Rogério Ferreira Rodrigues

FORMATAÇÃO TEMPORAL E ESPACIAL NO SISTEMA HYPERPROP

Dissertação apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Mestre em Ciência em Informática.

Orientador: Luiz Fernando Gomes Soares

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 12 de maio de 1997

Este trabalho é dedicado

Aos meus pais e à minha irmã, por tudo que tenho;

a todos aqueles que nascem e estão prontos para morrer.

AGRADECIMENTOS

- ao meu orientador, professor Luiz Fernando Gomes Soares, pelo exemplo profissional que para mim representa, pela confiança e atenção que sempre dedicou ao meu trabalho, e pelo grande amigo que é.

- a toda a minha família, pelo amor, carinho e união.

- a todos os meus amigos pelo carinho e confiança. Sem o futebol de terça, o chopp de quarta e o samba de toda hora, a vida não teria a menor graça.

- à equipe do Laboratório TeleMídia da PUC-Rio pela amizade e ajuda indispensáveis durante a elaboração desta dissertação.

- a todos os professores e funcionários do Departamento de Informática da PUC-Rio que colaboraram para a conclusão deste trabalho.

- à CAPES, CNPq e Embratel pelo apoio financeiro fornecido durante o mestrado.

RESUMO

A construção de sistemas multimídia/hipermídia exige a implementação de um elemento responsável por receber a especificação de exibição de um documento e concretizá-la na apresentação propriamente dita. Dentre as principais tarefas desse elemento, denominado formatador temporal e espacial, estão a geração e manutenção de um plano de execução, o controle da apresentação de cada um dos componentes do documento e a alocação dos recursos para a apresentação das mídias. Esta dissertação define os mecanismos de formatação temporal e espacial implementados no sistema HyperProp, assim como as interfaces do formatador com os demais elementos do sistema.

ABSTRACT

Multimedia/hypermedia systems development requires an element responsible for receiving a document presentation specification and to execute its exhibition to the user. The main tasks of this element, called temporal and spatial formatter, are the generation and the maintainance of an execution plan, the control of the presentation of the document's components, and the allocation of resources for media presentation. This thesis presents the spatial and temporal formating mechanisms implemented in the HyperProp system, besides the formatter's interfaces with other parts of the system.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Um sistema para tratamento de documentos multimídia/hipermídia pode ser dividido em três subsistemas (ou ambientes): um para autoria, outro para armazenamento e ainda outro para formatação (ou execução), espacial e temporal, dos documentos. No decorrer do texto, a palavra documento será sempre utilizada subentendendo-se um documento multimídia/hipermídia.

O ambiente de autoria deve oferecer, pelo menos, as ferramentas que permitam a criação dos documentos, através da edição de seus componentes, da especificação dos relacionamentos entre os mesmos e da descrição do comportamento que é esperado quando forem exibidos. São essas especificações, aliadas à descrição da plataforma de exibição, que servirão de entrada para o ambiente de execução, para que a apresentação, que é o resultado final desejado, ocorra.

O termo *apresentação* denota a exibição das várias mídias ao usuário. A apresentação das mídias pode ser classificada em: *apresentação dinâmica* (mídias contínuas como áudio e vídeo) e *apresentação estática* (mídias texto e imagens gráficas). Para as mídias gráficas e textuais, a apresentação implica em exibição no monitor de vídeo. No caso das mídias áudio

e vídeo, a apresentação indica exibição auditiva e visual da informação via dispositivos de entrada e saída apropriados. A apresentação de um documento multimídia/hipermídia, normalmente, é constituída por um conjunto de componentes dinâmicos e estáticos: imagens, vídeos, textos, trechos de áudio, além de outras apresentações já editadas.

É função do ambiente de armazenamento oferecer as ferramentas para gravar e recuperar os objetos multimídia, deixando transparente a localização dos mesmos, em um sistema de informação distribuído. Cabe a esses ambientes, ainda, oferecer parâmetros para negociação de uma qualidade de serviço (QoS) adequada para recuperação das diversas mídias, em especial as contínuas. Desses e de outros pontos vem o grande interesse do estudo na área de bancos de dados multimídia.

O ambiente de execução é responsável por controlar toda a apresentação dos documentos, buscando, no subsistema de armazenamento, os conteúdos de seus componentes e as descrições de exibição, feitas utilizando o subsistema de autoria. Devido ao fato dos documentos multimídia/hipermídia fazerem uso de objetos não convencionais, tais como áudio e vídeo que, ao contrário de texto e imagens estáticas, possuem características isócronas e, dependendo da aplicação, apresentam exigências de retardo máximo limitado, o ambiente de execução deve ser modelado de forma a suportar tais características. Para tanto, a idéia usual é gerar um plano de execução, baseado na especificação de exibição do documento, para servir como guia de apresentação ao ambiente de execução, permitindo ao mesmo antecipar operações (por exemplo, realizar pré-busca do conteúdo dos objetos), ou reagir em caso de alterações no comportamento esperado. Dessa forma, são três as principais tarefas do ambiente de execução: a geração e manutenção de um plano de execução, o controle da apresentação de cada um dos componentes do documento e a alocação dos recursos para a apresentação das mídias.

Uma possível forma de especificação da apresentação do documento é no formato da estrutura de dados interna do ambiente de execução, base da geração dos planos. No entanto, geralmente, essas estruturas são complexas para manipulação direta, principalmente quando os documentos possuem uma quantidade grande de componentes e relacionamentos, que escondem toda a estruturação dos mesmos. Como conseqüência, uma

linguagem de mais alto nível tem se mostrado mais adequada para utilização na autoria, sendo a construção de editores gráficos uma tendência observada. Como exemplos, podem ser citados os editores dos sistemas Firefly [BuZe92], CMIFed [RJMB93], I-HTSPN [WSSD96] e HyperProp [CMSS96, Cost96, Much96]. A tradução da abstração de mais alto nível para a estrutura interna do ambiente de execução torna-se, assim, a ponte entre o ambiente de autoria e o de execução.

Entre os principais relacionamentos utilizados na especificação da apresentação dos documentos estão as relações de sincronização. A *sincronização* trata do posicionamento dos componentes a serem exibidos no espaço, chamada sincronização espacial, e no tempo, chamada sincronização temporal [SoSC95].

A noção de *sincronização espacial* depende da mídia a ser exibida e baseia-se em operadores que definem como combinar objetos a serem apresentados em um dispositivo de saída, num dado instante do tempo. Por exemplo, a sincronização espacial de objetos contendo imagens, gráficos ou textos define como posicionar os mesmos para apresentação em uma janela da aplicação. A sincronização, nesse caso, envolve operações como mudanças de escala, corte, conversões de cores e posicionamento dentro da janela da aplicação. Para objetos contendo áudio, a sincronização espacial envolve, por exemplo, a mixagem de canais de áudio em um dispositivo de saída de som, com ajustes de ganho e tom.

A noção de *sincronização temporal* baseia-se em mecanismos que definem como os componentes de um documento a serem apresentados são escalonados nos dispositivos de saída. As relações temporais podem ser internas a um componente ou entre componentes [StNa95]. O primeiro caso, denominado *sincronização intra-mídia*, refere-se aos relacionamentos temporais entre as várias unidades de apresentação de um único objeto que contém uma mídia contínua. Por exemplo, no caso de um vídeo sendo exibido numa taxa de 30 quadros por segundo, onde cada quadro corresponde a uma unidade de apresentação do vídeo, existe uma relação temporal intra-mídia do vídeo, de que cada quadro deve ser apresentado a cada 33,3 milisegundos. O segundo caso, denominado *sincronização inter-mídia*, refere-se à sincronização entre diferentes objetos que compõem um documento,

relacionando unidades de apresentação dos segmentos de mídia dos mesmos. O exemplo de sincronização inter-mídia freqüentemente citado na literatura é o “lip-sync” entre áudio e vídeo.

Em sistemas multimídia/hipermídia com flexibilidade para descrição da apresentação dos documentos, a especificação dos relacionamentos anteriormente citados é feita, pelo autor, de forma independente da plataforma de exibição. Cabe ao ambiente de execução receber essa especificação, em conjunto com a descrição de uma plataforma de exibição, a fim de garantir que a determinação do autor, sempre que possível, seja respeitada. Como a plataforma de exibição só é descrita no momento da apresentação, é importante que existam mecanismos para verificar a consistência espacial dos relacionamentos, por exemplo, se existe dispositivo com suporte para som, se não há conflitos por recursos etc. Além disso, a edição do documento, normalmente, será feita por partes, principalmente quando o mesmo possuir uma quantidade grande de objetos. Pode ser que cada uma das partes, independentemente, esteja consistente, mas, ao serem reunidas, haja alguma inconsistência temporal que requisite ajustes. No caso da impossibilidade de que tais ajustes sejam feitos, é também importante que os mecanismos de verificação acusem esse tipo de inconsistência temporal.

Apenas o ajuste feito antes da exibição do documento, preparando para que todas as especificações sejam atendidas, pode não ser suficiente. Isso ocorre porque fatores que não são possíveis de serem previstos podem intervir na apresentação, tais como interação do usuário, atrasos na transmissão do conteúdo dos objetos na rede, limitações do ambiente de armazenamento, limitações do sistema operacional, entre outros. Para que o sistema seja capaz de reagir a tais condições, procurando manter a especificação original de apresentação do documento, é preciso que o ambiente de execução disponha, ainda, de mecanismos de ajustes em tempo de exibição do documento.

A formatação temporal e espacial consiste em dispor os objetos de um documento no tempo e no espaço, efetuando ajustes necessários, a fim de satisfazer as especificações de apresentação do autor para uma determinada plataforma de exibição. Dessa forma, é

importante que um sistema multimídia/hipermídia possua mecanismos que permitam executar esta formatação de maneira automática.

Uma vez que todas as funções do ambiente de execução giram em torno da formatação temporal e espacial dos documentos, a fim de que o objetivo final (apresentação dos documentos respeitando as especificações do autor) seja alcançado, esse ambiente é também denominado formatador temporal e espacial, ou simplesmente formatador.

1.2 Objetivos

Este trabalho faz parte do desenvolvimento do sistema HyperProp [SoCC93], cujo principal objetivo é a criação de um ambiente para dar suporte à construção de aplicações hipermídia. HyperProp provê não apenas um modelo conceitual de dados hipermídia, o Modelo de Contextos Aninhados (NCM - *Nested Context Model*) [SoCR95, SoSo97], como uma arquitetura aberta, cujo modelo de interface separa os componentes de dados e de exibição dos objetos, permitindo a construção de interfaces independentes da plataforma final de apresentação. Essa separação visa desenvolver um sistema hipermídia que não seja uma aplicação auto-contida, proporcionando facilidades para intercâmbio de informações, interoperabilidade e reuso de código entre aplicações.

A hierarquia de classes do NCM dispõe de funcionalidades, não só para tratar os documentos hipermídia em sua forma estrutural, como também para permitir ao usuário acesso, visualização, manipulação e navegação através da estrutura dos documentos. Esses aspectos tratados pelo modelo estão em conformidade com o Modelo Dexter [HaSc90], e são descritos em detalhes em [SoCR95, SoSo97].

O sistema HyperProp dispõe de ferramentas gráficas, já implementadas, que dão suporte à fase de autoria e orientação durante a navegação através dos documentos hipermídia, seguindo o NCM, estando as mesmas descritas em [Cost96, Much96]. No entanto, como mencionado anteriormente, é preciso que haja no sistema um módulo (o formatador temporal e espacial), capaz de receber a descrição de apresentação de um documento e controlar a sua exibição, garantindo que a especificação do autor seja cumprida.

O objetivo desta dissertação é implementar o ambiente responsável pela formatação temporal e espacial do sistema HyperProp. Uma vez que a implementação de um formatador, atendendo a todos os requisitos concluídos como necessários, é de uma complexidade que foge ao escopo deste trabalho, foram limitadas as funções a serem implementadas. Dentre os elementos da formatação abordados na dissertação estão:

- a implementação do módulo responsável por converter a especificação de apresentação do documento em uma estrutura de dados capaz de gerar o plano de execução (módulo compilador);

- a modelagem da estrutura de dados base para o plano de execução do documento, possibilitando a formatação temporal e espacial em tempo de apresentação;

- a implementação do módulo responsável pela formatação temporal e espacial em tempo de apresentação (módulo executor), sem os algoritmos de ajuste em tempo de execução, de verificação de inconsistências e de pré-carregamento do conteúdo dos objetos;

- a definição da interface do formatador com o ambiente de armazenamento, assim como a definição da interface do módulo executor com os elementos responsáveis por controlar a exibição de cada um dos objetos do documento quando este for apresentado.

A descrição dos ambientes de autoria e armazenamento não é objeto da dissertação, sendo mencionada apenas no que diz respeito às interfaces com o formatador. Também não faz parte deste trabalho a implementação dos elementos controladores para exibição dos componentes dos documentos.

1.3 Organização da Dissertação

A dissertação está organizada de forma a apresentar, no segundo capítulo, uma caracterização dos formatadores temporais e espaciais de sistemas multimídia/hipermídia, descrevendo os principais requisitos e propondo uma arquitetura genérica para estruturação dos mesmos. Dentro da arquitetura proposta, cada um dos componentes do formatador é

descrito segundo suas funções básicas. São também salientadas as informações que devem ser trocadas entre os diversos componentes do formatador.

No terceiro capítulo é especificado o formatador temporal e espacial do sistema HyperProp. São apresentados: o modelo de dados de entrada, a arquitetura interna do formatador e o modelo de dados interno utilizado pelo formatador para controlar a apresentação dos documentos. Além disso, a interface entre o formatador e os demais elementos do sistema é especificada de maneira detalhada.

O quarto capítulo descreve a implementação do formatador temporal e espacial do sistema HyperProp, apresentando a estrutura modular do formatador, a hierarquia de classes, a descrição dos principais métodos implementados e os diagramas de relacionamento entre as classes.

O quinto capítulo é dedicado à comparação com trabalhos relacionados. São descritos alguns sistemas que apresentam a formatação temporal e espacial automática para apresentação dos documentos. Neste capítulo, os formatadores desses sistemas são classificados de acordo com as características apresentadas no segundo capítulo e comparados com o formatador do sistema HyperProp.

O sexto e último capítulo é reservado às conclusões finais. Também neste capítulo, são citadas as principais contribuições da dissertação e feitas propostas para trabalhos futuros.

Capítulo 2

Formatadores Temporais e Espaciais em Sistemas Multimídia/Hipermídia

Este capítulo apresenta os principais requisitos que um sistema para apresentação de documentos multimídia/hipermídia com restrições temporais deve satisfazer. Dentre esses requisitos, são salientados os que estão relacionados com o ambiente de execução dos sistemas e, por conseguinte, com a formatação temporal e espacial automática dos documentos. Durante a apresentação desses requisitos, são introduzidos alguns termos relevantes em toda a dissertação.

A partir dos requisitos apresentados, são descritas algumas características que podem, ou não, estar presentes nos formatadores de sistemas multimídia/hipermídia. Tais características serão utilizadas, no Capítulo 5, como base de comparação entre exemplos de sistemas multimídia/hipermídia existentes que apresentam algum tipo de formatação temporal e espacial automática dos documentos.

Ainda neste capítulo, é descrita uma proposta de arquitetura genérica de um ambiente de execução, sendo cada um dos elementos presentes na arquitetura apresentados segundo suas respectivas funções básicas.

2.1 Requisitos de Formatação de Documentos Hipermídia com Restrições Temporais

É interessante que sistemas para apresentação de documentos multimídia/hipermídia com restrições temporais, além de possuírem um algoritmo de formatação temporal e espacial automática dos documentos, introduzam algumas facilidades desejáveis, tais como:

- suportar, para cada segmento de mídia, um conjunto extenso de capacidades (por exemplo: variabilidade de duração de apresentação, diferentes alternativas de exibição, mudanças de comportamento durante a exibição etc.);

- permitir a definição da qualidade de serviço exigida por cada segmento de mídia (por exemplo: variação de retardo intra-mídia (jitter), banda passante necessária etc.);

- permitir a definição do ambiente de apresentação (por exemplo: retardo na rede, tipos de mídia que a plataforma de exibição suporta etc.);

- permitir representar, explicitamente, relações temporais com tempos não determinísticos (por exemplo: “exiba o áudio, que pode durar entre 20 e 30 segundos, paralelo ao vídeo, que pode durar entre 25 e 28 segundos, e 10 segundos depois, com uma tolerância de 2 segundos, exiba o texto”);

- permitir representar, explicitamente, relações espaciais entre segmentos de mídia em um dado instante de tempo;

- permitir uma definição estruturada (hierárquica ou não) do documento;

- possuir um algoritmo de formatação temporal versátil, que leve em conta os não determinismos e permita a correção da apresentação no caso de ocorrência de eventos não previsíveis (por exemplo: atrasos na rede, diretivas do usuário alterando a taxa de apresentação de um documento etc.); e

- possuir um algoritmo de formatação que tire proveito das definições das QoS exigidas por cada segmento e das características da plataforma de exibição (por exemplo: realização de pré-busca do conteúdo dos objetos).

Os dois últimos pontos estão diretamente relacionados com o assunto tratado nesta dissertação, estando os demais requisitos relacionados às questões de especificação de apresentação dos documentos, ou seja, ao ambiente de autoria. No entanto, vários dos requisitos para a formatação temporal e espacial automática dos documentos dependem de requisitos que sejam satisfeitos pelo ambiente de autoria do sistema.

A maioria dos sistemas existentes para apresentação de documentos encontra-se no estágio de pré-formatação. Tais sistemas praticamente exigem que o autor crie os layouts temporais de apresentação manualmente, posicionando os segmentos de mídia em uma linha de tempo do documento [DrGr93, Macr89], constituindo-se em um processo tedioso e sujeito a erros. Outros métodos manuais incluem as linguagens de scripts temporais [FiTD87] e árvores de relações temporais [LiGh90]. Já vem de algum tempo a necessidade da geração automática dos layouts temporais, que só há alguns anos começou a ter soluções parciais em protótipos de laboratório [BuZe93a].

Em [BuZe93a] é feita uma analogia entre a evolução da formatação de documentos convencionais (apenas texto e imagens estáticas) e a formatação de documentos multimídia/hipermídia, que incorporam mídias contínuas, e, conseqüentemente, a noção de tempo. Pode ser observado que, assim como nos documentos multimídia/hipermídia, também no caso dos documentos convencionais, inicialmente os layouts espaciais eram fixados manualmente pelo autor, com o uso de uma máquina de escrever. Com o advento dos processadores de texto, como T_EX e Word, diretivas de alinhamento dos textos e imagens gráficas (tais como: centralizar, justificar etc.) passaram a permitir a criação dos layouts espaciais de apresentação de maneira automática, facilitando não só o trabalho de edição como de manutenção dos documentos. Dessa analogia vem a sugestão de que, também nos sistemas para tratamento de documentos multimídia/hipermídia, a edição dos relacionamentos temporais e espaciais para exibição dos objetos seja feita através de diretivas, permitindo a formatação automática da apresentação. Uma discussão detalhada

sobre as diretivas para edição de sincronismos na apresentação de documentos multimídia/hipermídia pode ser encontrada em [Cost96].

É função do formatador temporal e espacial nos sistemas multimídia/hipermídia a geração automática do layout de apresentação, através da construção de um plano de execução, baseado nas especificações de apresentação do documento e na descrição da plataforma de exibição. O objetivo do plano é orientar o próprio formatador no controle da apresentação dos documentos. Tal construção deve ser feita utilizando algum algoritmo para formatação automática que, preferencialmente, atenda aos requisitos anteriormente descritos nesta seção.

O plano de execução é um cronograma de eventos, onde, a cada início ou fim de evento, um instante esperado de ocorrência é associado. No contexto deste trabalho, um *evento* pode ser a exibição, *evento de exibição*, ou a seleção, *evento de seleção*, de um conjunto não vazio de unidades de informação, ou ainda a mudança de um atributo de um objeto componente do documento, por exemplo, o volume de exibição de um áudio, chamado *evento de atribuição*. A noção exata do que se constitui uma unidade de informação é parte da definição do tipo de mídia do objeto componente do documento. Por exemplo, uma unidade de informação de um objeto vídeo pode ser um quadro, enquanto uma unidade de informação de um objeto texto pode ser um caracter, ou uma palavra. Eventos com duração atômica são denominados *pontos de sincronização*.

O instante de ocorrência de um ponto de sincronização pode ser previsível, como o fim da exibição de um quadro de vídeo, ou imprevisível, como no caso da seleção, por parte do usuário, de um botão. Um evento é previsível quando seu início e fim são previsíveis. Obviamente, apenas os pontos de sincronização previsíveis podem, antes do início da apresentação do documento, ter um instante de ocorrência associado a si. O encadeamento de pontos de sincronização é denominado uma *cadeia temporal*. Uma cadeia temporal é dita *parcial* quando o seu primeiro evento é imprevisível. Um plano de execução de um documento é construído pela concatenação de uma ou mais cadeias temporais parciais, onde todos os pontos de sincronização previsíveis são associados a um tempo de ocorrência esperado, relativo ao ponto de sincronização anterior.

Ao formatador temporal e espacial cabe, além de construir o plano de execução, controlar a apresentação de cada um dos componentes do documento, que realimentam o plano com os eventos ocorridos, a fim de que o mesmo possa ser mantido atualizado durante a exibição.

2.2 Características dos Formatadores Temporais e Espaciais

Usando e estendendo a classificação apresentada em [BuZe93b], um formatador temporal e espacial pode ser analisado segundo seis parâmetros, citados a seguir e detalhados nas seções subseqüentes:

- momento em que se constrói o plano de execução;
- flexibilidade na construção do plano de execução;
- relacionamentos temporais suportados;
- mudanças de comportamento admitidas;
- inconsistências que são analisadas; e
- como reage a variações imprevisíveis.

2.2.1 Momento de Construção do Plano

Um formatador pode apresentar três fases: a pré-compilação, a compilação e a execução. Existem exemplos de sistemas que apresentam apenas a fase de compilação (Temporal Glue [HaSR92]), as fases de pré-compilação e compilação (I-HTSPN [WSSD96]), apenas a fase de execução (Trellis [StFu90]), as fases de compilação e execução (Firefly [BuZe92], CMIFed [HaRB93]), e as três fases juntas (HyperProp [SoCC93]). O que os diferencia é o momento em que o plano de execução é construído.

