos-01* e P_2=*/Coisa de Pele/Jorge Aragão/Letra/versos-01*. Essa flexibilidade do modelo de autoria, juntamente com o fato dos elos não estarem embutidos nos conteúdos dos nós, constitui-se em um suporte à adaptação estática dos relacionamentos da apresentação, pois os relacionamentos de um objeto poderão variar em função da perspectiva que o nó estiver sendo apresentado. No exemplo, dependendo da perspectiva, o relacionamento que o nó *versos-01* participa como condição é diferente (lb_4 em P_1 e lj_4 em P_2). Para toda perspectiva, o modelo denomina o nó mais interno (no caso do exemplo, o nó *versos-01*) como sendo o *nó base da perspectiva*.

De um modo geral, a operação do conversor NCM consiste em receber como entrada a perspectiva de um nó e retornar um contêiner de execução (Seção 2.3.4). Se o nó base da perspectiva for um nó de conteúdo, o contêiner gerado irá possuir os objetos de execução criados a partir desse nó[18], os elos de restrição (Seção 2.3.2.2) dos quais o nó participa, os elos causais (Seção 2.3.2.1) nos quais o nó atua como condição e os objetos de execução derivados a partir das perspectivas dos nós que ancoram nesses mesmos elos que o nó (e perspectiva) passado como parâmetro. O processo não é recursivo, ou seja, novos elos que ancorem nesses outros nós não são analisados nesse momento pelo conversor. Os objetos de execução derivados desses nós são marcados pelo conversor como

[18] Mais adiante ficará claro que um nó NCM pode resultar em mais de um objeto de execução.

parcialmente construídos. Esse tratamento é útil em um processo incremental de conversão, compilação e execução dos documentos.

Se o nó base da perspectiva for um nó de composição, o conversor irá estender o procedimento de criar os objetos de execução, elos e objetos de execução relacionados também para cada nó que esteja contido, ou recursivamente contido, na composição.

No exemplo da Figura 28, se fosse entregue ao conversor a perspectiva /*Coisa de Pele/Beth Carvalho/música-Beth*, o conversor iria gerar os objetos de execução derivados do nó *música-Beth*, na perspectiva passada como parâmetro, iria tratar os elos lb_1, lb_2 e lb_3, e iria gerar objetos de execução parcialmente construídos para os nós *versos-01*, *versos-02* e *versos-03*, nas respectivas perspectivas *Coisa de Pele/Beth Carvalho/Letra/versos-01*, *Coisa de Pele/Beth Carvalho/Letra/versos-02* e *Coisa de Pele/Beth Carvalho/Letra/versos-03*. Por outro lado, se o conversor recebesse a perspectiva */Coisa de Pele*, todos os nós, elos e perspectivas do exemplo seriam tratados pelo conversor.

No processo de conversão, cada nó NCM (simples ou composto) é diretamente mapeado em um objeto de dados no modelo de execução[19]. Pelo fato da perspectiva de um nó condicionar as relações de sincronização das quais ele participa, a criação dos objetos de execução leva em consideração as perspectivas dos nós. Para cada perspectiva diferente, o conversor NCM cria um objeto de execução separado. Vale lembrar que o modelo de execução permite que diferentes objetos de execução contenham um mesmo objeto de dados.

Além da perspectiva, uma outra entidade fundamental na criação e diferenciação dos objetos de execução é o descritor NCM. O *descritor* NCM reúne as características de apresentação de um nó e é idêntico, em seus atributos, ao descritor definido no modelo de execução (Seção 2.3.1.2). Todavia, devido à maneira flexível que o NCM especifica as associações entre nós e características da apresentação, o processo de criação dos descritores no modelo de execução não ocorre de maneira direta.

[19] Na realidade, o termo *objeto de dados* faz parte da definição do NCM, correspondendo a uma abstração do nó em uma aplicação cliente NCM (Soares, 1998a; Soares, 1998b; Soares, 2003). Essa definição é útil no esquema de tratamento de versões do modelo. Sendo assim, a conversão é, na verdade, de um objeto de dados NCM para um objeto de dados de execução.

Descritores NCM podem ser definidos como atributos do nó, na composição que contém o nó ou nas extremidades dos elos (Soares et al., 2000; Soares et al., 2003). Em todas essas opções, o NCM define, na realidade, uma lista de descritores alternativos, que pode inclusive encontrar-se vazia. Os descritores de execução são gerados pelo conversor NCM como uma união dessas possibilidades, seguindo uma regra de cascateamento explicada a seguir.

Para cada nó (em uma dada perspectiva) que ancora na extremidade de um elo NCM (por exemplo, o nó *música-Beth*, na perspectiva */Coisa de Pele/Beth Carvaho/música-Beth*, ancorando no elo lb_1), o cascateamento dos descritores é calculado através da união dos dados contidos no descritor do nó base da perspectiva (*música-Beth*), no descritor da composição que diretamente contém o nó base da perspectiva (*Beth Carvalho*) e no descritor da extremidade do elo (lb_1). Se, por acaso, dois descritores definirem um mesmo atributo com valores diferentes, o conversor resolve o conflito considerando que o descritor do elo tem prioridade sobre o descritor da composição que, por sua vez, tem precedência sobre o descritor do nó. Os atributos do descritor resultante são então utilizados para compor o descritor do objeto de execução. Quando em qualquer uma das opções (elo, composição ou nó), não houver um descritor definido, a opção é simplesmente ignorada no cascateamento. Por outro lado, se em qualquer uma delas, houver uma lista de descritores com mais de uma opção, ao invés de criar um único objeto de execução, o conversor utiliza cada alternativa de descritor para gerar um descritor de execução distinto e, conseqüentemente, criar objetos de execução diferentes. Nesse caso, o conversor cria também uma alternativa de objetos de execução (Seção 2.3.3) reunindo as várias possibilidades de objetos de execução geradas. As regras de exibição da alternativa de objetos são obtidas dos *atributos de teste* definidos para cada descritor NCM (Muchaluat-Saade, 2003; Soares et al., 2003). Se houver no cascateamento mais de uma lista com mais de uma opção, o conversor gera objetos de execução para todas as combinações possíveis. Evidentemente, esse comportamento pode levar a uma explosão no número de objetos de execução. Uma solução seria resolver a escolha do descritor estaticamente no próprio conversor. Contudo, isso dificultaria que uma mesma conversão fosse reusada em plataformas distintas, por mais de uma vez e por mais de um usuário. Como, na prática, apenas o nó possui descritor definido e, além

disso, dificilmente encontra-se uma lista com mais de dois descritores, a implementação atual de conversor NCM não se preocupa com essa questão.

Um outro suporte a alternativas que o NCM oferece são as alternativas de nó (nó de composição do tipo *switch*) (Muchaluat-Saade, 2003; Soares et al., 2003). Esse nó especial de composição contém uma lista ordenada de nós, com atributos de teste associados. A idéia é que os testes sejam analisados na ordem dos nós e o primeiro nó que tiver o teste satisfeito seja selecionado, oferecendo semântica idêntica à do elemento *switch* da linguagem SMIL (W3C, 2001a). Esses nós alternativos NCM são mapeados diretamente e na mesma ordem em alternativas de objetos de execução, com os atributos de teste sendo convertidos nas regras de exibição a serem associadas aos objetos de execução. Quando um primeiro nó NCM é encontrado em um nó *switch* sem um atributo de teste associado, esse nó é traduzido no objeto de execução *default*. A partir desse ponto os nós NCM restantes são ignorados pelo conversor. Se, por acaso, uma das alternativas de nó NCM em um nó de composição *switch* tiver uma lista de descritores com mais de uma alternativa, a conversão desse nó resultará em uma alternativa de objetos de execução, que por sua vez será inserida na alternativa de objetos de execução sendo construída para os nós. Nesse caso, o resultado no documento de execução será um aninhamento de alternativas.

O ponto de partida do processo de tradução feito pelo conversor é a análise dos elos NCM que ancoram no nó base da perspectiva que lhe foi passada como parâmetro. No caso do nó base ser uma composição, a análise feita também se estende aos elos NCM que ancoram nos nós recursivamente contidos na composição (sempre considerando a perspectiva de cada nó). O resultado dessa etapa é a criação dos elos de execução e dos objetos de execução (podendo esses objetos serem alternativas de objetos de execução) que participam dos relacionamentos, com as suas inserções no contêiner de resposta do conversor. Evidentemente, essa análise também identifica os eventos envolvidos nas sincronizações e os insere na lista de eventos de cada objeto de execução.

O *elo* NCM é definido como estando contido em um nó de composição e sendo constituído por um conector e um conjunto de associações entre os nós e o conector (Muchaluat-Saade & Soares, 2001; Muchaluat-Saade et al., 2002; Muchaluat-Saade, 2003). O conector contém a expressão do relacionamento e as

associações representam as extremidades dos elos, identificando os seus participantes. Essa separação permite que um mesmo conector seja reusado em mais de um elo, favorecendo a autoria e manutenção das relações nos documentos hipermídia (Muchaluat-Saade & Soares, 2001). O modelo NCM define dois tipos de conector: *conector causal* e *conector de restrição*. Elos NCM que usam conectores causais são convertidos em elos de execução causais (Seção 2.3.2.1), enquanto os que usam conectores de restrição são transformados em elos de execução de restrição (Seção 2.3.2.2). As informações contidas no conector são utilizadas pelo conversor para construir o ponto de encontro do elo de execução. A perspectiva do nó de composição que contém o elo juntamente com os dados das associações são utilizados pelo conversor para criar os objetos de execução e respectivos eventos que participam de um relacionamento.

Toda associação em um elo NCM contém a identificação de um nó NCM, a definição de uma seqüência de nós de composição recursivamente contidos na composição que contém o elo[20] e a especificação de uma lista de descritores alternativos. Dessas informações deriva ao menos um objeto de execução. O objeto de dados de execução é obtido diretamente do nó NCM, o descritor de execução é extraído da regra de cascateamento dos descritores NCM explicada anteriormente e, por fim, o identificador único do objeto de execução (Seção 2.3.1) deriva de uma concatenação de identificadores. A concatenação é formada pelos identificadores dos nós NCM que compõem a perspectiva do nó de composição que contém o elo, seguidos dos identificadores dos nós que formam o mapeamento, seguidos do identificador do nó NCM que participa do elo, seguido dos identificadores dos descritores que compõem o descritor resultante (resultado do cascateamento). Sendo assim, dois objetos de execução são considerados diferentes pelo conversor, se tiverem alguma distinção na perspectiva ou no cascateamento de descritores. Para todo objeto de execução criado pelo conversor, ele cria também um evento de apresentação que é inserido na lista de eventos do objeto. Esse evento representa a exibição do conteúdo do objeto como um todo.

[20] O NCM permite que uma composição defina mapeamentos de suas âncoras (pontos de interface) para âncoras de nós que estejam nela contidos. Se esses nós forem composições, o processo pode se repetir em vários níveis de aninhamento. Isso obriga que elos apontem para nós contidos, ou recursivamente contidos, em composições apenas através de pontos de interface bem definidos, garantindo a propriedade de composicionalidade do modelo.

Retornando ao exemplo da Figura 28 para ilustrar as definições apresentadas, o elo lb_1 está contido no nó de composição *Beth Carvalho*, que possui uma única perspectiva no documento dada por *Coisa de Pele/Beth Carvalho*. O elo possui ainda duas associações, uma com o nó *música-Beth* e outra com o nó *versos-01*, sendo que essa última associação passa por um mapeamento definido entre o nó de composição *Letra* e o nó de conteúdo *versos-01*. Assumindo que para esses nós de conteúdo existam apenas dois descritores definidos no documento: *desc-beth* como atributo do nó *música-Beth* e *desc-versos-01* como atributo do nó *versos-01*, os identificadores dos objetos de execução relacionados pelo elo de execução, resultado da tradução do elo NCM lb_1, seriam: *Coisa de Pele+Beth Carvalho+música-Beth+desc-Beth* e *Coisa de Pele+Beth Carvalho+Letra+versos-01+desc-versos-01*.

Além do nó, dos mapeamento para alcançar o nó e da lista de descritores, toda associação no elo NCM possui também a especificação de um tipo de evento (*exibição*, *clique do mouse*, *atribuição* etc.) e o identificador de uma âncora, ou de um atributo, do nó NCM. O conversor utiliza essas informações para descobrir o evento de execução (Seção 2.3.1.3) que deve participar do elo. Em seguida, caso o evento ainda não exista, o conversor o insere na lista de eventos do objeto de execução[21].

Após a análise de todos os elos e criação dos objetos de execução e eventos, o conversor percorre todas as perspectivas de nós que devam ser tratadas pelo conversor, e cria os objetos de execução (ou alternativas de objetos de execução), para cada nó base de perspectiva, aplicando a mesma regra de cascateamento apresentada anteriormente, porém, desconsiderando a lista de descritores dos elos. Essa etapa pode, ou não, resultar na inserção de mais alguns objetos de execução no contêiner a ser respondido.

[21] Os eventos NCM também são baseados em máquinas de estados e seus tipos e máquinas são traduzidos diretamente nas classes de evento e máquinas de estados definidas no modelo de execução. A única exceção são os eventos de preparação NCM, que no modelo de execução são tratados como fazendo parte da máquina de estados do evento de apresentação.

5.1.1
Suporte à apresentação de documentos descritos em outros modelos de autoria

O conversor NCM implementado junto ao formatador cria uma nova possibilidade para o controle de apresentações hipermídia no sistema HyperProp. Documentos definidos em outros modelos de autoria podem ser apresentados pelo ambiente de execução do sistema HyperProp, se houver um mapeamento do modelo em questão para o NCM. Seguindo essa lógica de integração, foram desenvolvidos e implementados um conversor de documentos SMIL versão 1.0 (W3C, 1998) para documentos NCM (Rodrigues et al., 1999; Rodrigues, 2000; Rodrigues et al., 2002) e um conversor da linguagem NCL para documentos NCM (Antonacci et al., 2000; Rodrigues et al., 2002; Muchaluat-Saade, 2003).