Quando o plano de execução é construído antes da apresentação, e não sofre alteração durante a exibição do documento, existem três possibilidades: ou o sistema possui apenas a fase de pré-compilação, ou o sistema possui apenas a fase de compilação, ou o sistema possui as fases de pré-compilação e compilação.

No caso em que a construção do plano de execução é feita apenas durante a apresentação do documento e o plano é mantido atualizado conforme se dá a execução da exibição, o sistema apresenta somente a fase de execução.

Quando a construção do plano é feita antes da apresentação e mantida atualizada durante a exibição do documento, existem três possibilidades: ou o sistema possui as fases de pré-compilação e execução, ou o sistema possui as fases de compilação e execução, ou o sistema possui as fases de pré-compilação, compilação e execução.

A Tabela 1 resume as combinações possíveis anteriormente descritas. A Seção 2.3 apresenta uma discussão aprofundada sobre as três fases.

Momento de Construção do Plano de Execução	Fases			Exemplos de Sistemas
	Pré-Compilação	Compilação	Execução	
Antes da Apresentação	•			
		•		Temporal Glue
	•	•		I-HTSPN
Durante a Apresentação			•	Trellis
Antes e Durante a Apresentação	•		•	
		•	•	Firefly, CMIFed
	•	•	•	HyperProp

Tabela 1 - Fases de um ambiente de execução

2.2.2 Flexibilidade

Independente de quando construído, o plano de execução pode ter, ou não, flexibilidade no cálculo para o momento esperado de ocorrência dos pontos de sincronização. Tal flexibilidade está intimamente relacionada com a flexibilidade da especificação realizada no ambiente de autoria.

Em sistemas sem flexibilidade, um valor fixo para duração de eventos é especificado pelo autor e é utilizado na confecção do plano. Já em sistemas com flexibilidade, tempos de ocorrência associados aos pontos de sincronização podem ser ajustados dentro de uma faixa de valores aceitáveis, determinada pelo autor quando da especificação da duração de cada evento. Por exemplo, para cada evento o autor pode determinar um tempo mínimo e um

tempo máximo para sua duração, que pode assumir valores discretos ou contínuos dentro desse intervalo. Além da faixa de valores válidos, pode ainda ser possível a especificação do valor ótimo, isto é, aquele que caracteriza a melhor qualidade de apresentação do objeto. O algoritmo de formatação, obviamente, tentará sempre exibir a mídia na duração ótima, no entanto, podem ocorrer situações em que isso não seja possível, cabendo ao formatador buscar uma outra duração dentro do intervalo. Suponha o caso ilustrado na Figura 1, em que o vídeo deve ser exibido em paralelo com o áudio (iniciar e terminar juntos). Nesse exemplo, é impossível que os dois elementos sejam apresentados com as respectivas durações ótimas, sendo necessária a escolha de um novo valor para duração de pelo menos um dos eventos. Tal escolha pode ser melhorada, quando direcionada por funções de flexibilidade que, por exemplo, indiquem um custo para aumentar ou diminuir a duração do evento. No exemplo, as funções de flexibilidade poderiam ser utilizadas para determinar se a degradação da apresentação seria menor ao aumentar o tempo de exibição do vídeo (alongamento - *stretching*), ou ao diminuir o tempo de exibição do áudio (encolhimento - *shrinking*), ou ainda por uma combinação das duas opções. Nos sistemas que não apresentam flexibilidade, os eventos são não ajustáveis, ou seja, o valor do tempo mínimo para o intervalo de duração é idêntico ao valor do tempo máximo.

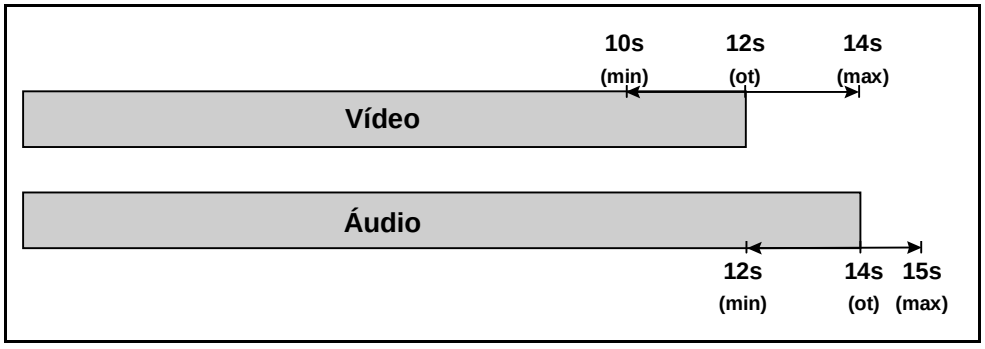


Figura 1 - Exemplo de flexibilidade para exibição dos segmentos de mídia

Além da flexibilidade no ajuste da duração de ocorrência dos eventos, existem ainda mais duas formas de flexibilidade que os formatadores podem apresentar. Uma delas é poder selecionar uma dentre várias alternativas de apresentação, por exemplo, um áudio que possa, eventualmente, ser exibido como um texto. A outra forma de flexibilidade é ignorar especificações de relacionamentos que sejam classificados como opcionais pelo autor.

Como nos sistemas analisados neste trabalho nenhum implementa essa última opção, ela não será considerada nas futuras comparações.

Como foi dito anteriormente, a flexibilidade dos formatadores está intimamente relacionada com a flexibilidade oferecida pelo ambiente de autoria, pois a atitude de ignorar relacionamentos opcionais, por exemplo, só faz sentido em sistemas que permitam ao autor classificar os relacionamentos como obrigatórios ou não. Ou ainda, ajustar a duração de ocorrência dos valores dentro de uma faixa válida só é possível em sistemas que permitam especificar tal faixa, para cada um dos eventos. Por outro lado, um sistema que apresente flexibilidade na especificação dos relacionamentos temporais pode, por simplificação, implementar um formatador temporal e espacial que não a utilize.

2.2.3 Relacionamentos Temporais

Quanto aos relacionamentos existentes entre os componentes de um documento, o formatador temporal e espacial pode dar suporte apenas a relacionamentos com eventos previsíveis, ou a relacionamentos que possuam, também, eventos imprevisíveis, tais como: interação com usuário na seleção de um botão, componentes com duração não determinística etc.

Evidentemente, o suporte a relcionamentos temporais com eventos imprevisíveis só faz sentido em formatadores que apresentem a fase de execução, pois o cálculo do tempo de ocorrência do evento não pode ser feito antes da apresentação do documento.

2.2.4 Mudanças de Comportamento

O formatador temporal e espacial, ao construir o plano de execução, pode dar suporte a mudanças de comportamento especificadas na fase de autoria, tais como aumentar o volume do áudio em um dado instante, acelerar o vídeo, aplicar um zoom em uma imagem etc. Essas mudanças de comportamento podem ser levadas em consideração tanto nas fases de pré-compilação e compilação, bem como na fase de execução.

Além das mudanças de comportamento especificadas na fase de autoria, o formatador pode suportar as chamadas meta-diretivas do usuário em tempo de exibição. *Meta-diretivas* são

mudanças de comportamento provenientes da interação dinâmica com o usuário no controle da taxa de apresentação do documento como um todo, por exemplo, dar uma pausa na exibição de um documento, apresentá-lo de forma acelerada, apresentá-lo de trás para frente etc. Normalmente, sistemas que apresentam esse tipo de característica oferecem interfaces para controle da apresentação semelhantes a de um aparelho de CD. A Figura 2 mostra a interface para meta-diretivas que faz parte do ambiente de execução do sistema CMIFed [RJMB93], descrito em maiores detalhes no Capítulo 5.

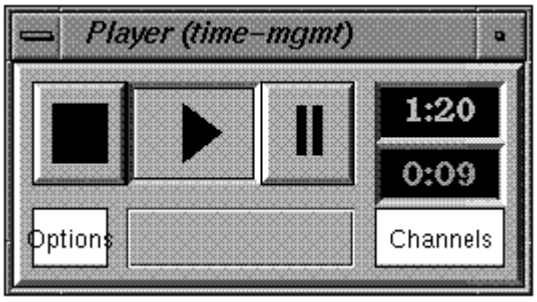


Figura 2 - Exemplo de suporte a meta-diretivas

2.2.5 Verificação de Inconsistências

Durante a geração do plano de execução, o formatador pode sinalizar inconsistências temporais e espaciais.

Inconsistências temporais correspondem à detecção de falhas na especificação do autor quanto à disposição temporal na exibição do documento, que resulta em uma apresentação impossível de ser realizada. Um exemplo de inconsistência desse tipo é ilustrado na Figura 3, onde é especificado em um sistema com flexibilidade, a exibição de dois objetos, A e B, em paralelo (iniciando e terminando juntos), onde o tempo mínimo para apresentação de B é maior que o tempo máximo para apresentação de A.

Inconsistências espaciais correspondem à detecção de falhas na apresentação das mídias nos respectivos recursos, em um dado instante do tempo. Existem duas formas de inconsistência desse tipo: inconsistência de plataforma, e conflito por recursos. Um exemplo de inconsistência na plataforma de exibição seria a ausência de suporte a áudio, numa determinada estação, para um documento que exige tal dispositivo. Conflitos por recursos

ocorrem quando, num dado instante, dois objetos tentam utilizar o mesmo dispositivo de saída, sem que isso seja possível (por exemplo: dois áudios que utilizam o dispositivo de som, sem que haja suporte à mixagem).

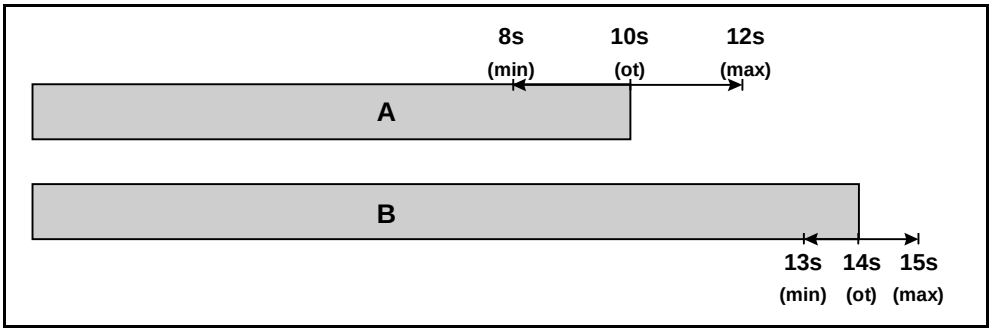


Figura 3 - Exemplo de inconsistência temporal

Note que a verificação da consistência temporal e espacial antes da apresentação do documento (fases de pré-compilação e compilação), não garante que durante a exibição não surjam inconsistências. No caso do sistema permitir eventos imprevisíveis, a ocorrência de tais eventos durante a apresentação do documento pode causar uma inconsistência espacial no plano de execução, ou mesmo uma inconsistência temporal. Sendo assim, idealmente, os ambientes de execução devem possuir, também, a verificação das inconsistências em tempo de apresentação (fase de execução).

2.2.6 Alterações Imprevisíveis

Finalmente, os formatadores temporais e espaciais podem suportar, ou não, indicações de variações na taxa de apresentação esperada. Exemplos de variações são atrasos ou adiantamentos de exibição dos objetos, que podem ser causados pela transmissão do conteúdo das mídias em redes que possuem retardos aleatórios, por limitações dos dispositivos de exibição e armazenamento, ou mesmo por limitações do sistema operacional.

Os formatadores que suportam alterações imprevisíveis devem reagir recalculando os tempos esperados para os pontos de sincronização e, eventualmente, corrigindo as taxas de apresentação dos objetos, a fim de satisfazer a especificação original do autor.

Evidentemente, só faz sentido esse tipo de característica estar presente em formatadores que possuam a fase de execução.

2.2.7 Considerações Gerais sobre as Características dos Formatadores

Das características apresentadas nas seções anteriores, algumas conclusões podem ser tiradas. Em primeiro lugar, quanto maior a flexibilidade de um formatador, menor são as chances de um plano de execução apresentar uma inconsistência temporal ou espacial. Em segundo lugar, ao suportar relacionamentos temporais com eventos imprevisíveis e mudanças de comportamento especificadas na fase de autoria, aumenta-se o poder de expressão dos documentos gerados. Em terceiro lugar, ao suportar mudanças de comportamento com meta-diretivas, aumenta-se o controle do usuário na apresentação dos documentos. Em quarto lugar, ao suportar alterações imprevisíveis nas taxas de apresentação dos objetos, assim como ao realizar a verificação de inconsistências temporais e espaciais, aumenta-se a qualidade do plano de execução e, conseqüentemente, da apresentação do documento. Finalmente, em quinto lugar, o momento em que a construção do plano de execução é feita afeta tanto a qualidade, como o poder de expressão da apresentação.

Quanto à questão do momento da construção do plano de execução, quando a mesma ocorre apenas nas fases de pré-compilação ou compilação do formatador, este pode tirar total vantagem da flexibilidade no ambiente de autoria para evitar inconsistências temporais e espaciais previsíveis, ajustando a duração dos eventos. Quando não for possível efetuar tal ajuste, os erros podem ser sinalizados para o autor, antes que a apresentação seja iniciada. No entanto, esses formatadores não suportam qualquer tratamento de comportamentos não previsíveis. Já formatadores que apresentam apenas a construção do plano em fase de execução, não conseguem detectar as inconsistências até que as mesmas ocorram. Por outro lado, apresentam como principal vantagem o fato de suportarem todos os tipos de comportamentos não previsíveis (eventos imprevisíveis, meta-diretivas, variações nos retardos) trazendo para a apresentação dos documentos todos os benefícios vistos como inerentes a essas características. Um formatador híbrido, que apresente tanto as fases de pré-compilação ou compilação, como a fase de execução, reune todas as vantagens, destinando a um módulo compilador (ou pré-compilador) o tratamento de todo o

comportamento previsível de apresentação, e a um módulo executor o tratamento de todo o comportamento imprevisível de apresentação do documento.

A Tabela 2 apresenta um resumo das características descritas neste capítulo. Para cada fase do formatador são dispostas as características que podem estar presentes e a forma que cada uma delas pode ser implementada.

Fases	Características	Possibilidades de Implementação
Pré-Compilação	Flexibilidade	inflexível, ajustável, seleciona alternativas
	Mudanças de Comportamento	especificações de autoria
	Inconsistência	temporal, plataforma, conflito
Compilação	Flexibilidade	inflexível, ajustável, seleciona alternativas
	Mudanças de Comportamento	especificações de autoria
	Inconsistência	temporal, plataforma, conflito
Execução	Flexibilidade	inflexível, ajustável, seleciona alternativas
	Relacionamentos Temporais	previsíveis, imprevisíveis
	Mudanças de Comportamento	especificações de autoria, meta-diretivas
	Inconsistência	temporal, plataforma, conflito
	Variação Imprevisível	não suporta, suporta

Tabela 2 - Características de formatadores temporais e espaciais

Em geral, os formatadores temporais e espaciais dos sistemas existentes suportam apenas um conjunto das características descritas neste capítulo. O Capítulo 5 baseia-se nestas características e apresenta uma comparação entre alguns formatadores de sistemas multimídia/hipermídia, incluindo o próprio sistema HyperProp.

2.3 Arquitetura Genérica de um Ambiente de Execução

Em [BuZe93b], Buchanan e Zellweger descrevem uma arquitetura genérica para formatadores temporais e espaciais. Esta seção estende a arquitetura pelas autoras introduzida, salientando as diversas interfaces internas e externas, necessárias à discussão dos demais capítulos da dissertação. A Figura 4 ilustra a arquitetura estendida, onde podem

ser percebidos quatro elementos básicos: o pré-compilador, o compilador, o executor e os controladores de exibição. A demais seções deste capítulo são destinadas a definir cada um desses elementos.

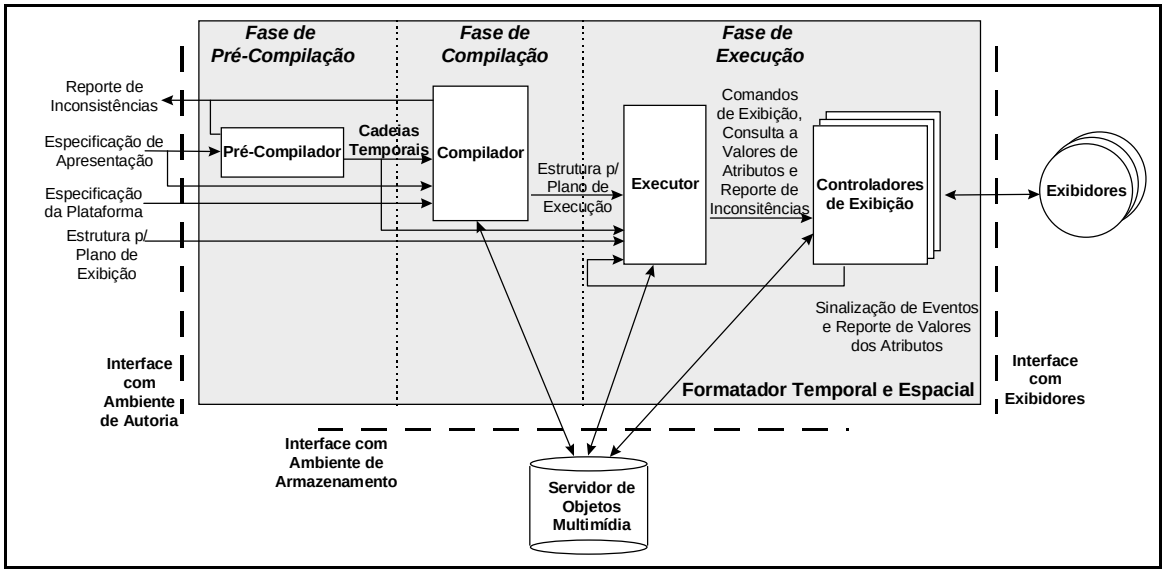


Figura 4 - Arquitetura genérica de um formatador

2.3.1 Pré-Compilador

Embora realize funções de formatação temporal e espacial, correspondentes ao ambiente de execução, o pré-compilador (que não existe na arquitetura proposta em [BuZe93b]) é um elemento de suporte ao ambiente de autoria. Sua principal função é verificar a consistência temporal e espacial para cada uma das cadeias temporais parciais, independentemente, dada uma especificação de apresentação. A idéia é que a análise seja feita de forma dinâmica, efetuando uma compilação incremental, conforme o autor cria os relacionamentos, alertando para possíveis erros de especificação.

As cadeias temporais parciais geradas pelo pré-compilador para verificar as inconsistências podem ser descartadas após tal análise. Porém, é mais interessante que essas cadeias sejam construídas em uma estrutura capaz de ser interpretada pelo compilador, ou mesmo pelo executor do sistema, a fim de reduzir o tempo de processamento desses módulos.

Cabe observar que, em ambientes de autoria onde a especificação da apresentação é feita de forma independente da plataforma de exibição, não faz sentido o pré-compilador executar testes de inconsistência de plataforma.

2.3.2 Compilador

O compilador é responsável por receber a especificação completa de apresentação do documento e gerar a estrutura de dados através da qual o executor será capaz de construir o plano de execução. Existem duas possibilidades para a entrada de dados do compilador. Ele pode receber do pré-compilador o conjunto de cadeias temporais parciais independentes, ou receber a especificação temporal e espacial do documento inteiro, diretamente do subsistema de autoria, sem passar pelo pré-compilador. Em ambas as opções, o compilador deve receber a descrição da estação onde o documento será exibido, para que possam ser feitos os testes de consistência de plataforma.

Cabe ao compilador entregar ao executor todas as informações de flexibilidade recebidas do subsistema de autoria, na estrutura de dados por ele gerada. Para cada uma das cadeias temporais parciais, o compilador deve calcular o instante de ocorrência esperado para os pontos de sincronização previsíveis (se tal tarefa já não tiver sido desempenhada pelo pré-compilador), podendo utilizar algum algoritmo de ajuste nos cálculos. É nesse momento que os testes de consistência devem ser feitos, já sendo levado em consideração a descrição da plataforma de exibição.

É interessante observar que, no caso do documento possuir todo o seu comportamento de apresentação previsível, tanto o pré-compilador como o compilador geram o mesmo resultado, pois haverá uma única cadeia temporal. Neste caso, o plano de execução pode ser gerado em tempo de pré-compilação.

2.3.3 Executor

O executor é o elemento destinado às questões que só podem ser resolvidas no momento da apresentação dos documentos e que, conseqüentemente, não puderam ser solucionadas pelo pré-compilador e pelo compilador. Ao executor cabe instanciar os controladores de exibição e reagir, conforme a especificação do autor, quando da ocorrência de sinalizações

de eventos informadas pelos controladores. Também é função do executor reagir a meta-diretivas sinalizadas pelo usuário em tempo de exibição.

É através das sinalizações recebidas dos controladores que o executor pode verificar a ocorrência de alterações na taxa originalmente prevista para apresentação. Com o cálculo dos valores dessas variações, e de posse de informações de QoS de toda a plataforma (estação e rede) de tratamento do documento de exibição, o executor pode atuar, através de comandos, ajustando em tempo de execução as durações dos eventos, para que a especificação do autor seja respeitada.

Como mencionado anteriormente, a especificação de apresentação pode ser feita diretamente na estrutura interna de dados do executor, ou através de uma interface de mais alto nível, traduzida pelo compilador, ou opcionalmente pelo pré-compilador, para a sua estrutura interna. Cabe ressaltar que em um sistema que só possui a fase de execução, o executor recebe diretamente do ambiente de autoria a estrutura de dados através da qual será capaz de gerar o plano.