A principal vantagem de efetuar a conversão entre os modelos no nível da autoria é permitir que as funcionalidades de um modelo com maior poder de expressão sejam acrescidas ainda na fase de edição dos documentos (Rodrigues et al., 2002). Evidentemente, entidades que não possam ser mapeadas de um modelo para o outro geram uma perda de características e recursos na conversão. Outra vantagem da conversão nesse nível é explorar as ferramentas de edição disponíveis para um determinado modelo, na autoria de documentos descritos em modelos diferentes. Nesse caso, o ideal é que a conversão entre modelos possa ser feita nos dois sentidos, novamente estando sujeita à perda de expressividade na passagem de um modelo para o outro.

Existe, por outro lado, uma desvantagem em termos de eficiência quando a conversão para apresentação é primeiro feita no nível de autoria, pois duas conversões em seqüência precisam ser aplicadas para simplesmente exibir (sem editar) um documento descrito em um modelo diferente do NCM. Por essa razão, como trabalho futuro, pretende-se desenvolver um conversor que faça diretamente a tradução de documentos escritos na versão 2.0 da linguagem SMIL (W3C, 2001a) para o modelo de execução definido nesta tese. Obviamente, a arquitetura possibilita que outros conversores também sejam desenvolvidos, desde que exista um mapeamento das características básicas para o modelo de execução.

5.2 Compilador e executor do orquestrador de apresentação do formatador HyperProp

Os dois principais componentes do formatador no controle da apresentação são o *compilador de apresentação* e o *executor de apresentação*, denominados simplesmente de *compilador* e *executor*. Esses componentes implementam as funções tanto do plano de serviços como do plano de meta-serviços descritas na Seção 4.2 e ilustradas na Figura 23.

O compilador HyperProp é o ponto de entrada no processo de orquestração das apresentações no sistema. O compilador dispõe de um método que espera como parâmetros um contêiner de execução, a identificação de um objeto de execução contido no contêiner e a identificação de um evento de apresentação contido no objeto de execução. O contêiner informa para o compilador a descrição do documento, enquanto o objeto de execução e o evento permitem que seja conhecido o ponto de entrada a partir do qual o documento será iniciado. Futuramente, pretende-se estender a interface para permitir que o usuário informe o tempo máximo permitido para que a compilação seja feita e o instante programado para que a execução do documento seja iniciada.

De posse das informações recebidas, o compilador constrói as cadeias temporais do documento, que são implementadas como grafos temporais, onde os nós representam transições nas máquinas de estados dos eventos e as arestas são relações síncronas e previsíveis entre os nós. Cada aresta possui associada uma informação de duração, que é modelada como uma função de custo do tipo *piece wise* convexa e linear em todas as suas subdivisões (Seção 4.3.1).

O compilador HyperProp também analisa os elos causais e registra cada elo como observador dos eventos que estejam associados a condições de transição em seu ponto de encontro. Isso faz com que as transições nas máquinas de estados só exijam a avaliação dos elos que realmente dependam daquele evento.

Outro tipo de associação de observação que o compilador estabelece é entre a propriedade de transição e o seu evento. Isso permite que quando a transição ocorra, a propriedade registre o instante da ocorrência para posterior avaliação por parte dos elos. O controle de deslocamento utilizando os atributos *minDelay* e *maxDelay* das expressões de propriedades (Seção 2.3.2.1) ainda é um ponto em

aberto na implementação do formatador, sendo os valores desses atributos desconsiderados na implementação atual. A única exceção é para as *expressões de propriedades de transição*. Nesse caso, o formatador sabe como manter a semântica estabelecida pelo modelo de execução.

Na fase de construção do plano de execução, o compilador avalia todas as alternativas de objetos de execução que podem ser resolvidas estaticamente e o faz com o auxílio do proxy de informações contextuais. Na implementação corrente, esse proxy é uma implementação simples de gerente de contexto que mantém informações da plataforma, carregadas a partir de arquivos locais. Como trabalho futuro, pretende-se integrar esse componente a um mecanismo real de gerência do contexto de exibição e incluir a manutenção de informações a respeito do usuário.

Após a construção do plano de execução, o compilador inicia, se estiverem presentes, as tarefas de ajuste dos tempos elásticos e compilação do plano de pré-busca, que serão comentadas mais adiante. A funcionalidade de gerência da orquestração intra-mídia é outro recurso que não está implementado e que deve ser abordado futuramente.

Durante a apresentação do documento, o executor identifica e instancia as ferramentas de exibição. Esse controle é feito com o auxílio do gerenciador de ferramentas (Seção 3.2). O gerenciador obtém do proxy de informações contextuais a relação das ferramentas disponíveis no sistema e os tipos MIME (Freed & Borenstein, 1996) que as ferramentas sabem tratar. Quando um objeto de execução precisa ser exibido, o gerenciador de ferramentas verifica primeiro se existe uma ferramenta especificada no descritor. As ferramentas devem ser identificadas pelo nome da classe Java que contém a implementação propriamente dita. Se não houver uma ferramenta especificada no descritor, ou se a ferramenta especificada não estiver disponível na plataforma, o gerenciador consulta o tipo MIME do conteúdo em procura de uma ferramenta que saiba lidar com o formato identificado. Se não for possível exibir o conteúdo, o gerenciador retorna o erro e, na implementação atual, o executor simplesmente desabilita o objeto de execução no documento. A maneira como está implementada a descoberta e instanciação das ferramentas permite que sejam instaladas novas ferramentas na plataforma, sem precisar que o gerente de ferramentas seja recompilado.

Para a disposição espacial das ferramentas, o executor utiliza um gerente de layout, que converte as informações descritas nas janelas e regiões do layout do documento (Seção 2.3.1.2) para janelas (classe *javax.swing.JFrame*) e painéis (classe *javax.swing.JPanel*) oferecidos pela plataforma Java 2. O gerenciador mantém também uma janela e uma região *default* que são utilizadas para exibir os objetos que não apresentem a identificação correta da região em seus descritores.

O executor permanece como um observador dos elos causais, sendo notificado toda vez que uma condição é satisfeita. O executor então coordena o disparo das ações, cuidando de requisitar os serviços do gerenciador de ferramentas, quando for necessário instanciar um novo exibidor. Se o executor percebe que um objeto de execução parcialmente construído precisa ter o evento de apresentação do seu conteúdo preparado, ou mesmo ter sua ocorrência iniciada, o executor gera uma notificação ao formatador, que a repassa para objetos que tenham se registrado como interessados nessa notificação. O formatador fica então esperando que um novo contêiner lhe seja entregue contendo o objeto de execução em questão completamente construído. A princípio, essa espera é independente do transcorrer do restante da apresentação.

O executor possui implementado um elemento que monitora o início da ocorrência dos eventos de apresentação e registra no plano de execução o instante efetivo. O próximo passo do trabalho é desenvolver estratégias de ajuste que corrijam o plano quando diferenças significativas entre os tempos programados e os tempos efetivamente ocorridos forem percebidas. No entanto, é importante destacar que, apesar de não dispor de adaptações e ajustes dos tempos na fase de execução, o fato do plano de execução não ser um simples *timeline*, o fato do disparo das ações ser baseado em um modelo de observação dos eventos e a maneira como as ferramentas de exibição monitoram as máquinas de estados (o monitoramento será explicado no final da próxima seção) já oferecem um suporte a uma execução adaptativa das apresentações.

É também função do executor observar se as ações a serem disparadas devem iniciar a execução de uma cadeia temporal parcial. Nesse caso, o executor realiza a junção dessa cadeia auxiliar à cadeia principal do documento.

O formatador HyperProp oferece em sua interface botões que permitem controlar a exibição como um todo. É permitido pausar, retomar e encerrar a

exibição a qualquer momento. O próximo passo é investigar a viabilidade de dispor o formatador de meios para avançar ou adiantar o ponto da apresentação, sem que isso comprometa em demasia a qualidade da apresentação.

5.3 Integração entre as ferramentas de exibição e o formatador

O sistema HyperProp possui implementadas ferramentas de exibição para tratar o conteúdo dos principais tipos de mídia (texto, imagens, áudio e vídeo) (Rodrigues, 2000; Rodrigues et al., 2001). A Figura 29 ilustra um resumo do diagrama de classes para implementação das ferramentas de exibição. Todas as ferramentas herdam, direta ou indiretamente, os atributos e métodos definidos na classe *PresentationTool*, apresentada no Capítulo 3.

O sistema dispõe de uma ferramenta para exibição de texto HTML (classe *HtmlTextPresentationTool*) que, para realizar a interpretação e formatação dos conteúdos, utiliza as classes *HTMLEditorKit* e *HTMLDocument*, disponíveis nos pacotes *javax.swing.text.html* e *javax.swing.text.html.parser* da plataforma padrão Java 2 (J2SE). Além disso, a ferramenta implementa a interface *HyperlinkListener* que lhe permite ser avisada sempre que ocorre algum evento de interação sobre uma âncora HTML (elemento <a> da linguagem HTML) (W3C, 1999c).

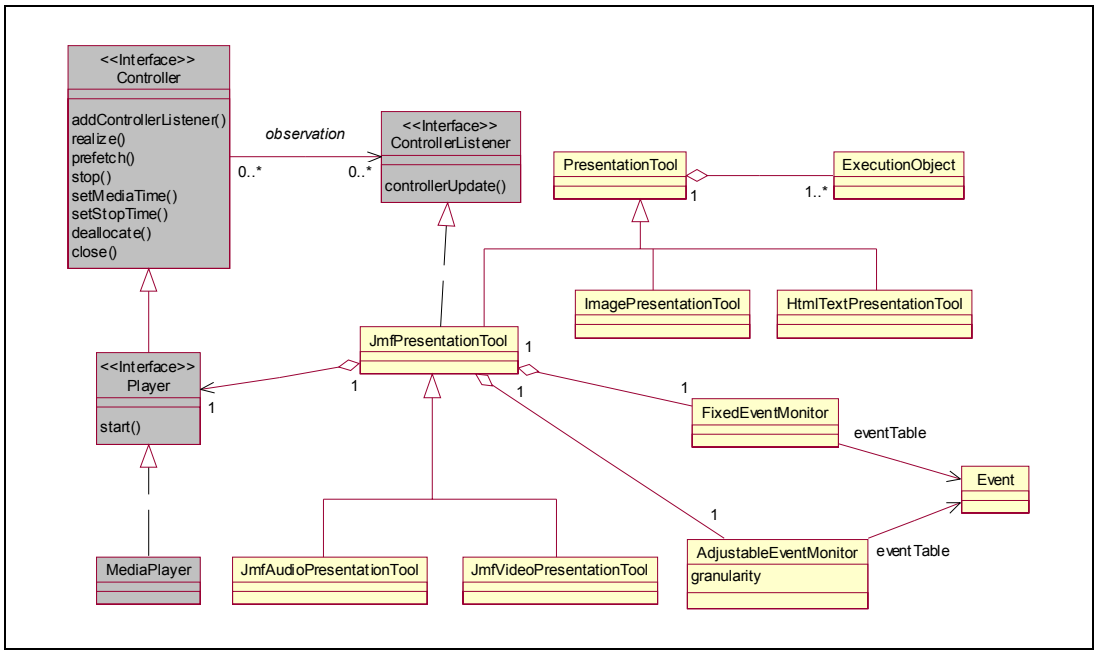


Figura 29 – Diagrama de classes das ferramentas de exibição.

Na etapa de preparação do objeto de execução, a ferramenta utiliza os atributos *posição* e *string* das âncoras textuais contidas nos eventos relacionados a ações do usuário (eventos das classes *MouseClickEvent*, *MouseOverEvent* e *FocusEvent* – Seção 2.3.1.3), para identificar no conteúdo as regiões do texto que devem estar sensíveis às respectivas interações. A ferramenta então insere, dinamicamente, no conteúdo, as marcações HTML que oferecem os pontos de interação. Essas marcações são incluídas no texto da seguinte forma: *âncora textual*. Toda vez que ocorre uma ação sobre âncora, a ferramenta é sinalizada por uma chamada à sua implementação do método *hyperlinkUpdate*, definido na interface *HyperlinkListener*. A ferramenta então identifica o evento que foi selecionado (*eventId*) e gera em sua máquina de estados a transição correspondente. A partir daí, o mecanismo de observação dos elos faz com que o executor seja notificado da transição e, caso um ou mais elos causais tenham suas condições satisfeitas, as ações dependentes dessa interação sejam disparadas.

Essa abordagem permite reusar conteúdo HTML e inserir elos de outros modelos, sem que seja necessário editar o conteúdo do objeto (Rodrigues & Soares, 1998). A forma como os elos são colocados no nó também permite que uma mesma página, dependendo da perspectiva (ou seja, do objeto de execução), apresente diferentes regiões selecionadas ou mesmo diferentes ações para a mesma região. A Figura 30 ilustra a interface da ferramenta de exibição para a mídia texto, onde a palavra *cantor(a)* é uma âncora HTML inserida dinamicamente pela ferramenta.

Textos podem ser exibidos com duração pré-definida, bastando para isso que o evento de apresentação do conteúdo como um todo tenha uma duração estipulada. Nesses casos, a ferramenta utiliza um monitor que controla os tempos dos eventos de apresentação e gera as transições quando transcorre a duração estipulada. Monitores de eventos serão discutidos em mais detalhes na descrição das ferramentas para exibição de mídias contínuas.

Para imagens estáticas, o sistema possui uma ferramenta (classe *ImagePresentationTool*), que utiliza o suporte da classe *Image*, presente no pacote *java.awt*, para controlar a exibição das figuras. A ferramenta permite que sejam definidas âncoras espaciais e temporais (Seção 2.3.1.1) e monitora a máquina de

estados para os eventos correspondentes. Da mesma forma que a ferramenta de texto, imagens também podem ser apresentadas com durações pré-definidas, sendo, para isso, utilizado o mesmo tipo de monitor citado anteriormente.