2.3.4 Controladores de Exibição

Os controladores de exibição são responsáveis pela interface do formatador com os exibidores (editores) sendo utilizados para apresentar os objetos. Para cada objeto pode, ou não, haver uma instância de controlador distinta. Por exemplo, um nó de texto A sendo exibido por um editor Word utiliza uma instância de controlador. Um nó de texto B sendo exibido também por um editor Word pode utilizar a mesma instância, ou definir uma outra instância de controlador, e assim sucessivamente. Para exibidores diferentes (texto A exibido pelo Word e texto B exibido pelo Write) certamente haverá instâncias de controladores distintas. No caso de documentos com composições, deve haver uma instância de controlador para cada composição de objetos do documento sendo exibida (composições em documentos multimídia/hipermídia são abordadas no Capítulo 3).

Dos objetos (exibidores), os controladores recebem as sinalizações de ocorrência de eventos, tais como interação do usuário ou apresentação de uma unidade de informação de um segmento de mídia, e as reporta ao executor. Do executor, por sua vez, os

controladores recebem comandos de apresentação, tais como preparar, iniciar, acelerar, retardar, parar ou abortar a exibição do conteúdo de um objeto, ou ainda modificar o valor de um atributo do objeto. Além disso, algumas especificações de apresentação podem depender da consulta a valores de atributos dos objetos, cabendo aos controladores respondê-las.

Além das interfaces com os exibidores e com o executor, fica a cargo dos controladores requisitar os conteúdos de dados dos objetos a serem apresentados ao subsistema de armazenamento, negociando parâmetros de QoS para entrega dos mesmos.

Evidentemente, em formatadores que não possuem a fase de execução, os elementos controladores de exibição estão presentes de forma passiva, ou seja, eles apenas recebem as mensagens com sinalização de ações a serem executadas do compilador, sem retornar qualquer sinalização de ocorrência de eventos.

Capítulo 3

Formatação Temporal e Espacial no Sistema HyperProp

Este capítulo destina-se à especificação do formatador temporal e espacial para o sistema HyperProp. Em primeiro lugar, são definidos os elementos do modelo de dados de entrada para o formatador, o Modelo de Contextos Aninhados (NCM - *Nested Context Model*), relevantes para compreensão da descrição dos mecanismos de formatação.

Em seguida, é especificada a arquitetura do formatador, sendo descritas as fases de pré-compilação, compilação e execução, onde é dada uma maior ênfase aos módulos compilador e executor do formatador, por terem sido os elementos implementados neste trabalho.

Ainda neste capítulo, é apresentado o modelo de dados da estrutura interna para controle da apresentação dos documentos, e são caracterizadas e definidas duas interfaces: entre os módulos compilador e executor e o ambiente de armazenamento do sistema; e entre o executor e os controladores de exibição. A interface entre os controladores de exibição e o ambiente de armazenamento é apenas caracterizada, sendo sua especificação deixada como trabalho futuro, uma vez que a implementação desses controladores não é objeto desta dissertação.

3.1 Modelo de Dados de Entrada

Uma vez que a entrada de dados do formatador temporal e espacial do sistema HyperProp baseia-se em especificações de apresentação de documentos modelados segundo o NCM, é importante que os conceitos do modelo, relevantes para compreensão do funcionamento do formatador, sejam apresentados. Esta seção resume os conceitos definidos em [SoSo97], onde é apresentada uma descrição completa do Modelo de Contextos Aninhados, revisão de uma versão anterior do modelo descrita em [SoCR95].

3.1.1 Nós, Âncoras e Perspectivas

Documentos hipermídia, no NCM, são definidos com base nos conceitos usuais de nós e elos. *Nós* são fragmentos de informação e *elos* são usados para a interconexão de nós que mantêm alguma relação. Todo nó possui, entre outros atributos, um *conteúdo*, que é formado por um conjunto de unidades de informação.

O modelo distingue duas classes básicas de nós, chamados de nós terminais e nós de composição, sendo estes últimos o ponto central do modelo. Intuitivamente, um *nó terminal* (ou nó de conteúdo) contém dados cuja estrutura interna é dependente da aplicação (se constituem nos nós hipermídia tradicionais). A classe de nós terminais pode ser especializada em outras classes (texto, gráfico, áudio, vídeo etc.), conforme requerido pelas aplicações. A noção exata do que constitui uma *unidade de informação* é parte da definição do nó terminal. Por exemplo, uma unidade de informação de um nó vídeo pode ser um quadro, enquanto uma unidade de informação de um nó texto pode ser um caracter, ou uma palavra. Um nó de *composição* possui como conteúdo (unidades de informação) uma coleção de elos e nós, de conteúdo ou de composição, recursivamente. O nó de composição pode ser especializado em outras classes. Dentre elas, relevante para a definição de documentos, está a classe nó de contexto. Um nó de *contexto* é um nó de composição tal que seu conteúdo possui apenas elos, nós terminais e nós de contextos, sem que um mesmo componente apareça repetidamente na composição. No NCM, os documentos são sempre definidos como nós de contexto.

Todo nó possui um conjunto de âncoras. Uma *âncora* tem como atributo uma *região*, que é um conjunto de unidades de informação marcadas. Qualquer subconjunto de unidades de informação de um nó pode ser marcado. Tem-se assim que a definição exata do conteúdo de um nó e da região de uma âncora depende da classe do nó, exceto que o modelo requer que o símbolo especial λ seja uma região válida, representando a marcação de todo o conteúdo do nó. Intuitivamente, uma âncora define um segmento dentro do conteúdo de um nó. O conjunto de âncoras age como uma interface externa do nó, no sentido de que qualquer entidade pode ter acesso a segmentos do conteúdo de um nó apenas através de seu conjunto de âncoras.

Uma vez que o modelo permite que nós de composição distintos contenham um mesmo nó, e nós de composição possam ser aninhados em qualquer profundidade, desde que a restrição de um nó não conter recursivamente a si mesmo seja obedecida, é necessário introduzir o conceito de perspectiva. Intuitivamente, a *perspectiva* de um nó identifica através de qual seqüência de nós de composição aninhados uma certa instância do nó está sendo observada. Formalmente, a perspectiva de um nó N é uma seqüência $P = (N_m,....,N_1)$, com $m \geq 1$, tal que $N_1 = N$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para $i \in [1,m)$ e N_m não está contido em qualquer nó. Note que podem haver várias perspectivas diferentes para um mesmo nó N, se este nó estiver contido em mais de uma composição. A *perspectiva corrente* de um nó é aquela percorrida pela última navegação ao nó (as diversas formas de navegação serão definidas posteriormente). Dada a perspectiva $P = (N_m,....,N_1)$, o nó N_1 é chamado *nó base da perspectiva*.

3.1.2 Elos, Eventos e Pontos de Encontro

No NCM, os elos são definidos nos nós de composição, ao contrário de modelos como os baseados em HTML, onde os mesmos se encontram definidos nos nós de conteúdo. Um elo é caracterizado como uma relação *m:n* entre um conjunto de *pontos terminais origem* e um conjunto de *pontos terminais destino*, onde os pontos terminais origem e destino definem eventos.

Estendendo a definição apresentada na Seção 2.1, um *evento* é a exibição, *evento de exibição*, ou a seleção, *evento de seleção*, da região de uma âncora de um nó. O modelo introduz também um outro tipo de evento, denominado *evento de atribuição*, que corresponde à mudança de um atributo de um nó, ou à alteração da condição de habilitação das mudanças de comportamento definidas no objeto descritor (o objeto descritor será discutido na Seção 3.1.4), ou ainda à ativação dessas mesmas operações de mudança de comportamento. Um *ponto de sincronização* é um caso particular de evento que possui duração atômica (ou seja, duração de uma unidade de informação).

Um evento pode se encontrar em um dos seguintes estados: dormindo, preparando, preparado, ocorrendo ou suspenso. Obviamente, eventos instantâneos (seleção e atribuição) assumem o estado ocorrendo por um tempo infinitesimal. O tempo de duração de um evento de exibição é especificado no descritor, conforme será visto na Seção 3.1.4.

Intuitivamente, tomando como exemplo um evento de exibição, inicialmente o mesmo se encontra no estado dormindo. Quando algum processamento de busca das unidades de informação é realizado, o evento passa para o estado preparando. Ao término desse processamento, o evento entra no estado preparado. Ao iniciar a exibição, o evento passa para o estado ocorrendo. A ocorrência do evento pode ser suspendida temporariamente, quando ele entra no estado suspenso. Ao término da exibição, o evento retorna para o estado preparado. A Figura 5 apresenta as máquinas de estados para os três tipos de evento.

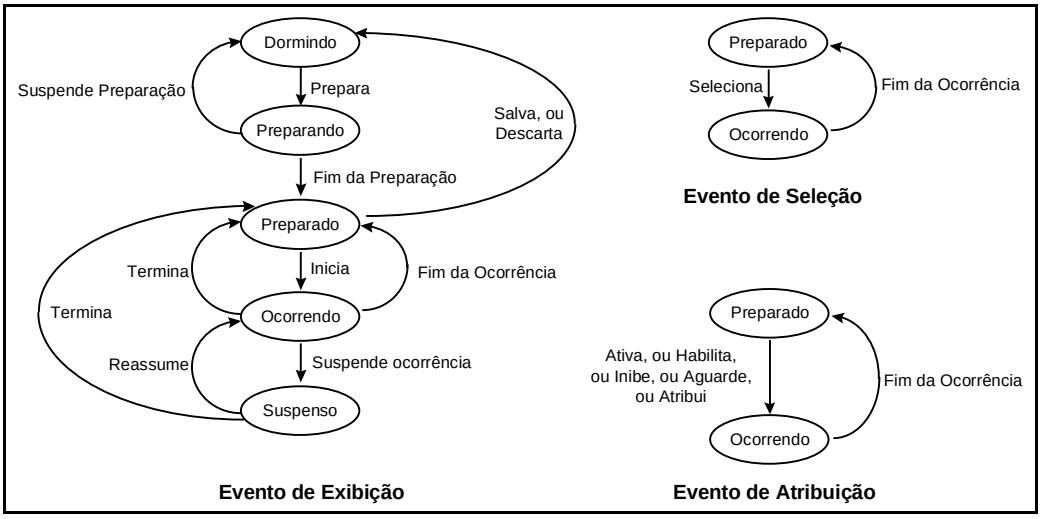


Figura 5 - Máquinas de estados dos diversos tipos de evento

É importante salientar que eventos de seleção e atribuição, para que ocorram, precisam ter suas respectivas âncoras exibidas. Dessa forma, esses dois eventos sempre se encontram no estado preparado, assumindo o estado ocorrendo apenas por um tempo infinitesimal, quando da ocorrência propriamente dita dos mesmos.

Além do estado, todo evento possui um atributo *ocorrido*, que indica quantas vezes o evento, em uma dada apresentação, passou do estado ocorrendo para o estado preparado. É responsabilidade do formatador a manutenção do estado, assim como do valor do atributo ocorrido, para cada um dos eventos

Como foi mencionado anteriormente, todo elemento do conjunto de pontos terminais de origem e do conjunto de pontos terminais de destino define um evento. Segundo a definição do NCM [SoSo97], um ponto terminal (origem ou destino) de um elo é uma tupla do tipo $<(N_k,..., N_2, N_1), A, \alpha, tipo>$ tal que:

1. N_{i+1} é um nó de composição e N_i está contido em N_{i+1}, para todo $i \in [1,k)$, com k>0.
2. A é uma âncora de N_1 , se este for um contexto de versão, senão A é nulo (ϕ)[1].
3. α é uma âncora ou a identificação de um atributo (e seu valor) de um nó N_1.
4. *tipo* especifica se o evento associado a α (ou seja, evento que o ponto terminal define) é um evento de exibição, de seleção ou de atribuição. No caso de evento de atribuição, o valor de α deve identificar o atributo e seu valor, a não ser para ativação ou alteração da condição de habilitação das mudanças de comportamento (explicadas na Seção 3.1.4), definidas no objeto descritor, quando α deve identificar uma âncora. No caso de evento de exibição ou seleção, α sempre identifica uma âncora.

O nó N_k é chamado *nó base* do elo. O nó que tem α como âncora, ou identificador de atributo, é denominado *nó âncora* do elo.

O elo possui ainda um *ponto de encontro*, que define um conjunto de *condições* e *ações*, associadas, respectivamente, aos eventos de origem e destino. O ponto de encontro também apresenta uma sentença baseada no conjunto de condições e ações, especificando relações

[1] A classe *contexto de versão* é uma especialização da classe nó de contexto, e não será abordada nesta dissertação. Dessa forma, em todos os exemplos citados ao longo do texto, este parâmetro no elo assumirá sempre o valor nulo, estando aqui mencionado apenas por completeza da definição. O leitor interessado na definição completa do NCM deve se reportar às referências [SoCR95, SoSo97].

entre os eventos. Dessa forma, um elo funciona como uma asserção lógica sobre uma coleção de eventos, onde condições precisam ser satisfeitas em eventos de origem (pontos terminais de origem) para que ações sejam aplicadas nos eventos de destino (pontos terminais de destino). Existe apenas um tipo de ação que não se aplica a um evento de destino, que é a ação *aguarde*, que especifica a espera por um determinado tempo (retardo). As condições e ações de um ponto de encontro são explicadas em detalhe a seguir.

As condições de um ponto de encontro avaliam valores booleanos e podem ser condições binárias simples ou condições compostas. Toda condição binária simples (ou apenas condição simples) é expressa por duas condições unárias: uma prévia e outra corrente, correspondendo a condições que devem ser satisfeitas, respectivamente, imediatamente antes e no instante de tempo onde a condição binária simples é avaliada. Uma condição binária simples é satisfeita se as suas condições prévia e corrente são satisfeitas. Tanto a condição prévia quanto a corrente podem receber o valor *VERDADE*, caso elas não sejam relevantes na avaliação da condição simples associada. Os operadores de comparação que podem ser utilizados na avaliação das condições simples são: = (igualdade), ≠ (desigualdade), < (inferioridade), ≤ (inferioridade ou igualdade), > (superioridade), ≥ (superioridade ou igualdade). As comparações podem ser realizadas com respeito a:

1. Estados de um evento *E*, definido por um ponto terminal origem do elo;
2. Variável *ocorrido* de um evento *E*, definido por um ponto terminal origem do elo;
3. Valores de atributos de um nó, no caso de um evento de atribuição, e o valor especificado no ponto terminal origem por α;

Qualquer expressão lógica de condições baseada nos operadores $\wedge$ (e), $\vee$ (ou), $\neg$ (negação) e $\oplus$ (retardo) definem uma condição composta. Os operadores lógicos $\wedge$, $\vee$ e $\neg$ possuem os significados usuais da lógica proposicional. Uma expressão ($C \oplus$ *retardo*) é satisfeita em um instante de tempo *t* se a condição *C* estava satisfeita no instante *t* menos o *retardo*. A sentença do elo é avaliada quando ocorre a transição no estado de um evento de uma de suas condições simples, ou seja, no instante em que o estado atual se torna diferente do estado prévio.

As ações de um ponto de encontro podem ser simples ou compostas. Segue a relação das ações simples previstas pelo modelo:

1. “Prepara” evento *E*, que passa para o estado preparando. O evento *E* deve ser do tipo exibição e estar no estado dormindo.
2. “Inicia” evento *E*, que passa para o estado ocorrendo. O evento *E* deve ser do tipo exibição e estar no estado preparado.
3. “Suspende” evento *E*, que passa para o estado suspenso. O evento *E* deve ser do tipo exibição e estar no estado ocorrendo.
4. “Reassume” evento *E*, que passa para o estado ocorrendo. O evento *E* deve ser do tipo exibição e estar no estado suspenso.
5. “Termina” evento *E*, que passa para o estado preparado. O evento *E* deve ser do tipo exibição e estar no estado ocorrendo ou suspenso.
6. “Ativa” as operações de mudança de comportamento associadas ao evento *E* (operações de mudança de comportamento serão definidas nos objetos da classe descritor), passando como parâmetro a condição satisfeita que desencadeou a ação. O evento *E* deve ser do tipo atribuição e aplicado a uma âncora do nó, definido no ponto terminal destino. As operações são ativadas apenas se estiverem habilitadas.
7. “Habilita” utilizada para habilitar as operações de mudança de comportamento associadas ao evento *E*. Assim como na ação *ativa*, o evento *E* deve ser do tipo atribuição e aplicado a uma âncora do nó definido no ponto terminal destino.
8. “Inibe” utilizada para inibir as operações de mudança de comportamento associadas ao evento *E*. Novamente, o evento *E* deve ser do tipo atribuição e aplicado a uma âncora do nó definido no ponto terminal destino.
9. “Atribui (valor)” utilizada para atribuir um valor a um determinado atributo do nó. Tanto o valor como o atributo são especificados no ponto terminal destino do elo, que define o evento do tipo atribuição.
10. “Aguarde (retardo)” utilizada para aguardar um tempo especificado.

Uma ação composta é definida por uma expressão de ações simples, baseada nos operadores $||$ (paralelo) e $\rightarrow$ (seqüencial), que definem a ordem de execução das mesmas.

A Figura 6 ilustra o esquema de um elo NCM, com *m* pontos terminais origem, definindo *m* eventos de origem, e *n* pontos terminais destino, definindo *n* eventos de destino. As condições do ponto de encontro estão associadas aos eventos de origem, enquanto que as ações estão associadas aos eventos de destino (com exceção da ação *aguarde*). A condição

na sentença do ponto de encontro do elo é uma condição composta baseada nas condições do conjunto de condições, ou uma condição binária simples, quando o conjunto de condições possuir uma única condição simples. Da mesma forma, a ação na sentença do elo é uma ação composta, baseada nas ações do conjunto de ações do ponto de encontro, ou uma ação simples, quando no conjunto de ações houver uma única ação simples.

Se um elo possui ao menos um evento do tipo seleção associado a um de seus pontos terminais origem, ele é classificado como um *elo hipermídia (hiper-elo)*. Caso contrário, o elo é classificado como *elo de sincronismo (sinc-elo)*. No NCM todos os relacionamentos entre eventos são definidos nos elos.

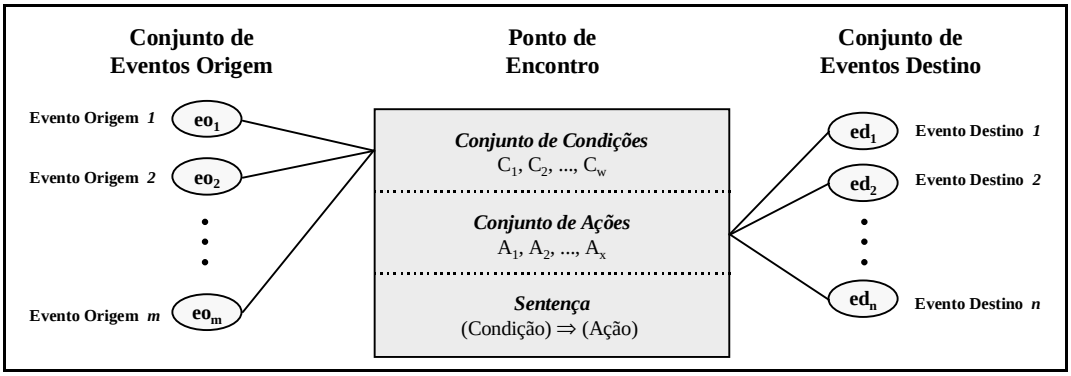


Figura 6 - Elo *m:n* NCM

Neste ponto cabe ressaltar que, diferente de modelos que definem composições apenas para hierarquizar a apresentação de um documento (I-HTSPN [WSSD96] e CMIF [HaRB93]), o NCM permite a definição, além dessas composições, de composições estruturais de fato, agrupando nós e qualquer tipo de elos. Mais ainda, permite a definição de elos ligando dois nós, na composição que contém esses nós, ou que recursivamente contém composições que contêm esses nós, o que permitirá um reuso eficiente de informações. A estruturação dada pelos nós de composição vai permitir a definição hierárquica não apenas de relações representadas por sinc-elos, mas também por hiper-elos, conforme detalhado em [CMSS96]. Maiores considerações sobre os reflexos dessas questões no formatador podem ser encontradas no Capítulo 5 e na referência [SoRo97].

O conceito de evento como exibição de nós já havia sido definido em outros modelos, como o MHEG [MHEG95], no entanto, o NCM estende este conceito introduzindo os eventos de

seleção e atribuição. Mais ainda, e principalmente, os eventos de exibição são definidos não como a exibição de todo o nó, mas como a exibição de qualquer região do nó. No MHEG, porém, os estados não estão nos eventos, mas sim nos nós, e conseqüentemente toda a definição de condições e ações está baseada no estado dos nós. A definição do NCM, atribuindo os estados aos eventos, vai permitir a especificação de pontos de sincronização mais finos entre objetos, sem a necessidade de subdividi-los, tornando o modelo bem mais geral que o modelo MHEG. Note que o modelo MHEG é um caso particular do NCM, onde o estado de um nó corresponde ao estado do evento de exibição da região inteira do nó (região λ do nó).

Como exemplo, suponha uma apresentação testando a sensibilidade auditiva de uma pessoa, onde um nó áudio é exibido apresentando o som de diversos instrumentos musicais. Durante a exibição do áudio, um nó texto com uma lista de botões com o nome de diversos instrumentos é também exibida ao usuário. Suponha também que o autor especifique que: enquanto cada um dos trechos de áudio correspondente a um determinado instrumento estiver sendo exibido, se o usuário sendo testado pressionar o botão com o nome do instrumento correto, ele soma pontos. Em uma especificação MHEG, o áudio deveria ser desmembrado em tantos nós de áudio quantos fossem os instrumentos e, para cada nó, o estado *ocorrendo*, em conjunto com a seleção do botão correto, corresponderia à ação de somar pontos. Por apresentar uma maior flexibilidade, no NCM basta que o usuário identifique no nó de áudio, as regiões (conjunto de unidades de informação, no caso amostras de áudio) correspondentes ao trecho em que o som de cada um dos instrumentos é exibido, e especificar cada um dos elos, baseado no estado *ocorrendo* da respectiva região em conjunto com a seleção do botão correto, a fim de executar a ação determinada pelo autor do documento.