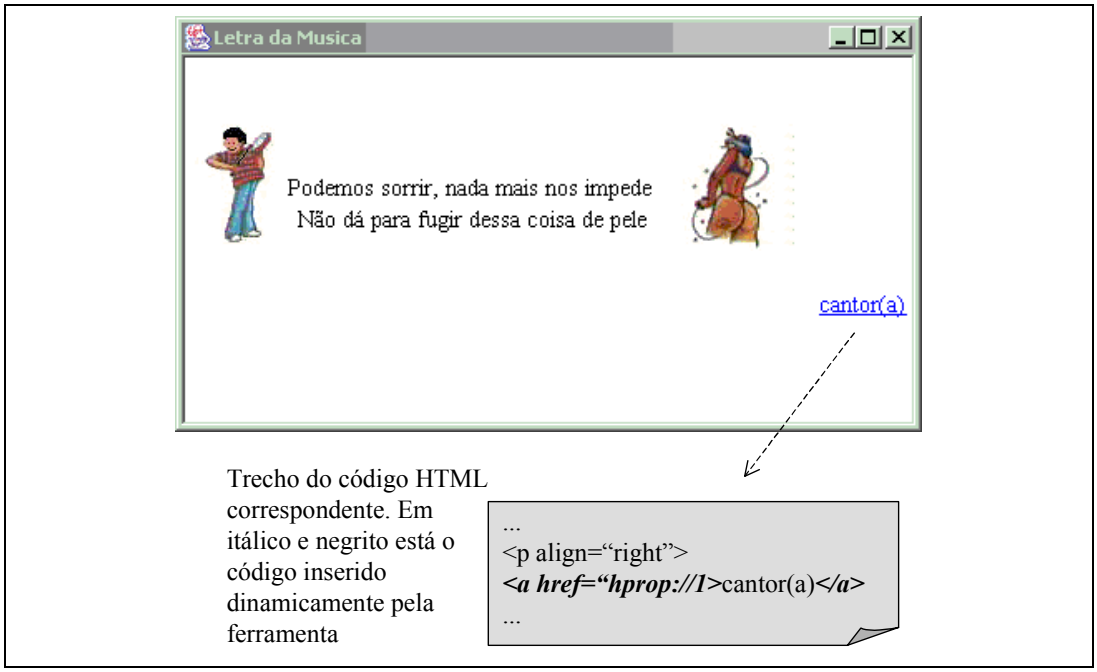


Figura 30 – Ferramenta de exibição de texto no formato HTML.

Para mídias contínuas, as ferramentas do sistema foram implementadas utilizando exibidores JMF (*Java Media Framework*), disponíveis no framework oferecido pela Sun Microsystems (Sun, 1999). O JMF é um pacote de classes e interfaces que oferece suporte à apresentação dos principais formatos de conteúdo dinâmico (PCM, MPEG-1, MP3 etc.).

Foi criada no sistema uma classe, chamada *JmfPresentationTool*, que reúne as características comuns às ferramentas baseadas nos exibidores JMF. Essa classe herda da classe *PresentationTool* e dela herdam duas subclasses: *JmfAudioPresentationTool* (para exibição de áudio) e *JmfVideoPresentationTool* (para exibição de vídeo). Como essas ferramentas utilizam os próprios exibidores JMF (os *players* JMF), as respectivas classes atuam, na verdade, como adaptadores para a API oferecida pelo framework da Sun, realizando o mapeamento entre a interface definida na Seção 3.3.1 e a API oferecida pelo JMF. A Figura 31 ilustra a ferramenta de exibição JMF exibindo um vídeo no formato MPEG-1.

Figura 31 – Ferramenta de exibição integrada a exibidores JMF.

No diagrama da Figura 29, as classes e interfaces com fundo cinza fazem parte do pacote da Sun. A interface *Player* é implementada pelos exibidores JMF (classes que herdam de *MediaPlayer*). A função de um *Player* é processar um fluxo de dados de uma mídia contínua e apresentá-lo nos dispositivos de saída, eventualmente oferecendo alguns componentes gráficos para permitir que o usuário interaja com a apresentação. Todo *Player* estende a interface *Controller*. Um *Controller* define uma máquina de estados, ilustrada na Figura 32, que descreve o funcionamento do exibidor JMF e, por conseqüência, o evento de apresentação do conteúdo do objeto de execução sendo exibido. Transições na máquina de estados são notificadas para aqueles que tenham se registrado como observadores do exibidor JMF, através da chamada ao método *addControllerListener*. Todo observador deve ser instância de uma classe que implemente a interface *ControllerListener*, pois as notificações ocorrem através de chamadas ao método *controllerUpdate*. Dessa forma, a classe *JmfPresentationTool* implementa a interface *ControllerListener*, já que, como um adaptador, ela necessita monitorar o estado do exibidor JMF.

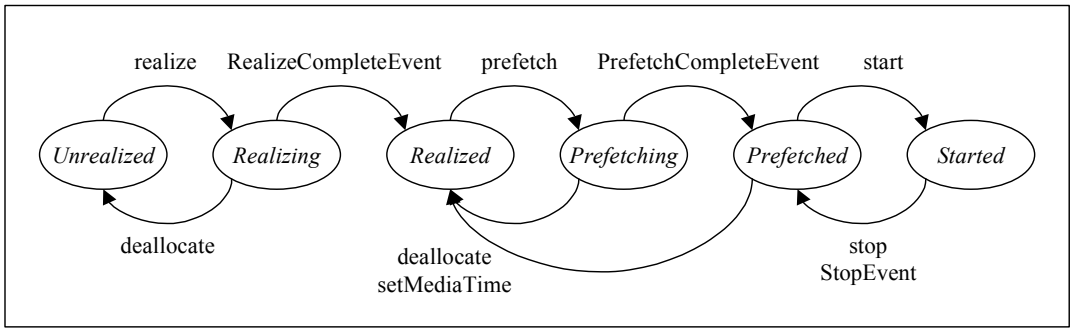


Figura 32 – Máquina de estados de um exibidor JMF

Quando um exibidor JMF é criado, ele se encontra no estado *unrealized*. A chamada ao método *realize*, faz com que o exibidor passe para o estado *realizing*, iniciando o processo de verificação e alocação de alguns dos recursos necessários. Ao final desse processo, seu estado torna-se *realized*, onde já possui informação sobre o tipo de mídia que irá apresentar. O método *prefetch* leva o exibidor para o estado *prefetching*, onde é iniciado o processo de carregamento do conteúdo a ser apresentado. Ao passar para o estado *prefetched*, o exibidor está pronto para iniciar a apresentação da mídia. No momento em que o método *start* é invocado, o exibidor passa para o estado *started*, quando efetivamente começa a apresentação. Um exibidor JMF no estado *started* retorna ao estado *prefetched* quando o método *stop* é chamado, quando atinge o fim da apresentação ou quando a transmissão dos dados é interrompida. Além de notificar as transições em seus estados, um exibidor também notifica seus observadores quando é destruído e quando ocorrem alterações nos valores de seus atributos, tais como duração e taxa de exibição.

Além dos métodos que geram transições na máquina de estados (*realize*, *prefetch*, *start* e *stop*), o exibidor JMF oferece outros dois importantes métodos em sua interface: *setMediaTime*, que permite alterar a posição corrente no fluxo da mídia, e *setStopTime*, que possibilita estabelecer a posição no fluxo da mídia onde a apresentação será interrompida[22].