Com as definições apresentadas nesta seção, é possível, através dos elos NCM, representar as treze relações básicas da álgebra de intervalos definidas por Allen [Alle83]. A Tabela 3 mostra como as relações podem ser expressas em elos NCM. A sigla *est*, que aparece na coluna *Ponto de Encontro*, significa estado do evento entre parênteses.

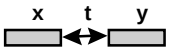
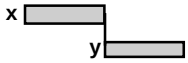
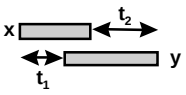
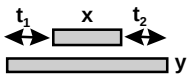
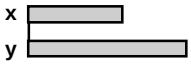
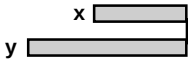
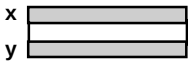
Allen		Elo NCM		
Relação	**Ilustração**	**Eventos de Origem**	**Eventos de Destino**	**Ponto de Encontro**
x before y y after x		eo_1: (<x>, ϕ, λ, exibição)	ed_1: (<y>, ϕ, λ, exibição)	*Condições:* C_1: (est(eo_1)=ocorrendo, est(eo_1)=preparado) *Ações:* A_1: Aguarde(t) A_2: Inicia(ed_1) *Sentença:* $C_1 \Rightarrow (A_1 \rightarrow A_2)$
x meets y y met by x		eo_1: (<x>, ϕ, λ, exibição)	ed_1: (<y>, ϕ, λ, exibição)	*Condições:* C_1: (est(eo_1)=ocorrendo, est(eo_1)=preparado) *Ações:* A_1: Inicia(ed_1) *Sentença:* $C_1 \Rightarrow A_1$
x overlaps y y overlapped by x		**ELO 1** eo_1: (<x>, ϕ, λ, exibição) **ELO 2** eo_1: (<x>, ϕ, λ, exibição)	**ELO 1** ed_1: (<y>, ϕ, λ, exibição) **ELO 2** ed_1: (<y>, ϕ, λ, exibição)	**ELO 1** *Condições:* C_1: (est(eo_1)=preparado, est(eo_1)=ocorrendo) *Ações:* A_1: Aguarde(t_1) A_2: Inicia(ed_1) *Sentença:* $C_1 \Rightarrow (A_1 \rightarrow A_2)$ **ELO 2** *Condições:* C_1: (est(eo_1)=ocorrendo, est(eo_1)=preparado) *Ações:* A_1: Aguarde(t_2) A_2: Termina(ed_1) *Sentença:* $C_1 \Rightarrow (A_1 \rightarrow A_2)$
x during y y contains x		**ELO 1** eo_1: (<y>, ϕ, λ, exibição) **ELO 2** eo_1: (<x>, ϕ, λ, exibição)	**ELO 1** ed_1: (<x>, ϕ, λ, exibição) **ELO 2** ed_1: (<y>, ϕ, λ, exibição)	**ELO 1** *Condições:* C_1: (est(eo_1)=preparado, est(eo_1)=ocorrendo) *Ações:* A_1: Aguarde(t_1) A_2: Inicia(ed_1) *Sentença:* $C_1 \Rightarrow (A_1 \rightarrow A_2)$ **ELO 2** *Condições:* C_1: (est(eo_1)=ocorrendo, est(eo_1)=preparado) *Ações:* A_1: Aguarde(t_2) A_2: Termina(ed_1) *Sentença:* $C_1 \Rightarrow (A_1 \rightarrow A_2)$
x starts y y started by x		eo_1: (<x>, ϕ, λ, exibição)	ed_1: (<y>, ϕ, λ, exibição)	*Condições:* C_1: (est(eo_1)=preparado, est(eo_1)=ocorrendo) *Ações:* A_1: Inicia(ed_1) *Sentença:* $C_1 \Rightarrow A_1$
x finishes y y finished by x		eo_1: (<x>, ϕ, λ, exibição)	ed_1: (<y>, ϕ, λ, exibição)	*Condições:* C_1: (est(eo_1)=ocorrendo, est(eo_1)=preparado) *Ações:* A_1: Termina(ed_1) *Sentença:* $C_1 \Rightarrow A_1$
x equals y		**ELO 1** eo_1: (<x>, ϕ, λ, exibição) **ELO 2** eo_1: (<x>, ϕ, λ, exibição)	**ELO 1** eo_1: (<x>, ϕ, λ, exibição) **ELO 2** ed_1: (<y>, ϕ, λ, exibição)	**ELO 1** *Condições:* C_1: (est(eo_1)=preparado, est(eo_1)=ocorrendo) *Ações:* A_1: Inicia(ed_1) *Sentença:* $C_1 \Rightarrow A_1$ **ELO 2** *Condições:* C_1: (est(eo_1)=ocorrendo, est(eo_1)=preparado) *Ações:* A_1: Termina(ed_1) *Sentença:* $C_1 \Rightarrow A_1$

Tabela 3 - Relacionamentos temporais de Allen expressos em elos NCM

3.1.3 Exemplo de Documento NCM

Com o objetivo de exemplificar os conceitos de nós de composição, nós terminais, elos e perspectivas, suponha o documento criado segundo o NCM, e esquematizado na Figura 7. O documento é representado pela composição *C*, que contém outras duas composições, C_p e C_i. Em C_p estão contidos os nós terminais *I* e A_p, e em C_i os nós terminais *I* e A_i.

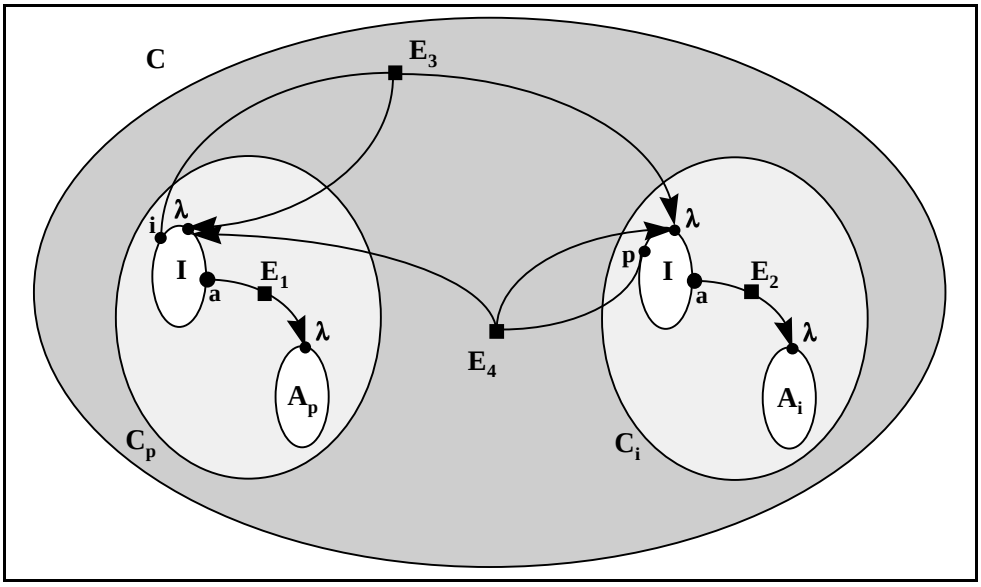


Figura 7 - Exemplo de Documento no NCM

O documento é uma apresentação da história do carnaval carioca, e as composições C_p e C_i representam, respectivamente, a apresentação do documento em português e em inglês. Nesse ponto já é possível perceber a utilidade dos nós de composição, permitindo a criação de diferentes visões para uma mesma apresentação. O nó imagem *I* é um desenho da passarela do samba, A_p é um nó áudio descrevendo as origens do carnaval carioca em português e A_i é um nó áudio que apresenta a mesma descrição, mas em inglês. O documento possui ainda quatro elos: E_1 (contido em C_p), E_2 (contido em C_i), E_3 e E_4 (contidos em *C*). E_1 e E_2 são elos *1:1*, enquanto E_3 e E_4 são elos *1:2*. O relacionamento especificado pelo elo E_1 determina que ao selecionar o alto-falante (simbolizado pela região *a*) no desenho da passarela do samba, no contexto da apresentação em português, o áudio em português contando as origens do carnaval carioca seja exibido. Já o relacionamento especificado por E_2 determina que ao selecionar o mesmo alto-falante no desenho da passarela do samba, mas no contexto da apresentação em inglês, o áudio em inglês contando as origens do carnaval carioca seja exibido. Os elos E_3 e E_4 permitem a troca entre

os contextos de apresentação, respectivamente, do português para o inglês e do inglês para o português. A Tabela 4 apresenta, para cada elo, os eventos de origem e destino, definidos pelos seus respectivos pontos terminais, enquanto que a Tabela 5 descreve os pontos de encontro dos elos.

Elos	Eventos de Origem	Eventos de Destino
E_1	eo_1: (<I>, ϕ, a, seleção)	ed_1: (<A_p>, ϕ, λ, exibição)
E_2	eo_1: (<I>, ϕ, a, seleção)	ed_1: (<A_i>, ϕ, λ, exibição)
E_3	eo_1: (<C_i, I>, ϕ, i, seleção)	ed_1: (<C_i, I>, ϕ, λ, exibição), ed_2: (<C_p, I>, ϕ, λ, exibição)
E_4	eo_1: (<C_p, I>, ϕ, p, seleção)	ed_1: (<C_p, I>, ϕ, λ, exibição), ed_2: (<C_i, I>, ϕ, λ, exibição)

Tabela 4 - Definição dos eventos dos elos

Elos	Condições	Ações	Sentença
E_1	C_1: <estado(eo_1)=ocorrendo, estado(eo_1)=preparado>	A_1: Inicia(ed_1)	$C_1 \Rightarrow A_1$
E_2	C_1: <estado(eo_1)=ocorrendo, estado(eo_1)=preparado>	A_1: Inicia(ed_1)	$C_1 \Rightarrow A_1$
E_3	C_1: <estado(eo_1)=ocorrendo, estado(eo_1)=preparado>	A_1: Termina(ed_1) A_2: Inicia(ed_2)	$C_1 \Rightarrow (A_1 \rightarrow A_2)$
E_4	C_1: <estado(eo_1)=ocorrendo, estado(eo_1)=preparado>	A_1: Termina(ed_1) A_2: Inicia(ed_2)	$C_1 \Rightarrow (A_1 \rightarrow A_2)$

Tabela 5 - Definição dos pontos de encontro dos elos

A importância da perspectiva para o ambiente de execução pode ser verificada no exemplo, onde o mesmo nó terminal *I* está contido em duas composições distintas de *C*. Em cada composição o nó *I* possui uma perspectiva diferente: <*C, C_p, I*> em C_p e <*C, C_i, I*> em C_i. É através da perspectiva de um nó que o formatador do sistema HyperProp sabe como reagir quando da ocorrência de um evento associado a esse nó. Observe que, apenas com a informação de que a âncora *a* do nó *I* foi selecionada, não é possível saber se a ação a ser executada é iniciar a apresentação do aúdio em português ou em inglês. Tal determinação só pode ser feita se a perspectiva através da qual o nó *I* foi exibido for especificada. Evidentemente, o formatador sempre deve tercomo determinar a perspectiva pela qual uma instância de um determinado nó foi exibida.

Nesse ponto cabe introduzir o conceito de elos visíveis. Lembrando que no NCM nós de composição diferentes podem conter o mesmo nó, e que elos podem ser definidos em qualquer composição da perspectiva de um nó, é necessário identificar quais elos efetivamente ancoram, ou passam por um nó em uma dada perspectiva. Os elos desse conjunto são chamados de elos visíveis. Mais precisamente, dado um nó N_1 e uma perspectiva P = (N_m,....,N_1), diz-se que um elo *l* é *visível em* P *por* N_1 se e somente se existe um nó de composição N_i tal que N_i ocorre em P, N_i contém *l* e N_1 ocorre em algum ponto terminal de *l*. O nome elo visível vem do fato de que, ao pedir a exibição um nó, o usuário provavelmente tem em mente a exibição também desses elos. No exemplo da Figura 7, ao exibir o nó *I* pela perspectiva <*C, C_p, I*>, por exemplo, passam a estar visíveis os elos E_1, E_3 e E_4, estando *I* definido nos pontos terminais de origem dos elos E_1 e E_3 , d nos pontos terminais de destino dos elos E_1 e E_4. Com isso, a seleção da âncora *a* dessa instância de *I*, implicará no início da exibição do nó A_p. A idéia de perspectiva oferece uma grande flexibilidade na especificação de apresentação dos documentos, pois apenas pela criação de composições e elos, um mesmo documento pode assumir comportamentos de exibição completamente diferentes.

3.1.4 Objetos de Dados, Descritores, Objetos de Representação e Base Privada

No NCM, associado a todo nó, a partir de agora denominado *nó objeto de dados*, existe um objeto descritor associado. A função do descritor é especificar o comportamento de apresentação do nó, definindo desde os métodos para iniciar a exibição, passando pelas alterações de comportamento programadas para exibição, até determinar a forma de encerramento da apresentação. Dessa maneira, o modelo separa a especificação estrutural do documento da especificação de apresentação.

A agregação de um descritor a um nó objeto de dados dá origem a um *nó objeto de representação,* ou simplesmente objeto de representação. A Figura 8 ilustra a associação entre objetos de dados e descritores. As linhas pontilhadas conectam os objetos de dados e descritores no plano inferior (plano de objetos de dados) com os objetos de representação instanciados no plano superior (plano de objetos de representação). Na figura, nós são representados por círculos de linha cheia, elos por arcos, e composições pela inclusão de

círculos de linha cheia e arcos em círculos de linha cheia maiores. Os descritores são representados por círculos de linha tracejada.

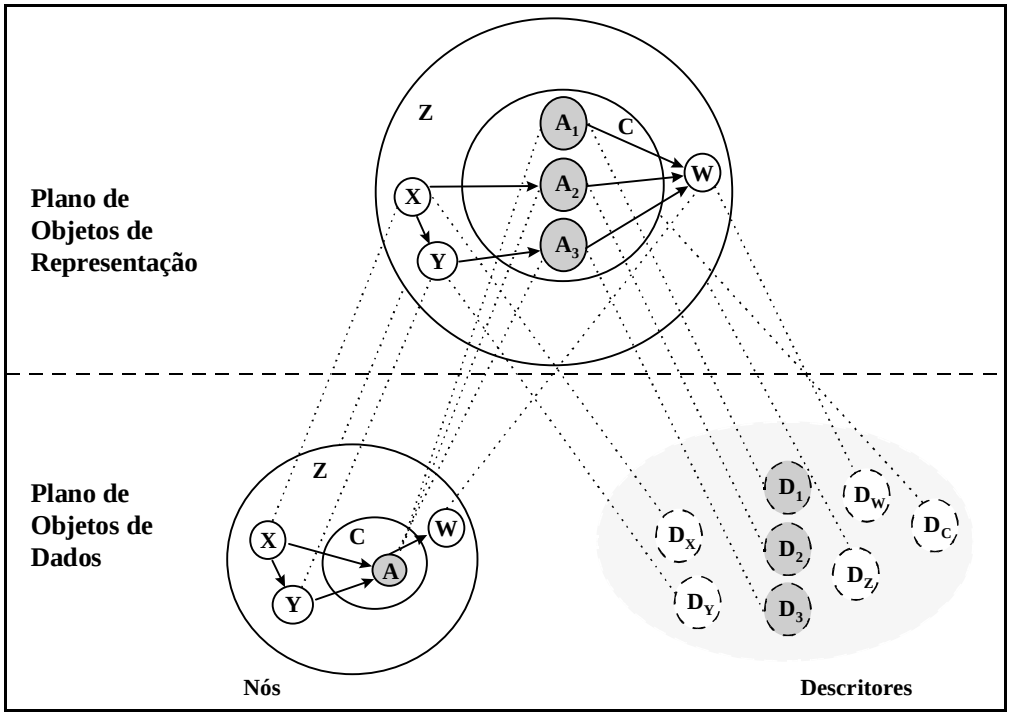


Figura 8 - Planos de objetos de dados e de representação

Conforme definido em [SoSo97], um *descritor* possui como atributos uma especificação de iniciação, uma especificação de término e uma coleção de descrições de eventos.

A *especificação de iniciação* do descritor contém as informações necessárias para iniciar a apresentação do nó, detalhando como será iniciado (métodos de exibição), qual dispositivo de E/S será utilizado etc. Uma especificação de iniciação pode, opcionalmente, possuir uma lista ordenada de operações que devem ser executadas para preparar a exibição do nó. A especificação de iniciação do descritor possui, também, toda informação necessária à obtenção do nó objeto de dados para a criação do nó objeto de representação, como por exemplo, o custo de transferência (retardo, banda passante requerida etc.). Já a *especificação de término* contém as informações necessárias para finalizar a apresentação do nó. Em particular, ela define os métodos que devem ser executados ao final de uma exibição. Assim como a especificação de iniciação, a especificação de término pode possuir,

opcionalmente, uma lista ordenada de operações que devem ser executadas ao finalizar a exibição do nó.

A noção exata do que constitui uma especificação de iniciação e uma especificação de término depende da classe do objeto de dados ao qual o descritor será associado. Por exemplo, para um nó texto, uma especificação de iniciação pode determinar como exibidor o editor de texto Word, passando como parâmetros o tamanho da janela de apresentação do texto e sua posição na tela.

A *lista de operações* apresenta uma seqüência ordenada de operações, onde cada operação é semelhante ao ponto de encontro do elo, sendo composta por um conjunto de condições e ações e uma expressão baseada nas condições e ações. Quando a condição da expressão é satisfeita a ação a ela associada é disparada. A condição de uma operação é semelhante à condição do objeto ponto de encontro, porém, o escopo das entradas está limitado aos eventos e atributos do objeto de representação criado a partir do descritor. Como no ponto de encontro, a condição avalia valores booleanos, podendo ser (binárias) simples ou composta. Toda condição binária simples é expressa por duas condições unárias: uma condição prévia, a ser satisfeita imediatamente antes do instante de tempo onde a condição binária simples é avaliada, e uma condição corrente, a ser satisfeita no instante de tempo onde a condição binária simples é avaliada. Uma condição simples é satisfeita se tanto a condição prévia quanto a condição corrente são satisfeitas. Tanto a condição prévia quanto a corrente podem receber o valor *VERDADE*, caso elas não sejam relevantes na avaliação da condição simples associada. Os operadores de comparação que podem ser utilizados na avaliação das condições simples são: = (igualdade), ≠ (desigualdade), < (inferioridade), ≤ (inferioridade ou igualdade), > (superioridade), ≥ (superioridade ou igualdade). As comparações podem ser realizadas com respeito a:

1. Estados de um evento *E* do objeto de representação criado a partir do descritor.
2. Variável *ocorrido* de um evento *E* do objeto de representação criado a partir do descritor.
3. Valores de atributos do objeto de representação criado.

Qualquer expressão lógica de condições baseada nos operadores $\wedge$ (e), $\vee$ (ou), $\neg$ (negação) e $\oplus$ (retardo) definem uma condição composta. Os operadores lógicos $\wedge$, $\vee$ e $\neg$ possuem os significados usuais da lógica proposicional. Uma expressão (*C* $\oplus$ *retardo*) é satisfeita em um instante de tempo *t* se a condição *C* estava satisfeita no instante *t* menos o *retardo*.

As ações de uma lista de operações, novamente, são dependentes da classe do nó ao qual o descritor será associado. As ações devem sempre corresponder a uma alteração de comportamento da exibição do objeto de representação, criado a partir do descritor. Por exemplo, no caso de um nó imagem estática, uma ação de mudança de comportamento seria "aplique um zoom de X% na imagem". Também são permitidas as ações apresentadas na Seção 3.1.2, desde que aplicadas a eventos do objeto de representação criado a partir do descritor. Dessa forma, as mudanças de comportamento dos nós são programadas em listas de operações que são associadas a eventos, semelhante aos procedimentos de controle encontrados no sistema Firefly [BuZe92]. Os eventos associados às listas de operações são também peças importantes na definição do sincronismo espacial e temporal dos documentos. Os eventos são definidos no conjunto de descrições de eventos do descritor.

A *descrição de um evento*, em um descritor, por sua vez, consiste da tupla <α_{id}, *tipo*, *duração*, *lista de operações*, *estado*>. O α_{id} é um identificador de uma âncora pertencente à coleção de âncoras do nó ao qual o descritor será associado para a formação do objeto de representação. *Tipo* especifica se o evento associado à âncora é de exibição ou de seleção. O *tipo* pode também possuir o valor especial ξ. A *duração* especifica o tempo de exibição do evento. No caso de um evento de seleção, a informação de duração recebe o valor default "infinitésimo", indicando que a duração é instantânea. A *lista de operações* especifica um objeto lista de operações, que deve ter suas condições avaliadas e as ações executadas, caso o evento associado à âncora ocorra e o parâmetro *estado* permita. *Estado* pode assumir dois valores, *habilitado* ou *inibido*, que indicam, respectivamente, se a lista de operações pode ou não ser executada. O valor do parâmetro *estado* é modificado através das ações "habilita" e "inibe", conforme apresentado na Seção 3.1.2. Caso o parâmetro *tipo* possua o valor ξ, a lista de operações deve ser executada, se permitida pelo parâmetro *estado*, quando o descritor receber uma mensagem de outro objeto especificando a âncora definida por α_{id} (no caso um elo, resultado da ação *ativa* de um ponto de encontro).

A Tabela 6 apresenta um resumo das condições em que a lista de operações, definidas em um descritor e associadas a uma determinada âncora do nó ao qual o descritor é associado, é executada, para todas as combinações *estado* x *tipo* numa descrição de evento.

Estado	Tipo	Situação em que a lista de operações de um evento é executada
inibido	qualquer	nunca
habilitado	ξ	Ação "ativa" de um elo aplicada à âncora do evento
	exibição	ocorreu exibição da âncora do evento
	seleção	ocorreu seleção da âncora do evento

Tabela 6 - Condições de execução de uma lista de operações definida em um evento do descritor

Voltando à duração de um evento de exibição, ela é especificada por uma tupla de valores $<t_{min}, t_{ot}, t_{esp}, t_{max}, f_{custo}>$. Os elementos com informação temporal da tupla indicam ao formatador uma faixa de valores que a duração de uma exibição pode assumir, dentro de um intervalo $[t_{min}, t_{max}]$, bem como a duração que levaria a uma melhor qualidade de exibição, dada pelo valor t_{ot}. Já f_{custo} corresponde a uma função que indica o custo correspondente a não ser assumida a qualidade ótima de exibição. O valor da duração esperada (t_{esp}), inicialmente assume o mesmo valor da duração ótima. O formatador pode, baseado em procedimentos de sincronização, alterar esse valor para aquele que atenda à melhor qualidade de exibição do documento como um todo, otimizando o custo global calculado a partir da função de custo especificada por cada evento.

O identificador do descritor, a ser associado a um nó objeto de dados, pode ser definido em um atributo do próprio nó a partir do qual o objeto de representação será criado. Os elos também podem possuir um conjunto de identificadores de descritores, como no Modelo Dexter [HaSc90][2]. Além do próprio nó, e dos elos, os nós de composição também possuem, para cada um dos seus nós componentes, um atributo onde pode ser definido o identificador do descritor que deve ser utilizado na criação do objeto representação do componente.

[2] O descritor em um elo é especificado, opcionalmente, nas ações *prepara* e *inicia*, aplicadas a eventos de exibição. Evidentemente, quando aplicado a ambas as ações em um mesmo evento de exibição, o descritor deve ser o mesmo.

Quando um nó vai ser apresentado, o identificador de descritor definido explicitamente pelo usuário sobrepõe-se ao identificador de descritor definido pelo elo utilizado para alcançar o nó. Esse, por sua vez, sobrepõe-se ao identificador do descritor definido pelo nó de composição que contém o nó, o qual sobrepõe-se ao identificador de descritor definido como atributo do nó. Cada classe de nó possui um descritor default, que é utilizado se nenhuma das formas de identificação do descritor já apresentadas for definida.

Dessa forma, o modelo de apresentação permite a combinação de um objeto de dados com diferentes descritores, originando diferentes objetos de representação. A Figura 8 mostra essa característica com a associação dos descritores D_1, D_2 e D_3 ao objeto de dados *A*, criando os objetos de representação A_1, A_2 e A_3. O nó *A* possui três diferentes representações porque existem três diferentes formas de navegação até ele. Uma através do elo com origem em um evento definido no nó *X*, que instancia o objeto de representação A_2, outra através do elo com origem em um evento definido no nó *Y*, que instancia o objeto de representação A_3, e por último uma através da hierarquia de composições < *Z*, *C* >, que instancia o objeto de representação A_1.

Cabe neste ponto introduzir o conceito de base privada do NCM. A *base privada* é o nó de composição contendo todos os nós (objetos de representação) e elos, representando todos os documentos (possivelmente outras composições) buscados durante uma seção de trabalho do usuário. A base privada deve possuir um método para exibir a sua visão estrutural e temporal, representando a estrutura de todos os documentos trazidos pelo usuário [CMSS96]. Na proposta deste trabalho, o formatador temporal e espacial do sistema HyperProp é implementado como um método da base privada, controlando a apresentação de todos os documentos na seção do usuário.

O leitor deve se reportar à [SoCR95, SoSo97], para uma discussão mais detalhada sobre o modelo de entrada de dados (NCM) do formatador temporal e espacial especificado para o sistema HyperProp.

3.2 Arquitetura

A estrutura do formatador temporal e espacial do sistema HyperProp baseia-se na arquitetura genérica apresentada na Seção 2.3, e está ilustrada na Figura 9, sendo composta pelos módulos pré-compilador, compilador, executor e pelos controladores de exibição. Na figura, as setas pontilhadas representam as comunicações previstas, mas ainda não implementadas no sistema HyperProp.

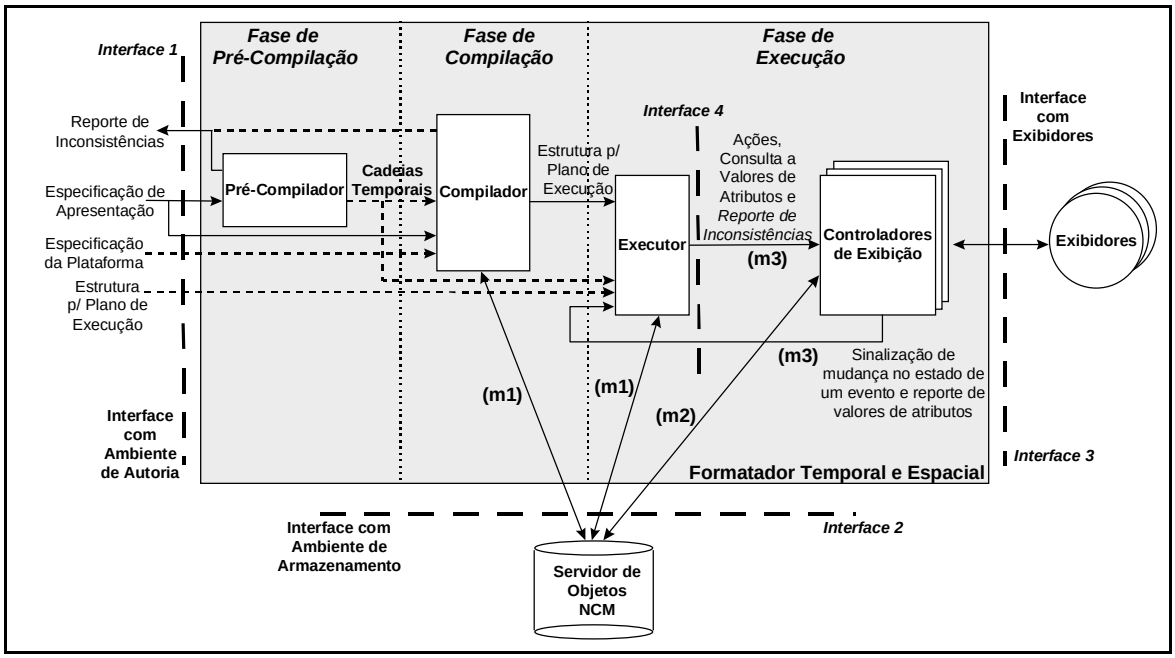


Figura 9 - Arquitetura do formatador temporal e espacial do sistema HyperProp

Além dos componentes internos do formatador (onde encontra-se incluída a interface 4, entre o executor e os controladores de exibição), a figura apresenta as interfaces com o ambiente de autoria (*interface 1*), com o ambiente de armazenamento (*interface 2*), no caso um servidor de objetos NCM, e com os exibidores (*interface 3*). O sistema HyperProp não implementa exibidores especiais de nós de conteúdo, apenas de nós de composição, necessários para exibição da estrutura do documento. Para os nós de conteúdo, a idéia do sistema é incorporar exibidores já disponíveis, através da implementação apenas dos controladores. Para isso, cada exibidor deve ter um controlador implementado que tenha conhecimento de seu funcionamento interno. Sendo assim, a *interface 3*, apresentada na Figura 9, entre os controladores e os exibidores, não é relevante para o executor, estando

dependente da implementação de cada exibidor. A especificação da *interface 4*, entre os controladores e o executor, assim como da *interface 2*, entre o formatador e o servidor de objetos NCM, são o assunto da Seção 3.4. Também não é objeto desta dissertação a definição da *interface 1*, entre o ambiente de execução e o ambiente de autoria. Os mecanismos de pré-compilação, e a comunicação entre o pré-compilador e o ambiente de autoria são descritos em [Cost96].

A organização do formatador na arquitetura apresentada faz com que o sistema atenda aos requisitos de interoperabilidade, a que se propõe, para exibição dos documentos.

3.2.1 Pré-Compilação

Na implementação atual, o pré-compilador realiza os testes de consistência temporal e de conflitos por recursos, retornando os resultados para o editor de sincronismos do sistema, descrito em detalhes em [Cost96]. Ainda não foi implementada a gravação das cadeias temporais parciais geradas por este módulo. Numa implementação futura, as cadeias temporais construídas no pré-compilador irão servir de entrada para o compilador, otimizando a tarefa deste módulo.

Dessa forma, na atual implementação do sistema HyperProp, na execução de um documento, o módulo compilador do formatador recebe a especificação de apresentação diretamente do ambiente de autoria, ou a busca no ambiente de armazenamento (quando o documento já está gravado no servidor de dados NCM).

3.2.2 Compilação

Existem duas formas de implementar a formatação temporal e espacial no sistema HyperProp: apenas com as fases de pré-compilação e execução, ou incluindo também a fase de compilação. No primeiro caso, todas as questões de elos visíveis (e consequëntemente eventos que estão ativos) são resolvidas em tempo de apresentação, conforme os objetos de representação são instanciados. Essa opção dificulta, principalmente, a implementação dos algoritmos de ajuste e pré-busca, pois o executor possui apenas uma visão local da apresentação. Por outro lado, a fase de carregamento do documento para ser exibido torna-se mais simples. Neste trabalho, optou-se por implementar a fase de compilação na

formatação, efetuando um carregamento de todos os elos possíveis de serem disparados durante a exibição do documento, a fim de, futuramente, permitir a implementação de algoritmos de ajuste e pré-carregamento mais poderosos, em detrimento de um maior processamento na preparação para a apresentação do documento. Esse carregamento inicial exige a previsão de todos os objetos de representação que possam vir a ser instanciados. Pode ser que, para documentos extremamente complexos, com muitos nós e elos, a fase de compilação completa torne-se inviável, sendo necessário um refinamento que introduza uma compilação por partes do documento. O ideal é que, em uma versão futura, o pré-compilador faça a compilação para cada composição e armazene o resultado na própria composição. O compilador simplesmente percorreria através dos aninhamentos da composição final que representasse o documento e uniria as diversas compilações. Esse refinamento é deixado como proposta para um trabalho futuro, por ser dependente da gravação do resultado da compilação do pré-compilador.

Sendo assim, devido à escolha de uma fase de compilação no formatador, quando é requisitada a apresentação de um determinado documento ao formatador temporal e espacial do sistema HyperProp, independente se a especificação é passada diretamente pelo ambiente de autoria, ou se está gravada no ambiente de armazenamento, o módulo compilador busca todas as composições que definem o documento, recursivamente. As composições são, então, armazenadas no formatador, já que podem ser necessárias posteriormente, formando uma espécie de memória cache controlada pelo executor. A partir dessas composições, o compilador constrói uma lista com todos os elos que podem ser disparados, ou seja, com todos os elos contidos nas composições trazidas.

O passo seguinte do compilador é construir uma estrutura para mapeamento dos eventos nos elos, estrutura esta que servirá de base ao executor para que ele saiba quais elos precisam ser avaliados quando da sinalização de um ponto de sincronização. Para obter as informações dos eventos que devem ser monitorados pelo formatador, durante a exibição do documento, o compilador precisa obter os pontos terminais dos elos. No entanto, os pontos terminais dos elos trazidos das composições estão associados a nós objetos de dados. Como foi visto, durante a apresentação do documento, uma instância de exibição de um objeto de dados corresponde a um objeto de representação, e um mesmo objeto de

dados origina diferentes objetos de representação, quando alcançado por perspectivas distintas. Conforme exemplificado na Seção 3.1.3, é possível que o conjunto de elos visíveis de um mesmo nó exibido através de diferentes perspectivas seja diferente, o que pode tornar o comportamento de execução completamente distinto (nó *I* no exemplo da Seção 3.1.3). Por isso, é fundamental para o formatador distinguir entre as diversas perspectivas possíveis em um documento.

Além da questão das diferentes perspectivas, um mesmo objeto de dados, numa mesma perspectiva, pode ser associado a diferentes descritores (descritor da classe, descritor definido no próprio nó, descritor definido na composição que contém o nó, descritor definido no elo que resultou a exibição do nó), o que também origina diferentes representações do mesmo objeto de dados.

Uma vez que as condições de um elo podem ser especificadas baseadas no estado de um evento, na variável *ocorrido* de um evento, ou em atributos de um objeto de representação, para cada possível representação de um objeto em um documento, devem ser criadas novas instâncias de todos os elos que partem desse objeto, que só serão avaliadas baseadas nas características do próprio objeto (atributos, estados dos eventos do objeto de representação etc.). Isso ocorre porque em cada um dos objetos de representação, os atributos e eventos nele definidos podem apresentar valores diferentes num dado instante da apresentação. Se novas instâncias de elos não fossem criadas, a avaliação das condições de um determinado elo poderia apresentar resultados diferentes, o que ocasionaria uma ambigüidade na execução.

Tal situação é ilustrada no exemplo da Figura 10(a), onde os elos L'_1 e L''_1, que partem das duas representações de *A*, a princípio poderiam ser considerados idênticos (possuem os mesmos eventos de origem e de destino). Suponha, no exemplo, que *A* seja um nó vídeo, *B* seja um nó texto, *W* seja um nó áudio, e que o ponto de encontro do nó L_1 (definido no plano de objeto de dados) especifique o seguinte relacionamento: se a âncora *b* no nó *B* for selecionada enquanto a região do nó *A*, definida pelos quadros 1 ao 1800 do vídeo (primeiro minuto de apresentação do vídeo, numa taxa de 30 quadros por segundo), estiver sendo exibida, inicie a exibição de *W*.

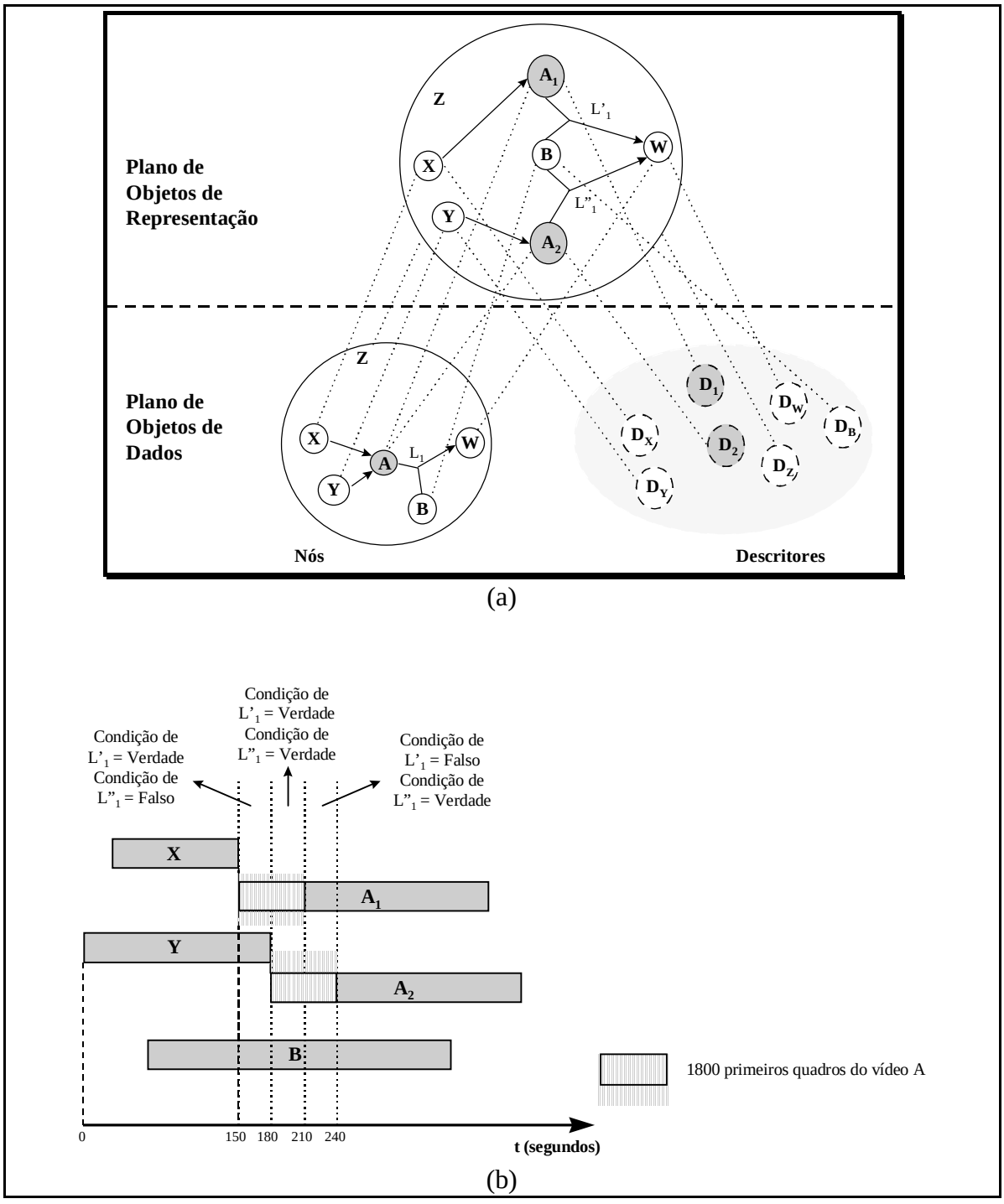


Figura 10 - Distinção de elos com origens em objetos de representação diferentes, originados do mesmo objeto de dados, numa mesma perspectiva

No exemplo, existem duas formas de iniciar a exibição do nó *A*, uma através do elo vindo de *X*, e outra através do elo vindo de *Y*. Se a representação A_1 (objeto de dados *A* associado ao descritor D_1, definido no elo vindo de *X*) começar a ser exibida, por exemplo, trinta

segundos antes da representação A_2 (objeto de dados *A* associado ao descritor D_2, definido no elo vindo de *Y*), haverá um intervalo de tempo em que, se a âncora *b* do nó *B* for selecionada, apenas a avaliação da condição do elo L'_1 retornará verdade. Num outro intervalo, tanto a avaliação da condição do elo L'_1 como do elo L''_1 retornará verdade. Haverá, ainda, um terceiro intervalo em que apenas a avaliação da condição do elo L''_1 retornará verdade, conforme ilustrado na linha temporal de apresentação na Figura 10(b).

Para resolver tal ambigüidade, é necessário que no plano de representação haja uma instância de apresentação do elo, vindo do plano de objeto de dados, para cada combinação das possibilidades de representação de cada um dos nós âncoras dos pontos terminais origem do elo. Pior que isso, a combinação deve considerar também as diversas possibilidades de representação das composições que recursivamente contém os nós âncoras dos pontos terminais origem do elo, na perspectiva utilizada para exibir cada um dos respectivos nós, o que gera uma explosão no número de elos do documento.

Para exemplificar essa explosão combinatória, o exemplo anterior é ligeiramente modificado. São incluídos mais três nós de conteúdo (*C*, *T* e *U*) no nó de composição *Z*, e este é considerado contido em uma outra composição *M*, que além de *Z* também contém o nó de composição *N*. Na composição *Z*, ao elo L_1 é atribuído mais um ponto terminal origem definido no nó de conteúdo *C*. O nó *C*, por sua vez pode ser alcançado por dois elos, um vindo de *T* e outro vindo de *U*. A modificação está ilustrada na Figura 11, onde o valor entre parênteses em cada objeto de dados indica o número de descritores que podem ser utilizados para exibir o mesmo.

No exemplo, o elo L_1 possui setenta e duas instâncias de apresentação, resultado do cálculo apresentado a seguir[3]:

$$(\mathrm{NR}(M) \times \mathrm{NR}(Z) \times \mathrm{NR}(A)) \times (\mathrm{NR}(M) \times \mathrm{NR}(Z) \times \mathrm{NR}(B)) \times (\mathrm{NR}(M) \times \mathrm{NR}(Z) \times \mathrm{NR}(C)) =$$
$$(1 \times 2 \times 3)) \times (1 \times 2 \times 1) \times (1 \times 2 \times 3) =$$
$$6 \times 2 \times 6 = 72$$

[3] NR na expressão significa número de representações

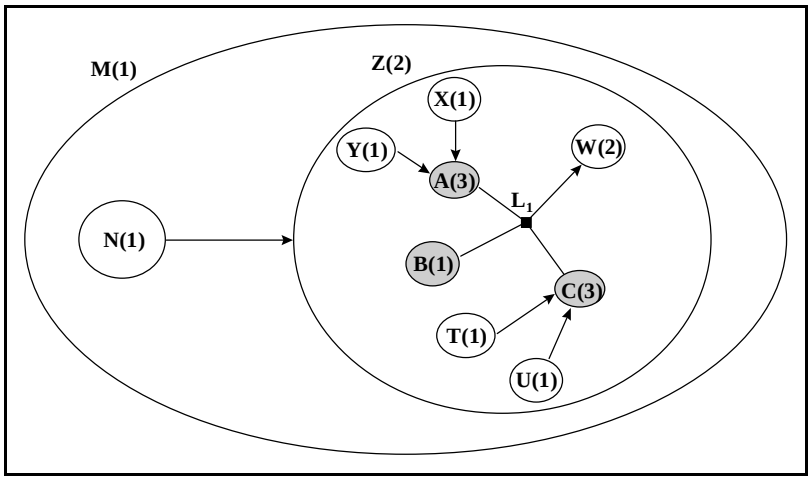


Figura 11 - Instâncias de apresentação de elos

Em conseqüência da questão apresentada, por simplificação, na versão atual da implementação do sistema, apenas os nós terminais apresentam mais de uma possibilidade de representação, o que reduz consideravelmente o número de instâncias de elo criadas na exibição do documento.

Tendo em vista essa questão, de posse das composições e dos elos, o compilador constrói uma estrutura com informações básicas de todos os objetos de representação que podem ser instanciados, denominada *tabela de gerência dos objetos de representação* (TGOR). A cada objeto de representação é atribuído um código, que o identificará univocamente na apresentação. Em cada elemento da tabela, denominado *registro do objeto de representação*, é armazenada a identificação única do objeto, a perspectiva que será utilizada para exibir o nó, a identificação do descritor que será associado, e uma tabela de eventos do objeto de representação que estarão ativos, obtidos dos pontos terminais de origem dos elos que estarão visíveis para essa instância de apresentação do nó. Além disso, em cada registro, um campo é reservado para ser preenchido pelo executor, que determinará a identificação do controlador que estará gerenciando a exibição do objeto de representação, com quem as mensagens deverão ser trocadas na apresentação do documento.