Como pode ser percebido, é grande a semelhança entre a máquina de estados dos eventos de apresentação (Figura 8) e a máquina de estados de um exibidor JMF. Conseqüentemente, são também similares os métodos oferecidos pela classe *PresentationTool* (Tabela 4) e pela interface *Controller*. A Tabela 5 resume a

[22] Os métodos *setMediaTime*, *setStopTime* e *stop* são, na realidade, definidos na interface *Clock*, da qual deriva a interface *Controller*. Porém, essa e algumas outras interfaces e classes do JMF foram omitidas para que os diagramas não ficassem sobrecarregados.

maneira como é feita a conversão entre as duas interfaces, destacando apenas os métodos definidos na Seção 3.3.1 relevantes para o mapeamento.

Métodos de *PresentationTool*	**Métodos correspondentes para o exibidor JMF**
initialize(exObj)	*realize()*
prepare(exObj, evt)	*setMediaTime(T_i) + setStopTime(T_f) + prefetch()*
start(exObj, evt)	*start()*
stop(exObj, evt)	*stop() + setMediaTime(T_i)*
pause(exObj, evt)	*stop()*
resume(exObj, evt)	*start()*
abort(exObj, evt)	*stop() + setMediaTime(T_i)*
Unprepare(exObj, evt)	*deallocate() + close()*

T_i = instante de início da apresentação de E, T_f = instante de término da apresentação de E

Tabela 5 – Mapeamento entre ações solicitadas pelo formatador e os métodos invocados no exibidor JMF.

A iniciação da ferramenta JMF reflete na preparação do exibidor JMF, representada pela chamada ao seu método *realize*. A ação *prepare* resulta na chamada a três métodos do exibidor. O método *setMediaTime* estabelece a posição no fluxo de mídia onde a apresentação será iniciada, enquanto o método *setStopTime* especifica a posição de término da apresentação. Chamadas a esses dois métodos são realizadas com base nos instantes de início e término do evento passado como parâmetro na preparação (ou mesmo pedido de disparo da exibição). Se o evento for o objeto inteiro e a duração não estiver explicitada, os métodos *setMediaTime* e *setStopTime* não precisam ser chamados, fazendo com que a apresentação seja conduzida do princípio do conteúdo até o seu término natural, ou até que seja interrompida pelo usuário. Após a definição das posições de início e fim da apresentação, o método *prefetch* é acionado para disparar a busca do conteúdo. A ação *start* causa uma chamada ao método *start* do exibidor JMF, enquanto a ação *stop* gera uma chamada ao método *stop* do exibidor, seguida de uma chamada ao método *setMediaTime*, para recolocar a exibição em seu ponto inicial, pois o método *stop* do JMF possui a semântica de simples pausa. Por esse motivo, a ação *pause* gera apenas uma chamada ao método *stop* do exibidor. Por sua vez, a ação *resume* reinicia a exibição a partir do ponto onde ela foi interrompida, através de uma nova chamada ao método *start* do exibidor JMF. A ação *abort* finaliza a apresentação, de modo semelhante à ação *stop*, porém fazendo com que os eventos que estejam no estado *ocorrendo* ou *suspenso* passem

instantaneamente pelo estado *abortado* antes de retornar ao estado *preparado*. Por fim, a ação *unprepare* libera os recursos ocupados pelo exibidor, sendo ignorada se algum evento estiver ocorrendo ou suspenso. Para os métodos *unprepare*, *stop*, *pause*, *resume* e *abort* o evento passado como parâmetro pode ser nulo, pois o mesmo não é considerado. Essas ações são sempre aplicadas em todos os eventos passíveis de terem em suas máquinas de estados as respectivas transições realizadas.

Outra função importante da ferramenta/adaptador JMF é interceptar as notificações geradas pelo exibidor JMF e efetuar a tradução para o modelo de máquinas de estados de eventos utilizado pelo formatador. Os principais tipos de notificação gerados pelos exibidores JMF são: final do carregamento do conteúdo necessário para iniciar a exibição (*PrefetchCompleteEvent*); início da exibição (*StartEvent*); suspensão da exibição por chamada explícita ao exibidor (*StopByRequestEvent*); final da exibição por alcance do tempo de término previamente estabelecido (*StopAtTimeEvent*); final da exibição por término natural (*EndOfMediaEvent*); e destruição do exibidor (*ControllerClosedEvent*). A Tabela 6 apresenta o mapeamento dos diferentes tipos de sinalizações dos exibidores JMF para as transições correspondentes na máquina de estados dos eventos de apresentação.

Sinalizações do exibidor JMF	Transições no estado do evento de apresentação
PrefetchCompleteEvent	*preparando* → *preparado*
StartEvent	*preparado* | *suspenso* → *ocorrendo*
StopByRequestEvent *em resposta ao método stop* *em resposta ao método pause* *em resposta ao método abort*	 *ocorrendo* | *suspenso* → *preparado* *ocorrendo* → *suspenso* *ocorrendo* | *suspenso* → *abortado* → *preparado*
StopAtTimeEvent	*ocorrendo* → *preparado*
EndOfMediaEvent	*ocorrendo* → *preparado*
ControllerClosedEvent	*ocorrendo* | *suspenso* → *abortado* → *preparado*

Tabela 6 – Mapeamento entre notificações do exibidor JMF e transições de estado do evento de apresentação

Nesse ponto cabe destacar uma importante distinção entre a API JMF e a interface para integração com formatadores proposta neste trabalho. Enquanto as ferramentas de exibição devem controlar uma máquina de estados para cada

evento/âncora do objeto sendo exibido, de uma forma mais limitada, os exibidores JMF trabalham com apenas uma máquina de estados, correspondendo, na maioria das vezes, ao objeto como um todo. As ferramentas JMF implementadas no formatador HyperProp utilizam monitores temporais, descritos a seguir, para alcançar esse controle mais granular das exibições dos objetos.

5.3.1 Monitorando os tempos dos eventos de apresentação

Enquanto um evento de apresentação, definido por um intervalo temporal *I* (evento externo), permanece no estado *ocorrendo*, outros eventos de apresentação do mesmo objeto, definidos em subintervalos de *I* (eventos internos), podem ocorrer também. Uma funcionalidade que deve estar presente nas ferramentas de exibição, principalmente as de mídia contínua, é a capacidade de sinalizar o início e o fim da exibição de todos os eventos de apresentação que venham a ocorrer na exibição de um objeto. Isso permite definir relações de sincronização entre sub-regiões temporais dos objetos, conforme ilustrado no Exemplo 1 da Seção 2.3.2.1. No exemplo, a apresentação do vídeo V_1 corresponde ao evento externo, enquanto ev_1 corresponde a um evento interno. Entretanto, esse controle granular, que é oferecido de uma forma simples pela interface proposta nesta tese, não é trivial de ser obtido através da API dos exibidores JMF. Sendo assim, efetuar essa adaptação constitui-se em uma tarefa importante para a classe *JmfPresentationTool* e uma das principais contribuições dessa implementação. De fato, é essa adaptação que permite transformar a máquina de estados única do JMF nas várias máquinas de estados dos eventos de um objeto de execução.