Cada elemento da tabela de eventos, denominado *registro de evento*, possui a identificação do tipo do evento e a região ou atributo ao qual o evento está associado. Essas informações são trazidas do ponto terminal do elo que define o evento. Cada registro de evento possui

ainda um campo com a informação de duração de ocorrência buscada do descritor, além de um campo que identifica o estado prévio, um campo que identifica o estado corrente do evento e um campo que guarda o número de ocorrências do evento. Esses campos são inicializados pelo compilador e mantidos pelo executor. A informação de duração da ocorrência será utilizada nos algoritmos de ajuste e de testes de consistência. Os estados serão informações indispensáveis para avaliação das condições dos elos. O registro de evento possui também um ponteiro para cada instância de elo que tem o evento do objeto de representação específico definido em pelo menos um de seus pontos terminais de origem. Esse é o mapeamento que permitirá ao executor avaliar apenas os elos relevantes, quando o controlador de exibição sinalizar a mudança de estado de um evento. Para que esse mapeamento seja feito, é preciso que a lista de elos seja estendida com todas as instâncias de elo possíveis, conforme explicado anteriormente. Na versão atual do formatador, eventos definidos em operações de mudança de comportamento (descritas na Seção 3.1.4), se não forem origem de algum elo do documento, não são incluídos na tabela de registros de eventos. Dessa forma, na versão atual do formatador, cabe aos controladores o total controle sobre esses eventos.

Resumindo, o procedimento de compilação no sistema HyperProp consiste em buscar todas as composições recursivamente, e obter todos os elos do documento. De posse dos elos e das composições, o compilador constrói a tabela de gerência dos objetos de representação, prevendo todas as possibilidades de representação para cada um dos objetos de dados. Em seguida, o compilador estende a lista de elos com todas as instâncias de apresentação possíveis para cada um dos elos. Finalmente, o compilador faz o mapeamento entre cada um dos registros de evento contidos nos registros de objetos de representação, com a instância do elo a ser avaliado, quando alguma alteração no estado do evento ocorrer. A Figura 12 ilustra a estrutura construída pelo compilador para execução dos elos.

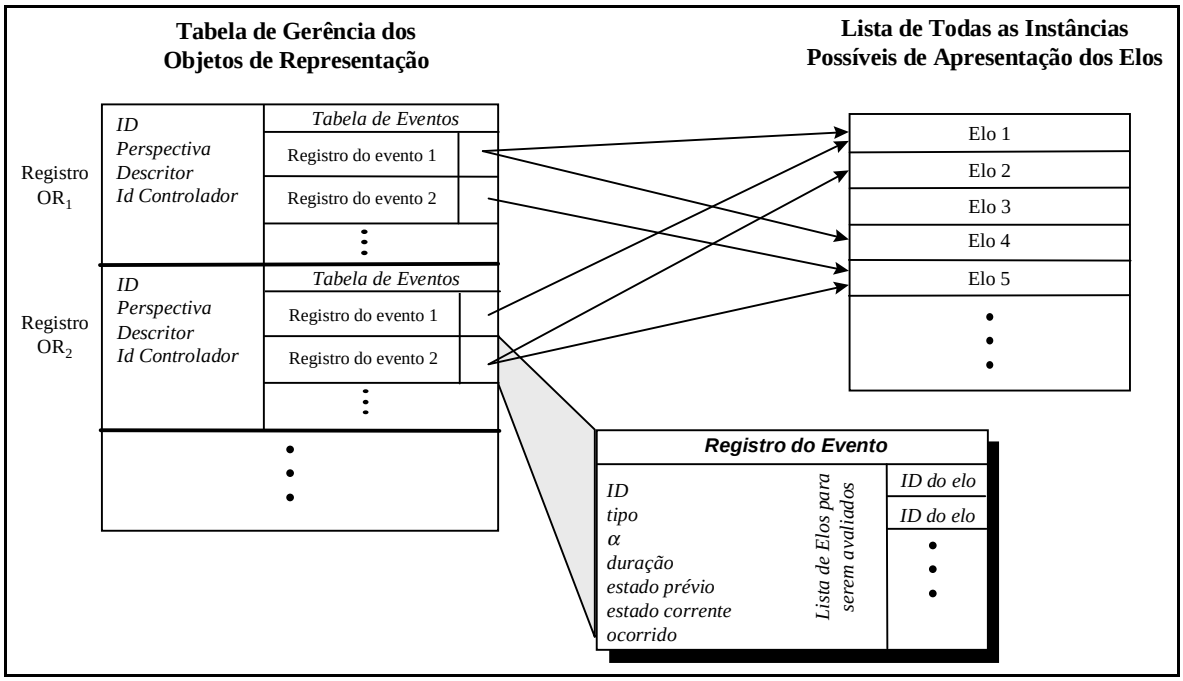


Figura 12 - Estrutura para execução dos elos criada pelo compilador (mapeamento entre eventos e elos)

A última função do compilador é construir a estrutura de dados que deverá ser passada ao executor, para que o mesmo monte o plano de execução. É o plano de execução que permitirá ao formatador implementar algoritmos de ajuste da exibição, de pré-carregamento dos conteúdos dos objetos, e de teste de inconsistências temporais e espaciais durante a apresentação do documento. A idéia é buscar todos os eventos que possuam algum relacionamento temporal previsível e construir as cadeias temporais parciais. Se o pré-compilador constrói as cadeias e as passa para o compilador, ou as grava no subsistema de armazenamento, esse processamento é desnecessário, cabendo ao compilador, apenas unir as diversas cadeias temporais. A união de todas as cadeias temporais parciais, através de seus eventos imprevisíveis, compõe uma máquina de estados de exibição. A estrutura tem como modelo uma rede de Petri com temporização, semelhante à TSPN (Time Stream Petri Net), conforme definido em [DiSé94], com uma pequena extensão. A Seção 3.3 descreve de maneira detalhada o modelo de dados da estrutura de execução do formatador.

3.2.3 Execução

Encerrada a fase de compilação, que pode ser vista como um carregamento do documento, é iniciada a apresentação, propriamente dita. A partir da estrutura recebida do compilador, ou diretamente do ambiente de autoria, o executor constrói o plano de execução, cujo modelo de estrutura está descrito na Seção 3.3.

Antes de iniciar a apresentação do documento, o executor instancia um controlador do sistema, responsável por gerenciar toda a apresentação do documento. Esse controlador oferece, ao usuário uma interface semelhante à ilustrada na Figura 2, permitindo que comandos de meta-diretiva sejam requisitados, como pausa, reinicio e fim da exibição como um todo. O controlador de meta-diretivas está no mesmo nível dos controladores de exibição, passando informações para o executor através da interface 4, apresentada na Figura 9.

Toda vez que a ação de iniciar ou preparar a exibição de um determinado nó deve ser disparada (ação *inicia* ou ação *prepara* aplicada a um evento de exibição da região inteira do nó - região λ), ou quando uma dessas ações for aplicada a um evento de exibição de uma região de um nó cuja região λ não foi exibida[4], um objeto de representação deve ser instanciado.

O executor não cria diretamente o objeto de representação, mas sim um controlador para o objeto. Para tanto, o executor consulta os dados do registro de objeto de representação, inicializados pelo compilador. No registro está a especificação do descritor do nó, que como foi visto, contém toda informação para a recuperação dos seus dados, para sua exibição e a especificação de todas as mudanças de comportamento que podem ocorrer durante a sua exibição. Apenas de posse do descritor é possível saber a forma como o conteúdo de um determinado nó será exibido. Por exemplo, o NCM permite que um nó esteja armazenado como um arquivo texto e que, numa apresentação desse texto, seja utilizado um exibidor que tenha a capacidade de sintetizá-lo em áudio. Para criar o controlador, o executor consulta a especificação de iniciação do descritor e obtém o

[4] Um nó pode começar a ser exibido sem ser do início. Nesse caso, não ocorre explicitamente a transição (preparado, ocorrendo) na região λ (nó inteiro), o objeto de representação pode não ter sido criado.

exibidor especificado. Nesse ponto, o processo que implementa o controlador é iniciado. O sistema mantém um cadastro dos processos controladores, para cada tipo de exibidor.

Após iniciado o controlador, o executor envia os dados para a criação do objeto de representação propriamente dito. Se o descritor a ser associado estiver na cache do formatador, o executor o passa para o controlador diretamente. Caso contrário, ele informa apenas a identificação do descritor, ficando a cargo do controlador ir buscar o objeto no servidor NCM. Quanto aos objetos de dados, no caso em que o nó a ser exibido é de conteúdo (nó terminal), o executor passa apenas sua identificação. Quando o nó a ser exibido é de composição (visualização da estrutura de parte do documento), da mesma forma que o descritor, se o mesmo se encontrar na cache do ambiente, o executor o passa para o controlador, caso contrário, ele informa apenas o seu identificador. Além dessas informações, o executor passa também ao controlador, a lista de eventos que o objeto de representação a ser instanciado deve reportar, obtida através da estrutura de dados recebida do compilador. Cabe ao controlador interpretar as informações de mudança de comportamento e monitorá-las, só devendo informar ao executor os eventos por ele requisitados, que são os eventos que afetam o comportamento temporal do documento.

A fim de evitar que retardos na rede, no subsistema de armazenamento e nos próprios sistemas operacionais impossibilitem que as ações ocorram no instante desejado, o sistema deve possuir algum mecanismo de pré-carregamento. Existem duas formas de implementar o pré-carregamento. Na primeira, o elemento responsável pelo cálculo da preparação é o executor, cabendo ao controlador negociar a qualidade de serviço com o servidor NCM, e buscar o conteúdo. Nesse caso, após criar o controlador, mas antes de iniciar a exibição, o executor deve enviar uma mensagem de preparação para o controlador, com base nos requisitos de busca da informação contidos na especificação de iniciação no descritor. Se a ação *prepara* for explicitamente definida pelo autor, o executor fica isento de fazer qualquer cálculo. Uma outra opção de implementação é deixar toda a decisão de pré-busca por conta do controlador. Nesse caso, o executor deverá instanciar o controlador com alguma antecedência, informando daqui a quanto tempo a exibição do nó será iniciada. Fica a cargo do controlador determinar se será feita, e em que momento, a pré-busca, também baseado nas informações do descritor.

Neste trabalho, como não é objetivo implementar os algoritmos de pré-busca, foi adotada uma simplificação da segunda opção. O objeto de representação é criado sem nenhuma antecedência, e os eventos de exibição já começam do estado preparado (simplificação da máquina de estados apresentada na Figura 5). Ações de prepara, explicitamente especificadas pelo autor, apenas criam o objeto de representação. Dessa forma, quando um nó vai ser exibido, o seu controlador é instanciado, em seguida as informações para criação do objeto de representação são passadas, e a mensagem com a ação *inicia* finalmente é enviada.

Toda vez que uma mensagem de sinalização de ocorrência de um ponto de sincronização chega ao executor, cabe a ele atualizar o estado do evento que acusou o ponto de sincronização e avaliar os elos cuja condição do ponto de encontro pode ter se tornado verdadeira. Se for o caso, as ações do elo são disparadas. Ações, que não sejam as de iniciar e preparar a exibição de um nó, são enviadas diretamente aos controladores. A única exceção é a ação de aguardar um determinado tempo, sendo responsabilidade do executor a contagem do tempo.

Cabe também ao executor manter o plano de execução, recalcular os tempos esperados para os eventos e verificar inconsistências temporais ou espaciais que não puderam ser percebidas antes da apresentação. Os ajustes são feitos pela alteração da duração dos objetos sendo apresentados através do envio de mensagens aos controladores O estudo dos algoritmos de ajuste e os testes de inconsistência também não fazem parte dos objetivos desta dissertação, contudo o modelo de dados proposto para a estrutura de execução é descrito na próxima seção, oferecendo toda a base para implementação desses algoritmos em um trabalho futuro.

3.3 Modelo de Dados da Estrutura de Execução

A escolha de um modelo de fluxo de dados temporais foi feita após observar que a estrutura linear de uma *timeline* é inadequada tanto para edição, conforme amplamente descrito na literatura [BuZe92], como para execução. Ao linearizar a ordem dos eventos, o executor perde as informações sobre os relacionamentos entre os mesmos, expressas nos elos. Para

possuir essas informações, a estrutura deve modelar uma máquina de estados com suporte a paralelismo, o que é conseguido em uma rede de Petri. Em um executor que não permite qualquer ajuste do plano de exibição em tempo de execução, tal característica não é importante, mas para aqueles que o fazem, é essencial.

Diversas têm sido as propostas de utilização de redes de Petri para modelar a apresentação de documentos multimídia, principalmente pela facilidade de representar a sincronização intra-mídia e inter-mídia. Dentre elas podem ser citadas: OCPN [LiGh90], XOCPN [WoQG94], TSPN [DiSé94] e I-HTSPN [WSSD96]. No entanto, todas elas utilizam a rede de Petri como forma de especificação da apresentação do documento, podendo a estrutura de execução ser modelada de outra forma, como é o caso no I-HTSPN, onde é feita uma passagem para um modelo de mais alto nível de abstração no nível de execução, o MHEG.

A proposta neste trabalho é, baseado em uma especificação em mais alto nível (NCM que é conforme com o padrão MHEG), construir a estrutura de execução em rede de Petri. Foi escolhido o modelo TSPN (Time Stream Petri Net) para servir como base da definição da estrutura, devido às suas características, que praticamente atendem a todos os requisitos para que possam ser implementados os algoritmos de ajuste, de pré-carregamento e de testes de inconsistência durante a exibição do documento. A única alteração foi a associação aos arcos de saída de um lugar da rede, não só das durações mínima, ótima e máxima dos eventos (a tupla $\{t_{min}, t_{ot}, t_{max}\}$), mas também o tempo estimado para a duração, calculado pelo formatador (tem-se assim a tupla $\{t_{min}, t_{ot}, t_{esp}, t_{max}\}$). A extensão da TSPN foi denominada TSPN-E (TSPN Estendida) e é apresentada na Seção 3.3.2.

3.3.1 Rede de Petri com Intervalos Temporais nos Arcos (TAPN)

Antes de apresentar a TSPN e sua extensão, é descrito um outro modelo de rede de Petri que em muito influenciou a sua definição. Um modelo de rede de Petri com intervalos temporais nos arcos, classificado em [DiSé94] como TAPN (Time Arc Petri Net), foi proposto por [Walt83].

Definição: Uma TAPN é uma tupla (P, T, B, F, M_o, IM) em que:

- (P, T, B, F, M_o) definem uma rede de Petri, onde P é um conjunto de lugares, T é um conjunto de transições, B e F são as funções de incidência (que definem respectivamente os arcos entre lugares e transições, e entre transições e lugares), e M_o é a marcação inicial da rede.

- Sendo $A = \{a_i = (p_i, t_i) \in P \times T \mid B(p_i, t_i) > 0\}$, que define o conjunto de arcos que chegam em uma transição.
  IM é uma função de mapeamento de intervalos tal que:
  $IM: A \rightarrow Q^+ \times (Q^+ \cup +\infty)$, $IM(a_i) = (x_i, y_i)$, tais que: $0 \leq x_i \leq y_i$.

A função IM define a *semântica temporal* dos arcos. O intervalo $[x_i, y_i]$ é chamado o *intervalo temporal relativo válido* do arco a_i.

Na TAPN, quando um lugar recebe uma ficha, um relógio local para cada arco que parte do lugar é iniciado, e o arco é dito *habilitado*. Assumindo que o lugar p_i é marcado no instante de tempo absoluto τ_i, e supondo que $IM(a_i) = (x_i, y_i)$, onde $a_i = (p_i, t)$ então:

- a transição t não pode ser disparada antes do instante de tempo $(\tau_i + x_i)$ e

- a transição t tem que ser disparada antes ou no máximo no instante de tempo $(\tau_i + y_i)$.

O intervalo de tempo $[\tau_i + x_i, \tau_i + y_i]$ é chamado *intervalo temporal absoluto válido* do arco a_i. Dessa forma, quando dois ou mais lugares estiverem sincronizados por uma transição, a condição precisa ser satisfeita para todos os arcos. Como conseqüência, a transição *t* é *disparável* se a intersecção de todos os intervalos temporais absolutos válidos de todos os arcos não for vazia. Nos casos em que a intersecção é vazia (inconsistência temporal), [Walt83] considera o comportamento da rede indeterminado. Já [Bolo90] define, baseado em um processo algébrico, que nesses casos, as fichas não disparáveis são descartadas e desaparecem dos lugares aos quais estavam associadas. No entanto, para apresentação de documentos multimídia, esse procedimento de sincronização pode não vir a ser adequado. Seria melhor alguma regra de transição mais flexível, pois às vezes o desejo do autor é que a sincronização seja feita, por exemplo, baseada no elemento mais atrasado, sem que seja desejável que a exibição venha a ser interrompida. A fim de tornar a especificação dos

documentos mais flexível, [DiSé94] definiu um outro modelo de redes de Petri, baseado na proposta da TAPN, onde novas regras de disparo são definidas.

3.3.2 Redes de Petri com Intervalos Temporais nos Arcos e Regras de Disparo Alternativas (TSPN e TSPN-E)

Conforme definido em [DiSé94], a TSPN também coloca os intervalos temporais nos arcos que saem dos lugares da rede. No entanto, novas regras de disparo são introduzidas, com o objetivo de ajustar a execução da rede às variações imprevisíveis de apresentação dos objetos, permitindo uma especificação da apresentação mais flexível. A fim de apresentar essas novas regras são definidos cinco instantes de tempo. Para toda transição t e todo arco $a_i = (p_i, t)$ com um intervalo temporal relativo válido $[x_i, y_i]$, são definidos:

- τ_i^{tok}: instante em que a ficha entra no lugar p_i e habilita o arco a_i.
- τ_i^{min}: instante em que o tempo da ficha (tempo do arco) atinge o valor x_i.
- τ_i^{max}: instante em que o tempo da ficha atinge o valor y_i.
- τ^{tra}: instante em que a transição torna-se habilitada: o primeiro instante em que todos os arcos estão habilitados, isto é, quando a última ficha alcança o tempo τ_i^{tok}.
- τ^{fir}: instante em que a transição é disparada.

Note que o intervalo $[\tau_i^{min}, \tau_i^{max}]$ é o *intervalo temporal absoluto válido* do arco a_i, conforme definido na Seção 3.3.1.

As regras de disparo possuem as seguintes propriedades:

1. Cada ficha permanece em um arco a_i por um intervalo de tempo compreendido entre τ_i^{min} e τ_i^{max}, que define a *semântica temporal* do arco a_i. É preciso que $t^{fir} \in [\tau_i^{min}, \tau_i^{max}]$ a fim de garantir a semântica temporal do arco a_i.

2. Quando uma transição t possui mais de um arco de entrada, (p_i, t), a semântica temporal de disparo depende do conjunto de intervalos $\{[\tau_i^{min}, \tau_i^{max}]\}$, relacionados aos diferentes arcos, e da regra de disparo.

A fim de evitar a exigência de que a propriedade (1) seja satisfeita para todos os arcos, requerendo que a seguinte condição seja satisfeita: $\tau_i^{fir} \in \cap_i [\tau_i^{min}, \tau_i^{max}]$, que como foi visto na Seção 3.3.1 pode levar a um intervalo vazio, são introduzidas três novas classes de estratégias de sincronização (regras de disparo):

- estratégia de sincronização dinâmica, baseada no processo mais adiantado, denominada *strong-or*, (isto é, o arco que primeiro atingir seu limite máximo de duração).

- estratégia de sincronização dinâmica, baseada no processo mais atrasado, denominada *weak-and*, (isto é, o arco que por último atingir seu limite máximo de duração).

- estratégia de sincronização estática, baseada em um processo pré-determinado, denominada *master*, (isto é, apenas o intervalo de duração de um arco selecionado, o arco master, é considerado).

Dessa forma, os intervalos temporais absolutos válidos (isto é $[\tau_i^{min}, \tau_i^{max}]$) dos arcos que chegam em uma determinada transição podem ser combinados formando um conjunto de nove intervalos de disparo que definem as nove regras de disparo para as TSPNs, conforme ilustrado na Figura 13. No exemplo, quando a sincronização master é utilizada, o arco (p_2, t) é quem determina o disparo. As nove regras de disparo são equivalentes quando a transição apresenta um único arco de entrada. Dessa forma, na representação gráfica de uma TSPN, não é associado qualquer regra especial de disparo em transições desse tipo.

É importante notar que para o caso da regra de disparo *and* continua a haver a possibilidade do intervalo de intersecção ser vazio. Nesse caso, [DiSé94] define que o momento de disparo ocorrerá na maior duração mínima de todos os arcos que incidem na transição.

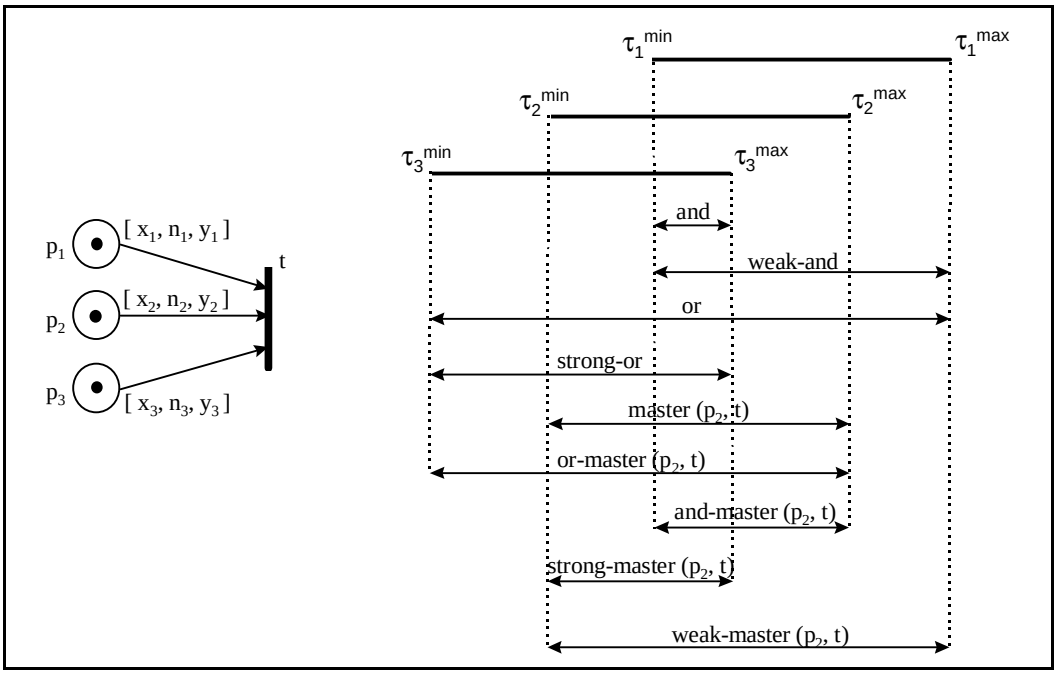


Figura 13 - Semânticas de sincronização das TSPNs

Segue a definição formal de uma TSPN-E, onde a definição de [DiSé94] é estendida para compreender também o valor esperado em cada arco que sai de um lugar.

Definição: Uma TSPN-E é uma tupla do tipo (P, T, B, F, M_o, IM, SYN, MA) onde:

- (P, T, B, F, M_o) definem uma rede de Petri.

- Sendo $A = \{a_i = (p_i, t_i) \mid B(p_i, t_i) > 0\}$, representando o conjunto de arcos que chegam em uma transição.
  IM é uma função de mapeamento de duração tal que:
  $IM: A \rightarrow Q^+ \times Q^+ \times Q^+ \times (Q^+ \cup +\infty)$, $IM(a_i) = (x_i, n_i, e_i, y_i)$
  onde os elementos da tupla (x_i, n_i, e_i, y_i) possuem, respectivamente, o valor mínimo, o valor ótimo, o valor esperado e o valor máximo de duração para o arco a_i, onde as seguintes condições são satisfeitas:
  $x_i \leq n_i \leq y_i$, $x_i \leq e_i \leq y_i$, $x_i \leq n_i < +\infty$ e $x_i \leq e_i < +\infty$.
  O intervalo [x_i, y_i] define o intervalo temporal relativo válido para o arco a_i.

- SYN: $T \rightarrow$ {and, weak-and, or, strong-or, master, or-master, and-master, strong-master, weak-master}, é uma função que associa uma regra de disparo a uma transição.

- MA: $T_m \rightarrow A$ é uma função de mapeamento entre uma transição do tipo master e o arco que comanda o disparo, onde:
  $T_m = \{ t \in T \mid SYN(t) \in \{master, or\text{-}master, and\text{-}master, strong\text{-}master, weak\text{-}master\}\}$

3.3.2.1 Estados e Mudanças de Estados numa TSPN-E

Um estado S de uma TSPN-E é definido como um par (M, I) consistindo de:

- uma marcação M, e

- uma lista I de intervalos temporais válidos dos arcos habilitados. O número de entradas nessa lista é o número de arcos habilitados pela marcação M.

Assumindo que uma transição t seja disparada no instante θ no estado S, então o estado S'=(M', I') alcançado do estado S pode ser calculado como:

1. M' é computado como:
   $(\forall\ p)\ M'(p) = M(p) - B(t, p) + F(t, p)$

2. I' é computado em três passos:

   a) Remover de I todas as tuplas que são relacionadas aos arcos não mais habilitados quando t é disparada;

   b) Subtrair θ dos tempos de todos os intervalos temporais válidos em I e truncá-los, quando necessário, para não possuírem valores negativos. Dessa forma, para todos os arcos $a_k = (p_k, t_n)$, que permaneçam habilitados e que possuam como intervalo temporal válido (x_k, n_k, e_k, y_k) em I, passarão a ter o valor $(\max(0, x_k - \theta), n_k - \theta, e_k - \theta, \max(0, y_k - \theta))$, onde $\max(0, x_k - \theta)$ e $\max(0, y_k - \theta)$ representam, respectivamente, a mínima e máxima duração de habilitação do arco

a_k no estado S', (n_k - θ) representa avanço em relação ao valor nominal, quando positivo, ou atraso quando negativo, e (e_k - θ) representa avanço em relação ao valor esperado, quando positivo, ou atraso quando negativo.

c) Introduzir no domínio os intervalos dos novos arcos habilitados.

3.3.2.2 Regras de Disparo na TSPN

3.3.2.2.1 Regra de Disparo *And*

Uma transição t_k, tal que SYN(t_k) = *and*, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $\max_i (\tau_i^{tok} + x_i) \leq \tau^{tra} + \theta \leq \max (\min_i (\tau_i^{tok} + y_i), \max_i (\tau_i^{tok} + x_i))$, onde
$i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *and* só pode ser disparada em um intervalo compreendido entre a maior duração mínima e a menor duração máxima de todos os arcos que incidem em t. No caso do intervalo ser vazio, o momento de disparo é na maior duração mínima de todos os arcos que incidem em t.

3.3.2.2.2 Regra de Disparo *Weak-And*

Uma transição t_k, tal que SYN(t_k) = *weak-and*, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $\max_i (\tau_i^{tok} + x_i) \leq \tau^{tra} + \theta \leq \max_i (\tau_i^{tok} + y_i)$, onde
$i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *weak-and* só pode ser disparada em um intervalo compreendido entre a maior duração mínima e a maior duração máxima de todos os arcos que incidem em t.

3.3.2.2.3 Regra de Disparo *OR*

Uma transição t_k, tal que SYN(t_k) = *or*, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $\min_i (\tau_i^{tok} + x_i) \leq \tau^{tra} + \theta \leq \max_i (\tau_i^{tok} + y_i)$, onde
   $i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *or* só pode ser disparada em um intervalo compreendido entre a menor duração mínima e a maior duração máxima de todos os arcos que incidem em t.

3.3.2.2.4 Regra de Disparo *Strong-Or*

Uma transição t_k, tal que SYN(t_k) = *strong-or*, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $\min_i (\tau_i^{tok} + x_i) \leq \tau^{tra} + \theta \leq \min_i (\tau_i^{tok} + y_i)$, onde
   $i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *strong-or* só pode ser disparada em um intervalo compreendido entre a menor duração mínima e a menor duração máxima de todos os arcos que incidem em t.

3.3.2.2.5 Regra de Disparo *Master*

Uma transição t_k, tal que SYN(t_k) = *master*, com arco a_m como master, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $(\tau_m^{tok} + x_m) \leq \tau^{tra} + \theta \leq (\tau_m^{tok} + y_m)$

Em outras palavras, uma transição t do tipo *master* só pode ser disparada em um intervalo compreendido entre a duração mínima e máxima do arco master que incide em t.

3.3.2.2.6 Regra de Disparo *Or-Master*

Uma transição t_k, tal que $SYN(t_k)$ = *or-master*, com arco a_m como master, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $\min_i (\tau_i^{tok} + x_i) \leq \tau^{tra} + \theta \leq (\tau_m^{tok} + y_m)$, onde
   $i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *or-master* só pode ser disparada em um intervalo compreendido entre a menor duração mínima de todos os arcos que incidem em t e a duração máxima do arco master.

3.3.2.2.7 Regra de Disparo *And-Master*

Uma transição t_k, tal que $SYN(t_k)$ = *and-master*, com arco a_m como master, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $\max_i (\tau_i^{tok} + x_i) \leq \tau^{tra} + \theta \leq \max ((\tau_m^{tok} + y_m), \max_i (\tau_i^{tok} + x_i))$, onde
   $i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *and-master* só pode ser disparada em um intervalo compreendido entre a maior duração mínima de todos os arcos que incidem em t e a duração máxima do arco master. No caso do intervalo ser vazio, o momento de disparo é na maior duração mínima de todos os arcos que incidem em t.

3.3.2.2.8 Regra de Disparo *Strong-Master*

Uma transição t_k, tal que $SYN(t_k)$ = *strong-master*, com arco a_m como master, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $(\tau_m^{tok} + x_m) \leq \tau^{tra} + \theta \leq \min_i(\tau_i^{tok} + y_i)$, onde
$i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *strong-master* só pode ser disparada em um intervalo compreendido entre a duração mínima do arco master e a menor duração máxima de todos os arcos que incidem em t.

3.3.2.2.9 Regra de Disparo *Weak-Master*

Uma transição t_k, tal que $SYN(t_k)$ = *weak-master*, com arco a_m como master, é disparada em um tempo $\tau^{fir} = \tau^{tra} + \theta$, em um estado S = (M, I), se e somente se:

1) t_k está habilitada pela marcação M no instante τ^{fir}, e

2) $(\tau_m^{tok} + x_m) \leq \tau^{tra} + \theta \leq \max_i(\tau_i^{tok} + y_i)$, onde
$i \in N_{tk} = \{i \in N \mid a_i \in A_{tk}\}$ e $A_{tk} = \{a \in A \mid a = (p_n, t_k)\}$

Em outras palavras, uma transição t do tipo *weak-master* só pode ser disparada em um intervalo compreendido entre a duração mínima do arco master e a maior duração máxima de todos os arcos que incidem em t.

3.3.3 Construção do Plano de Execução

Para construir o plano de execução, o formatador precisa fazer um mapeamento entre os eventos e pontos de encontro dos elos do documento, em lugares, arcos e transições da rede. Dessa forma, os lugares representam pontos de sincronização, os arcos que partem dos lugares representam eventos em um determinado estado, e as transições representam relações entre eventos (que podem ser explícitas, vindas de um ponto de encontro, ou implícitas como será explicado mais adiante). Utilizando essa semântica, foram construídas duas redes, uma que ilustra a máquina de estados de um evento de exibição, e outra que representa a máquina de estados dos eventos de seleção e atribuição, conforme definidas na Seção 3.1.2. As redes de Petri estão apresentadas na Figura 14.

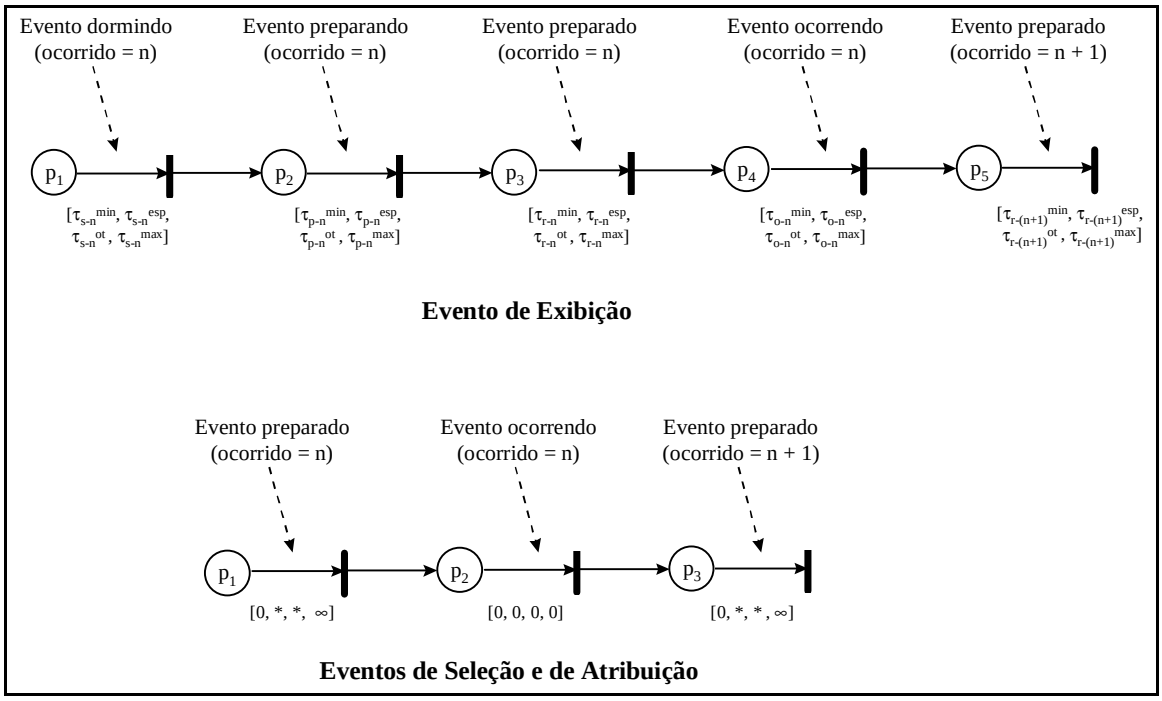


Figura 14 - Modelagem em TSPN-E das máquinas de estados dos eventos NCM[5]

Sempre que uma transição é disparada, significa uma mudança no estado de todos os eventos associados aos arcos que entram na transição. Na construção do plano, em primeiro lugar o formatador deve, para cada evento, construir a rede do evento de forma independente. No caso do evento ser de exibição, os tempos de duração mínima, ótima e máxima do evento no estado ocorrendo, também podem ser inicializados, com os valores trazidos do descritor. Nesse momento, o tempo esperado é colocado com valor igual ao do tempo ótimo. Para os eventos de seleção e atribuição, os tempos recebem os valores colocados na figura. O passo seguinte do formatador é relacionar as diversas redes, através das informações contidas nos elos, e das informações implícitas entre pontos de sincronização de um mesmo objeto de representação.

Os relacionamentos implícitos, entre pontos de sincronização de um mesmo objeto de representação, são obtidos a partir dos registros de eventos definidos em cada um dos objetos. A idéia é que para todo objeto, em paralelo ao lugar da rede que representa o início de sua exibição e ao arco que representa a ocorrência da exibição, estejam interligados todos os pontos de sincronização do objeto, que são origem em algum elo do documento. A

[5] O símbolo * representa tempos indeterminados.

Figura 15(a) ilustra um exemplo de especificação de apresentação de dois nós, estando o encadeamento resultante da união dos diversos pontos de sincronização apresentado na Figura 15(b).

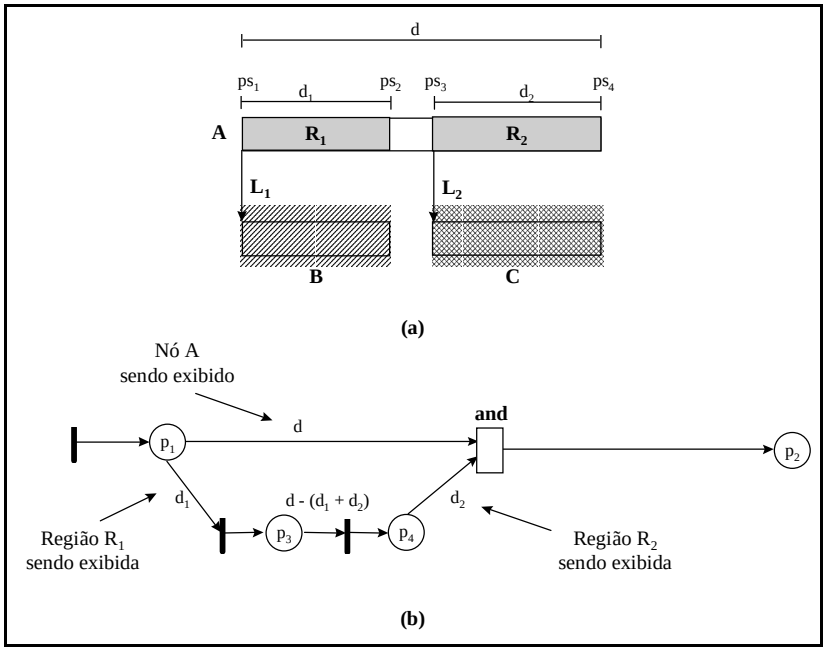


Figura 15 - Relacionamentos temporais internos aos objetos de representação

No exemplo, *A* é um nó áudio que, além do evento de exibição do nó inteiro com duração *d*, possui outros dois eventos de exibição: exibição da região R_1 e exibição da região R_2 que possuem duração d_1 e d_2[6], respectivamente. O evento de exibição da região inteira do nó *A* é representado na rede de Petri pelo lugar p_1 e o arco com duração *d*. O evento de exibição da região R_1 é representado pelo arco que sai do lugar p_1 com duração d_1, enquanto o evento de exibição da região R_2 é representado na rede pelo lugar p_4 e o arco com duração d_2. Cabe ao formatador inferir que entre o fim da exibição da região R_1 e o início da exibição da região R_2 deve passar um tempo igual a $d - (d_1 + d_2)$. Como conseqüência, um lugar na rede com um arco possuindo a duração calculada deve ser colocado entre os pontos de sincronização. No exemplo, p_3 representa esse lugar.

A exibição da primeira região do nó é sempre representada por um arco que parte do lugar que marca o início da exibição do nó por inteiro. Já o fim da exibição da última região do nó

6 Cada duração é composta pela tupla (mínimo, ótimo, esperado, máximo).

é sempre representada como um arco ligado à transição que determina o fim da exibição do nó, sendo essa transição associada a uma regra de disparo do tipo *and*. Para os nós que apresentam como pontos de sincronização de exibição, apenas o início e o fim da exibição da região inteira do nó, a construção da rede em paralelo é dispensável.

É importante observar que, na realidade, o que ocorre é uma pesquisa de todos os pontos de sincronização que definem regiões disjuntas em cada um dos objetos de representação. A partir desses pontos são calculadas as durações entre cada um deles. No exemplo anterior, são obtidos quatro pontos: ps_1, ps_2, ps_3 e ps_4. Os pontos de sincronização que não correspondem ao início ou fim da exibição do nó como um todo definem uma rede paralela ao evento de exibição do nó. A pesquisa para obter as regiões disjuntas é necessária porque o modelo de entrada permite que sejam definidas regiões que possuam intersecções. A Figura 16(a) apresenta o exemplo anterior, onde as regiões do nó *A* possuem unidades de informação comuns. Na Figura 16(b) está a rede de Petri resultante, já com os cálculos de duração entre os pontos de sincronização adjacentes. Dessa forma, a rede de Petri que modela o plano de execução possui eventos de exibição explicitamente definidos pelo autor e eventos de exibição inferidos pelo compilador do formatador.

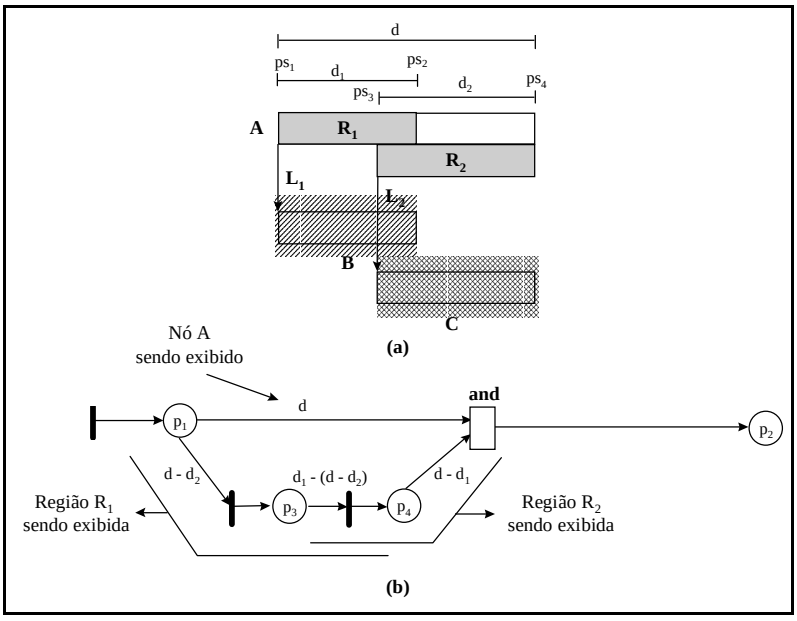


Figura 16 - Regiões disjuntas de um objeto de representação

Os relacionamentos explicitados nos elos também devem originar encadeamentos na rede. A Figura 17 mostra a rede de Petri resultante da especificação mostrada na Figura 15. É

importante lembrar que os lugares, arcos e transições representando os estados dormindo, preparando e preparado foram omitidos para melhorar a compreensão do leitor.

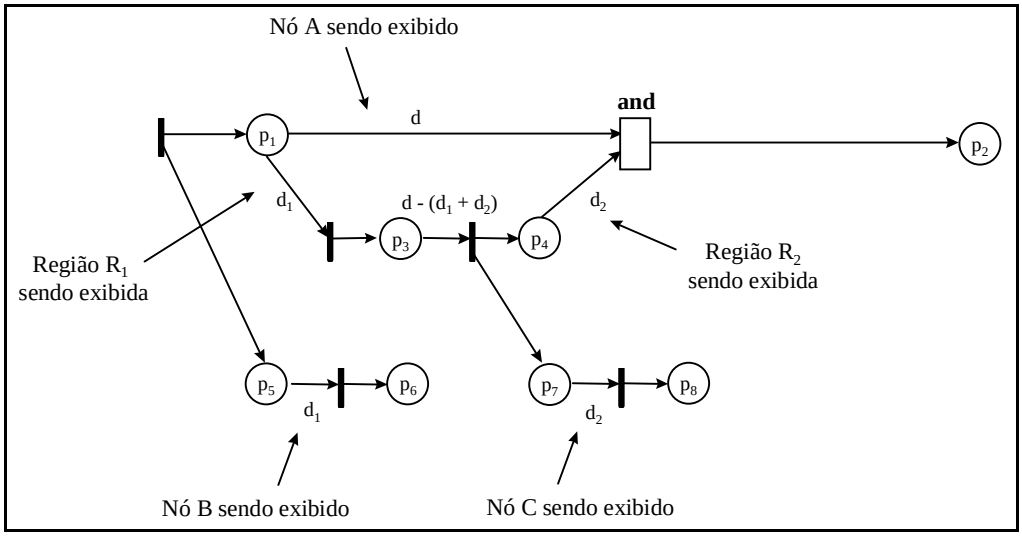


Figura 17 - Mapeamento da especificação NCM em TSPN-E do documento exemplificado na Figura 15(a)

Para elos *1:1*, apenas um arco ligando os pontos de sincronização relacionados é acrescentado à rede. Elos com pontos de encontro que possuam condições compostas (conforme definido na Seção 3.1.2) devem ter regras de disparo associadas de acordo com os operadores da expressão. O operador *e lógico* em uma condição cria uma transição *weak-and* na rede. O operador *ou lógico* origina uma transição do tipo *strong-or*. A *negação lógica* é mais complicada, exigindo que novos lugares sejam acrescentados na rede. A idéia é que a negação seja um lugar ao lado do lugar que representa a condição a ser negada. Quando o lugar representando a condição a ser negada tiver uma ficha a negação não pode ter ficha, e sempre que a condição não tiver ficha, sua negação deve possuir uma ficha. Operadores de retardo são colocados como novos lugares, seguidos por um arco com durações mínima, ótima, esperada e máxima todas iguais, dadas pelo valor do retardo. As regras de disparo *or*, *master*, *and-master*, *or-master*, *strong-master* e *weak-master*, não possuem utilidade no mapeamento dos elos NCM, pois as mesmas são importantes para dar um maior poder de especificação ao autor. Como no sistema HyperProp a especificação não é feita em rede de Petri, e sim em um modelo de mais alto nível, essas regras não são necessárias.