Quando iniciada, a ferramenta percorre a lista de eventos do objeto de execução e identifica as âncoras dos eventos de apresentação que precisam ter suas ocorrências controladas. A maneira como são especificados os tempos que delimitam o intervalo da âncora de um evento de apresentação pode variar e é importante que seja tratada pela ferramenta. Os delimitadores da região podem definir explicitamente unidades de tempo medidas em relação ao início da apresentação do objeto como um todo, sem levar em consideração variações na taxa de exibição da mídia (*eventos fixos* – âncoras da classe *TimeIntervalAnchor*, definida na Seção 2.3.1.1). Nesse caso, a ferramenta precisa guiar-se apenas pelo

relógio do sistema (relógio da máquina em que estiver executando). De uma outra forma, as extremidades da região temporal podem estar associadas a unidades de informação específicas do conteúdo, como por exemplo quadros de um vídeo ou amostras de um áudio (*eventos ajustáveis* – âncoras da classe *SampleIntervalAnchor*, definida na Seção 2.3.1.1). Nesse modo de definição, a ferramenta deve ter um controle exato sobre a taxa de apresentação da mídia, pois variações devem refletir em atualizações nos instantes de delimitação do intervalo temporal que o evento especifica.

Para auxiliar no controle dos eventos de apresentação, a ferramenta *JmfPresentationTool* possui dois monitores (Figura 29). O *monitor não ajustável* (*FixedEventMonitor*) auxilia no controle dos eventos fixos do objeto, enquanto o *monitor ajustável* (*AdjustableEventMonitor*) controla os eventos ajustáveis. Ambos os monitores constróem tabelas com registros que contêm as seguintes informações: a identificação do evento, o tipo da transição (início ou fim da apresentação da região em questão), o instante de tempo esperado para que a transição aconteça e se a transição está ou não habilitada. As tabelas de eventos têm seus registros ordenados de forma crescente pelos instantes de tempo esperados para as transições.

A habilitação de uma transição depende do ponto em que o conteúdo do objeto começa a ser exibido, pois dependendo do evento de apresentação inicial, algumas máquinas de estados não devem ser consideradas pela ferramenta de exibição. A Tabela 7 resume as regras adotadas pelos monitores das ferramentas JMF para decidir quais transições devem ou não estar habilitadas na tabela de eventos. Essas regras são baseadas nas relações entre o intervalo do evento solicitado para ser apresentado (evento α) e o intervalo de qualquer outro evento de apresentação do objeto (evento β). Na tabela, é considerado que os eventos α e β possuem intervalos de duração $[T_{\alpha i}, T_{\alpha f}]$ e $[T_{\beta i}, T_{\beta f}]$, respectivamente.

Os monitores de eventos são implementados como *threads*, disparadas pela própria ferramenta. O monitor não ajustável possui um funcionamento bastante simples. Quando iniciado, ele procura o primeiro registro na tabela que esteja habilitado e consulta o tempo esperado para a transição. Em seguida, ele se coloca para dormir até que esse instante seja alcançado. Quando acorda, o monitor simplesmente coloca o evento no estado adequado (*ocorrendo* se for uma

transição de início e *preparado* se for uma transição de fim), procura na tabela o próximo registro habilitado e volta a dormir até que o novo instante de tempo previsto seja atingido.

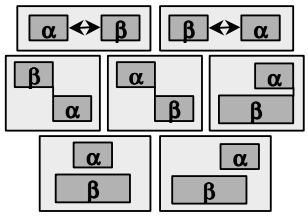
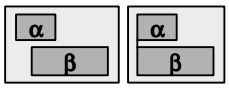
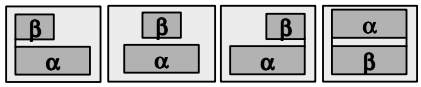
Relação Temporal	Ilustração da Relação	Regra
$T_{\alpha i} > T_{\beta i}$ OU $T_{\alpha f} \leq T_{\beta i}$		As transições de início e de fim de α permanecerão habilitadas. As transições do evento β serão desabilitadas, pois a máquina de estados de β deverá ser desconsiderada.
$T_{\alpha i} \leq T_{\beta i} < T_{\alpha f}$ E $T_{\alpha f} < T_{\beta f}$		Todas as transições permanecerão habilitadas. No entanto, o fim da ocorrência de α irá causar o corte na apresentação de β. Dessa forma, o evento β retornará para o estado *preparado* prematuramente.
$T_{\alpha i} \leq T_{\beta i}$ E $T_{\alpha f} \geq T_{\beta f}$		Todas as transições permanecerão habilitadas e serão notificadas se não houver uma intervenção externa do usuário.

Tabela 7 – Regras para habilitação dos eventos de apresentação.

O monitor ajustável possui um funcionamento mais elaborado. Essa maior complexidade vem do fato da taxa de exibição do objeto de mídia poder variar por fatores imprevisíveis, tais como atrasos nos sistemas de comunicação e operacional ou interações diretas do usuário com os botões de controle da ferramenta (barra de controle inferior na Figura 31). Isso torna impossível para o monitor estimar, de maneira precisa, o tempo que deve dormir até uma próxima transição. Além disso, a API JMF não oferece uma forma para que os observadores de seus exibidores se registrem para serem notificados quando pontos internos da exibição do objeto forem atingidos. Por essas razões, o monitor ajustável realiza sucessivas verificações junto ao exibidor JMF para saber o instante de exibição corrente do conteúdo. O intervalo entre verificações é dado pela variável *granularity* do monitor, que pode ser configurada. Evidentemente, se

a freqüência de verificação for alta, ocorrerá uma sobrecarga na utilização da CPU. Por outro lado, se a freqüência for baixa, poderá haver uma imprecisão nos instantes de notificação. Quando a ferramenta tem a taxa de apresentação da mídia alterada, tanto pela ação do usuário como do formatador, a granularidade configurada no monitor é também modificada proporcionalmente.

Após cada verificação, o monitor ajustável procura por transições habilitadas cujo tempo esperado tenha sido alcançado, ou mesmo ultrapassado, e que ainda não tenham sido sinalizadas para o formatador. Se houver alguma, o monitor efetua a transição correspondente na máquina de estados do evento, da mesma forma que o monitor não ajustável.

Os monitores também observam alguns aspectos imprevisíveis da apresentação para corrigir suas posições correntes nas tabelas, recalcular os tempos em que devem dormir e acertar os estados dos eventos que estão sendo monitorados. Um caso que exige correção é quando o usuário modifica o ponto de apresentação da mídia.

Os monitores oferecem métodos que permitem que eles sejam pausados, retomados no controle ou tenham sua execução encerrada. Esse controle é feito pela ferramenta JMF, respondendo a chamadas aos seus métodos *pause*, *resume*, *abort* e *stop*.

Por uma questão de simplificação, quando a apresentação de um evento acaba de ser preparada pela ação *prepara*, todos os eventos de apresentação cujas transições permaneceram habilitadas são também colocados no estado *preparado*. Isso permite que, à medida que as suas regiões sejam apresentadas, os eventos associados passem diretamente do estado *preparado* para o estado *ocorrendo*.

Por fim, cabe mencionar que as ferramentas de exibição desenvolvidas também são capazes de tratar os eventos de seleção em regiões espaciais dos objetos, assim como a especificação de eventos que combinam apresentação e seleção. Maiores detalhes podem ser obtidos na referência (Rodrigues, 2000).

5.4
Mecanismo de pré-busca

Uma instanciação simplificada do framework apresentado na Seção 4.1.1 foi realizada com o intuito de incorporar mecanismos de pré-busca ao formatador do sistema HyperProp (Rodrigues & Soares, 2002).

O trabalho de instanciação do framework resultou nas implementações de estratégias de compilação e de escalonamento para o plano de pré-busca e de um módulo que estima os parâmetros para o cálculo dos tempos de busca simplificado. O formatador HyperProp dispõe de duas estratégias de compilação da pré-busca: uma que não leva em consideração o paralelismo das buscas e outra que coloca as buscas em série e impõe um retardo inicial para o início da apresentação do documento, para compensar as latências de exibição excedentes já antevistas pelo compilador de pré-busca. A heurística para o cálculo das durações de busca leva em conta, não apenas os tempos gastos na transmissão e carregamento em memória dos conteúdos, mas também o custo de criação e preparação das próprias ferramentas (principalmente os *players* JMF). Na implementação atual, o plano de pré-busca leva em consideração apenas a cadeia temporal principal, não prevendo a busca para os objetos inseridos nas cadeias auxiliares.

A execução da pré-busca é feita com um escalonador que opera sinalizando para o executor HyperProp os momentos para requisitar a instanciação (se for o caso) de uma ferramenta e a preparação do evento de apresentação correspondente ao conteúdo do objeto que deve ser buscado.

Os próximos passos do trabalho envolvem a integração do módulo que estima os parâmetros de busca com um suporte real de gerência de contexto, a implementação dos mecanismos de ajuste do plano de pré-busca em tempo de execução, a integração com mecanismos de reserva e, futuramente, a investigação do uso de mecanismos de reserva antecipada. Esses dois últimos pontos envolvem também a implementação do módulo de gerência de orquestração intra-mídia no formatador.

Ainda como trabalhos futuros, pretende-se incluir no formatador HyperProp estratégias que construam o plano de pré-busca levando em consideração a

probabilidade de navegação do usuário e que desempenhem uma gerência da ocupação dos *buffers*.

5.5
Ajuste dos tempos elásticos e escolha das alternativas

O principal recurso de adaptação disponível na implementação atual do formatador HyperProp é o ajuste das durações flexíveis, feito em tempo de compilação. Para desempenhar essa tarefa, o compilador de adaptação possui uma estratégia de ajuste que utiliza um programa externo, integrado ao formatador em um projeto de cooperação que vem sendo desenvolvido com o grupo francês de pesquisa em otimização da Universidade Blaise Pascal (Bachelet et al., 2000).

O compilador HyperProp constrói as cadeias temporais e as entrega, uma a uma, para o módulo adaptador, que repassa o pedido de adaptação para a estratégia de ajuste de tempo de compilação (Seção 4.3.2). A estratégia de ajuste então utiliza as relações impostas pelo autor e as funções de custo definidas para as durações dos eventos para obter as durações esperadas. O cálculo é feito de tal maneira que o custo total para deslocar as durações dos seus valores ideais seja minimizado.

A implementação da estratégia de adaptação é, na verdade, dividida em duas partes: um programa executável (ferramenta *solve_tension.exe*)[23], que efetivamente calcula a solução ótima para as durações, e uma classe Java, que faz a ponte entre a ferramenta externa e o formatador HyperProp. A troca dos dados entre esses dois componentes é feita através de dois arquivos, um que contém a descrição do grafo esperado pela ferramenta e outro que possui os resultados da otimização. As informações obtidas são então utilizadas pela classe Java para iniciar o valor esperado das durações dos eventos não instantâneos e até mesmo dos tempos de espera para execução das ações dos elos causais (Seção 2.3.2.1).

A ferramenta de otimização utilizada pelo formatador HyperProp resolve o problema de minimização do custo total de reduzir ou aumentar a duração dos objetos como um problema de custo mínimo em um grafo de tensões (Bachelet et al., 2000). A ferramenta espera as informações de duração nas arestas do grafo,

[23] http://www.nawouak.net/?doc=bpp_library+ch=bpp_tools

que devem ser descritas como funções de custo lineares, piece-wise e do tipo convexa. O grafo construído pela classe Java que realiza a comunicação com a ferramenta é uma simplificação de cada uma das cadeias temporais. Cada nó do grafo representa a transição na máquina de estados de um evento não interativo (normalmente, eventos de apresentação, mas outros eventos como os de atribuição também podem estar presentes). Para cada objeto de execução que deve ser exibido, apenas o evento de apresentação mais externo (Seção 5.3.1) é inserido no grafo, sendo colocada uma aresta entre as transições de início e fim da ocorrência, contendo a função de custo que descreve a duração do evento. Arestas entre transições relacionadas como condições e ações de elos causais, ou entre transições que façam parte de uma mesma restrição temporal, também são acrescentadas ao grafo.

A estratégia de ajuste dos tempos elásticos também pode receber, opcionalmente, uma função de custo para a duração da cadeia como um todo. Quando presente, essa informação resulta na inserção de uma aresta paralela ao grafo inteiro.

A utilização do algoritmo de ajuste como uma ferramenta externa e dependente de plataforma limita a portabilidade do formatador. Além disso, a necessidade de iniciar um processo externo e trocar os dados através de arquivos impõe retardos que provavelmente inviabilizarão o uso do algoritmo para ajustar as durações em tempo de execução. Por outro lado, ter a ferramenta em código executável diminui o tempo de resposta do cálculo. Como nesse primeiro estágio, o foco do trabalho era a integração das pesquisas e a colocação do formatador como uma aplicação usuária das propostas de otimização desenvolvidas no projeto de cooperação, os objetivos iniciais foram alcançados. A perda da portabilidade é minimizada com a disponibilidade do código da ferramenta e das bibliotecas utilizadas, que podem ser compiladas para outras plataformas alvo.

Como próximo passo na integração das implementações, está a eliminação dos arquivos como meio para troca dos dados entre os dois sistemas e a inclusão de uma API implementada em Java pela ferramenta de otimização para receber os dados e devolver o resultado do ajuste. Outra modificação para favorecer o ajuste em tempo de execução é procurar manter a ferramenta de otimização em memória, sem que um novo processo tenha que ser iniciado a cada vez. Finalmente, cabe

uma tentativa de implementar uma versão da ferramenta em Java e estabelecer uma comparação da eficiência com a abordagem utilizando código nativo.

Além do ajuste dos tempos elásticos, o módulo de adaptação oferece um método para realizar a seleção de uma alternativa de objeto de execução. As regras de exibição são avaliadas com base nos valores de parâmetros retornados pelo proxy de informações contextuais. Na versão atual, os parâmetros são mantidos de maneira estática, mas com a integração ao mecanismo de gerência de contexto, espera-se que seja possível realizar adaptações dinâmicas, sem que isso implique em modificações na estrutura e implementação do formatador.