De posse de todas as cadeias logicamente relacionadas, o passo seguinte do formatador seria, utilizando as informações de custo e duração, calcular os tempos esperados para os eventos de exibição, utilizando algum algoritmo que visasse, por exemplo, minimizar o custo total da apresentação. Também nesse ponto, tempos aos estados de preparação deveriam ser atribuídos, utilizando as informações dos descritores, a fim de facilitar a implementação da pré-busca dos conteúdos. Como essas tarefas não foram abordadas nesta dissertação, um estudo mais aprofundado desses procedimentos é deixado como trabalho futuro.

Da forma como se encontra modelada, a rede não dá suporte à suspensão de partes da apresentação. A princípio, poderia se pensar em modelar a pausa na exibição de um evento incluindo mais um lugar na rede, conforme ilustrado na Figura 18.

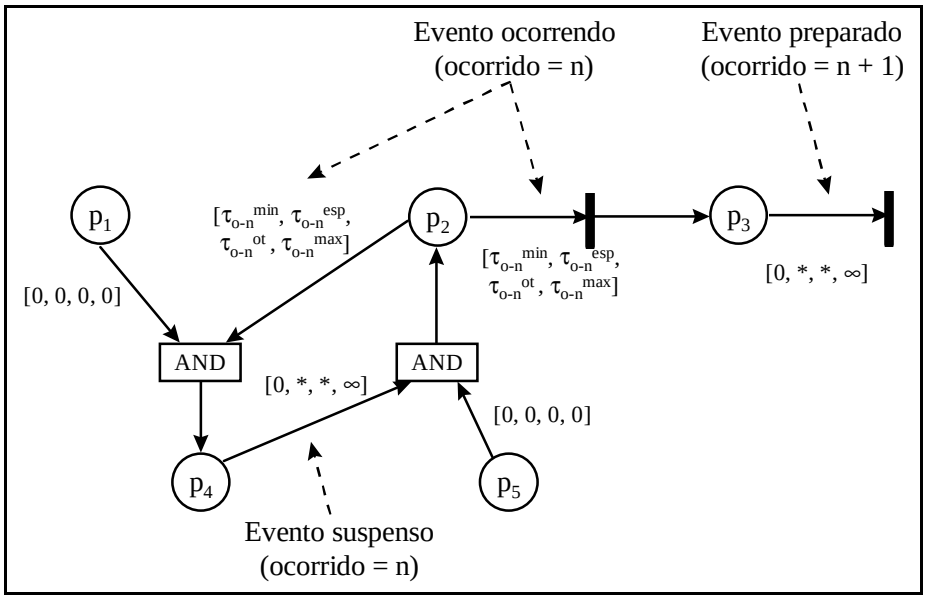


Figura 18 - Tratamento da pausa na exibição

A idéia é que de todo lugar que representasse o início da exibição (lugar p_2, na figura), partisse também um arco para uma transição *and* com um outro arco vindo de um ponto de sincronização que representasse o comando de pausa do usuário (lugar p_1, na figura). No momento em que a ficha fosse colocada em p_1 e o evento de exibição estivesse ocorrendo, a ocorrência seria suspensa e uma ficha seria colocada em p_4, representando o estado suspenso da apresentação. Da mesma forma, do estado suspenso deveria partir um arco

associado por uma transição *and* a um arco vindo de um ponto de sincronização que representasse o reinicio da exibição (lugar p_5, na figura). Tudo funcionaria perfeitamente, se a ficha ao entrar no lugar p_2 não fizesse o tempo dos arcos que de lá partem terem seus contadores novamente inicializados. Por isso, para representar a pausa de apenas parte da execução é necessário um outro artifício. A proposta é que seja utilizada a rede da Figura 14, e que, a todo arco, seja associado um outro estado além de habilitado e inibido, que seria o estado suspenso. A regra de mudanças de estados na rede, apresentada na Seção 3.3.2.1, seria modificada para que o valor θ só fosse subtraído dos arcos que se encontrassem na lista I e não estivessem suspensos. É deixado como proposta de trabalho futuro um estudo mais aprofundado dessa modificação, a fim de observar se as propriedades já verificadas para TSPN poderiam ser consideradas para a TSPN-E.

Uma vez construída a estrutura de execução e calculados os tempos esperados na fase de compilação, é iniciada a apresentação do documento, propriamente dita. Pontos de sincronização associados a eventos de exibição desencadeiam uma comparação do instante ocorrido, com o valor esperado na TSPN e, no caso de estarem conflitantes, indicam se houve um atraso ou adiantamento na exibição. Cabe ao executor analisar os tipos de mídia sendo exibidas para fazer os ajustes nas taxas de apresentação, se necessário, cuidando de novamente testar a consistência temporal e espacial. O sistema ainda não implementa esse tipo de ajuste, sendo previsto o estudo de algoritmos para desempenhá-lo como um trabalho futuro. Neste sentido, cita-se [KiSo95], que apresenta uma interface de definição gráfica temporal muito semelhante a do HyperProp, e que servirá de ponto de partida para o estudo.

Mudanças de comportamento especificadas na fase de autoria, ou provenientes de interação dinâmica com o usuário durante a apresentação do documento, não são modeladas na TSPN. Mudanças que afetam apenas o comportamento espacial nos objetos exibidos são tratadas pelos seus respectivos controladores. Mudanças que afetam o plano de execução ainda não são tratadas na implementação atual, e exigem o mesmo mecanismo comentado no parágrafo imediatamente anterior.

3.4 Interfaces do Formatador

Na Figura 9 são salientadas quatro interfaces do formatador temporal e espacial do sistema HyperProp. A *interface 1*, entre o formatador e o ambiente de autoria, a *interface 2*, entre o formatador e o ambiente de armazenamento, a *interface 3*, entre o formatador e os exibidores propriamente ditos, e a *interface 4*, entre o executor e os controladores de exibição. As três primeiras interfaces são externas ao formatador, enquanto a última é uma interface interna. Assunto desta dissertação, em particular desta seção, é a especificação total da *interface 4* e de parte da *interface 2*. A *interface 1*, a *interface 3* e a parte não especificada da *interface 2*, serão apenas resumidas.

A *interface 1*, entre o ambiente de autoria e o formatador, é composta pelo envio de mensagens informando a especificação de apresentação de um documento e a descrição da plataforma onde a apresentação será feita. Também estão nessa interface mensagens que permitem ao ambiente de autoria requisitar o início de apresentação de um determinado documento que se encontra no subsistema de armazenamento, passando para o formatador a descrição da plataforma de exibição. Opcionalmente, esta interface pode oferecer métodos para que a especificação de apresentação seja passada diretamente, na estrutura de dados que é utilizada para gerar o plano de execução. Do formatador para o subsistema de autoria, são enviadas mensagens que acusam inconsistências temporais ou espaciais possíveis de serem previstas antes do início da execução do documento. O pré-compilador sinaliza inconsistências apenas na especificação de apresentação de uma cadeia temporal passada como parâmetro, enquanto o compilador sinaliza inconsistências em todas as cadeias temporais parciais, tanto na especificação de apresentação de um documento passado como parâmetro, como na especificação de apresentação de um documento trazido do subsistema de armazenamento.

A *interface 3*, entre os controladores e os exibidores, é particular para cada exibidor, devendo explorar todos os recursos do mesmo. Nenhuma padronização nessa interface é feita, uma vez que o objetivo é a interoperabilidade, não só com os exibidores existentes, mas também com aqueles que venham a ser desenvolvidos no futuro. Cabe ao controlador implementado explorar ao máximo os recursos oferecidos pelo exibidor, a fim de alterar

atributos do objeto de representação sendo apresentado pelo exibidor, como por exemplo: aumentar volume no caso de ser um nó áudio, alterar a velocidade de exibição no caso de ser um nó vídeo, modificar a posição da janela de apresentação na tela no caso de ser um nó texto etc. Essencial nessa interface, para que o executor possa manter a máquina de estados de cada um dos eventos, é que o controlador saiba receber do exibidor, e repassar para o executor, a sinalização de ocorrência dos seguintes pontos de sincronização:

- para os eventos de exibição: início de preparação, fim de preparação, início de exibição, fim de exibição, pausa na exibição e retorno da exibição.
- para os eventos de seleção: selecionou.
- para os eventos de atribuição: atribuiu.

A *interface 2*, entre o formatador e o ambiente de armazenamento, possui suas mensagens divididas em dois conjuntos, *m1* e *m2*. As mensagens do grupo *m1* são trocadas na comunicação entre o compilador e o executor do formatador com o servidor NCM, enquanto as mensagens do grupo *m2* são trocadas na comunicação entre os controladores de exibição e o servidor NCM.

No conjunto *m1* existem dois tipos de mensagens. A requisição, por parte do compilador, dos nós de composição, recursivamente (para construção da estrutura de dados que deverá gerar o plano de execução), e as requisições dos descritores para criar os registros de objetos de representação. Numa implementação em que não haja fase de compilação no formatador, ou que a compilação seja feita apenas em parte do documento, o executor trocará mensagens com o servidor NCM, para trazer composições e elos em fase de apresentação. Na implementação corrente, em que a compilação carrega todo o documento, não há troca de mensagens entre o executor e o ambiente de armazenamento. A Tabela 7 descreve as mensagens trocadas nessa interface, com seus respectivos parâmetros e requisitantes.

Mensagem	Parâmetros	Requisitante

RequisitaNóComposição	1) identificação única do nó de composição	Formatador
IndicaNóComposição	1) objeto nó de composição	Servidor NCM
RequisitaDescritor	1) identificação única do descritor	Formatador
IndicaDescritor	1) objeto descritor	Servidor NCM

Tabela 7 - Mensagens trocadas na interface formatador/ambiente de armazenamento

No conjunto *m2* estão as mensagens para busca dos objetos de dados e descritores que constituem o objeto de representação gerenciado pelo controlador, e que não tenham sido passados como parâmetros ao mesmo no momento de sua instanciação. Nessa interface deverá haver meios para que o controlador negocie com o servidor NCM uma qualidade de serviço, a fim de garantir que o conteúdo do objeto seja buscado dentro de limites aceitáveis de vazão, retardo, taxas de erro etc. A proposta é que uma interface baseada no conceito de *media pipe*, como a apresentada em [CSCS96] seja utilizada para a busca do conteúdo do objetos. A implementação atual busca os dados sem qualquer controle da qualidade de serviço.

A *interface 4*, entre o executor e os controladores de exibição, abrange as mensagens que devem ser trocadas durante a execução do documento, para controle da apresentação e para sinalização dos pontos de sincronização ocorridos. Na Figura 9, essas mensagens são reunidas no conjunto *m3*. Do executor para os controladores, existem mensagens para criação e destruição dos objetos de representação, para controlar a exibição dos nós e para consultar valores de atributos mantidos pelo controlador. Dos controladores para o executor, são enviadas mensagens de sinalização descrevendo os pontos de sincronização ocorridos e informando o valor de algum atributo que tenha sido requisitado. A Tabela 8 descreve as mensagens trocadas nessa interface, com seus respectivos parâmetros e requisitantes.

Mensagem	Parâmetros	Requisitante
CriaObjetoRepresentação	1) identificação única do objeto de dados (ou o próprio objeto de dados) 2) identificação única do descritor (ou o próprio descritor) 3) lista de eventos origem visíveis 4) tempo para requisição do início da exibição 5) identificação única do objeto de representação	Executor
DestróiObjetoRepresentação	1) identificação única do objeto de representação	Executor
Prepara	1) identificação única do objeto de representação 2) evento	Executor
Inicia	1) identificação única do objeto de representação 2) evento	Executor
Suspende	1) identificação única do objeto de representação 2) evento	Executor
Reassume	1) identificação única do objeto de representação 2) evento	Executor
Termina	1) identificação única do objeto de representação 2) evento	Executor
Habilita	1) identificação única do objeto de representação 2) evento	Executor
Inibe	1) identificação única do objeto de representação 2) evento	Executor
Ativa	1) identificação única do objeto de representação 2) evento	Executor
Atribui	1) identificação única do objeto de representação 2) evento 3) valor	Executor
RequisitaValorAtributo	1) identificação única do objeto de representação 2) evento	Executor
IndicaValorAtributo	1) identificação única do objeto de representação 2) evento 3) valor	Controlador
IndicaPontoSincronização	1) identificação única do objeto de representação 2) evento 3) ponto de sincronização ocorrido	Controlador
IndicaMetaDiretiva	1) identificação da meta-diretiva	Controlador geral da exibição

Tabela 8 - Mensagens trocadas na interface executor/controladores de exibição

As mensagem *CriaObjetoRepresentação* é passada ao controlador, requisitando a instanciação de um objeto de representação. Ela informa o objeto de dados e o descritor que devem ser associados. No caso em que o executor já possui o objeto de dados ou o descritor, ele os passa diretamente ao controlador. Caso contrário, é informado o identificador único do objeto ou do descritor, para que o próprio controlador os busque no servidor NCM. Outro parâmetro passado na criação do objeto de representação é a lista de eventos definidos por pontos terminais origem de elos, denominados *eventos origem visíveis*. Esses eventos visíveis são obtidos dos pontos terminais, apenas dos elos visíveis para a instância de representação sendo criada. A informação dos eventos origem visíveis é importante para evitar que o exibidor apresente uma âncora para o usuário, sem que a mesma faça parte de qualquer elo visível naquela perspectiva de exibição. Isso poderia levar a uma confusão para o usuário assistindo ao documento, pois ele poderia vir a selecionar a âncora, sem que nada ocorresse, pensando que a falta de resposta fosse um erro do sistema, quando na verdade a âncora para ele exibida não estava associada a nenhum ponto terminal origem de elo. A informação dos eventos que são visíveis é também importante para evitar que o controlador monitore e informe eventos que não tenham nenhuma utilidade na execução do documento requisitada, reduzindo o número de mensagens trocadas no sistema. Na criação do objeto de representação, o executor também envia o tempo que falta para que o início da apresentação seja requisitado. Esse dado irá permitir, no futuro, que o controlador implemente o pré-carregamento. Finalmente, existe um parâmetro na criação do objeto de representação, que informa sua identificação única. Essa identificação é criada pelo próprio formatador, conforme gera os registros dos objetos de representação. O objetivo de informar a identificação única para o controlador, é permitir que uma única instância de controlador gerencie a apresentação de mais de um objeto de representação, tornando a implementação dos controladores mais flexível.

DestróiObjetoRepresentação indica que um determinado objeto de representação não será mais exibido na apresentação de um documento. Se esse objeto for o único sendo gerenciado pelo controlador, o próprio controlador pode encerrar sua execução.

As mensagens *Prepara, Inicia, Suspende*, *Reassume* e *Termina* são todas aplicadas a eventos de exibição de um determinado objeto de representação. Já as mensagens *Habilita,*

Inibe, *Ativa* e *Atribui* são aplicadas a eventos de atribuição dos objetos de representação. Nessas quatro últimas mensagens, a identificação do atributo, ou da âncora, está implícita no evento.

Quando o executor precisa avaliar um elo que possui como uma de suas condições o valor de algum atributo de um objeto de representação, é passada uma mensagem de *RequisitaValorAtributo* ao controlador gerenciando o objeto. O executor recebe a resposta através da mensagem *IndicaValorAtributo*.

As ocorrências dos pontos de sincronização são informadas pelos controladores através da mensagem *IndicaPontoSincronização*, onde são passados o objeto de representação onde o ponto de sincronização ocorreu, o evento e tipo de sincronização (selecionou, iniciou preparação, terminou preparação, iniciou exibição etc.).

A ocorrência de meta-diretivas também é sinalizada através dessa interface, utilizando a mensagem *IndicaMetaDiretiva*. O único campo passado nessa mensagem é a identificação da meta-diretiva ocorrida (pausa, reinicio ou encerramento da exibição). A mensagem é enviada pelo controlador especial de exibição do documento, descrito na Seção 3.2.3.

Capítulo 4

Implementação

Este capítulo descreve a implementação do primeiro protótipo do formatador temporal e espacial desenvolvido para o sistema HyperProp. O capítulo apresenta a organização modular do formatador, onde foram implementados os elementos compilador e executor. A estrutura do formatador foi modelada seguindo o paradigma de orientação a objetos. A metodologia de Grady Booch [Booc94] é utilizada para descrever os diagramas de classes projetados. Para as principais classes do formatador são apresentados os atributos e métodos definidos. O capítulo descreve também como foi implementada a comunicação do formatador com o servidor NCM e do executor com os controladores. Finalmente, são apresentados alguns testes feitos, utilizando um controlador básico implementado apenas para que algumas simulações pudessem ser executadas.

4.1 Organização Modular do Formatador

Como foi colocado no primeiro capítulo, apenas algumas questões da formatação temporal e espacial foram abordadas na implementação do primeiro protótipo do formatador HyperProp. Dessa forma, a fase de compilação constrói a estrutura para execução, fazendo todo o mapeamento entre objetos de representação, eventos e elos, conforme descrito na Seção 3.2.2. No entanto, como não foram desenvolvidos algoritmos de ajuste, pré-carregamento e verificação de inconsistências temporais e espaciais, em tempo de execução,

não há a construção do plano de execução utilizando a rede de Petri definida na Seção 3.3. Por esse motivos, a fase de execução implementa apenas a comunicação com os controladores e o suporte a meta-diretivas do usuário.

A Figura 19 ilustra a organização modular do núcleo do formatador HyperProp. Esses componentes formam apenas o núcleo de formatação, porque os controladores, que são processos independentes e que podem inclusive executar em outras máquinas, também fazem parte do formatador.

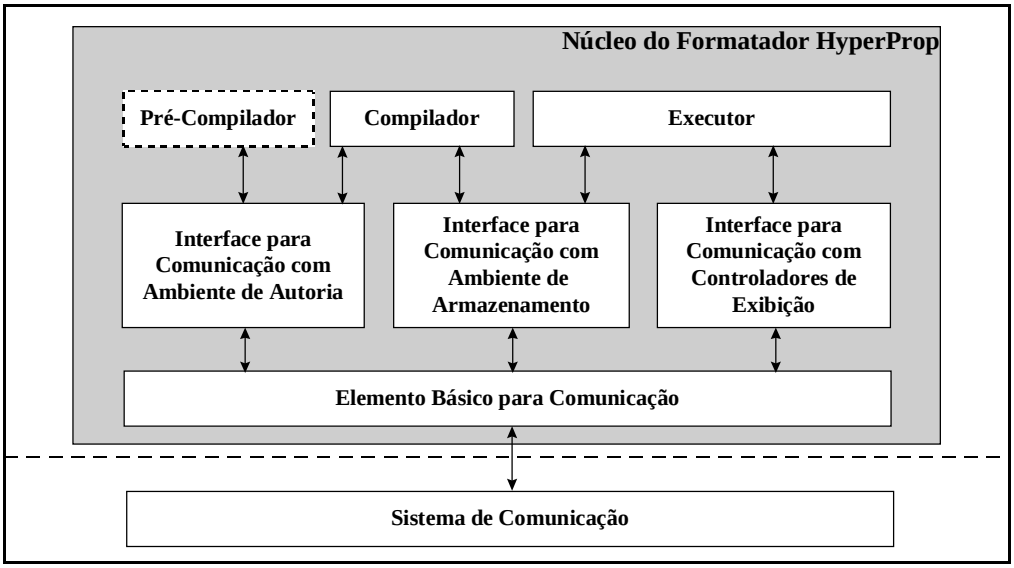


Figura 19 - Organização Modular do Núcleo do Formatador HyperProp

O pré-compilador encontra-se tracejado porque, apesar de fazer parte do núcleo de formatação, o mesmo foi implementado como parte de outro trabalho [Cost96], não sendo mais mencionado ao longo deste capítulo.

Um dos requisitos do sistema HyperProp é ser implementado independente de plataforma, buscando uma total interoperabilidade. Nesse sentido, o formatador foi implementado buscando obedecer a padrões, com o objetivo de tronar-se portável. A atual implementação foi desenvolvida em plataforma UNIX, utilizando a linguagem de programação C++ e bibliotecas padronizadas. A comunicação foi implementada utilizando a biblioteca de sockets, sobre o protocolo TCP/IP. Internamente, o executor utiliza mecanismos de threads, seguindo o padrão POSIX da ISO. Dessa forma, qualquer ambiente com

compilador C++, com bibliotecas que dêem suporte à comunicação TCP/IP (sockets) e suporte a threads (padrão POSIX), pode, teoricamente, ter uma versão compilada do formatador HyperProp.

A comunicação entre os elementos do núcleo do formatador é feita pelo compartilhamento de memória, ou seja, compilador e executor são elementos lógicos de um mesmo processo. No entanto, nada impede que uma implementação distribuída do próprio núcleo de formatação venha a ser desenvolvida, desde que elementos para tratar a interface de comunicação entre os módulos sejam incorporados ao formatador.

A Figura 20 apresenta o diagrama de classes que modela a estrutura para avaliação e disparo dos elos. Todas as classes definidas para utilização do formatador possuem o prefixo HYPF (HyperProp Formatter). As classes que não possuem o prefixo HYPF são utilizadas conforme definidas no NCM. Um elo no ambiente de execução (HYPF_Link) é composto por duas expressões, uma que retrata as condições e outra que retrata as ações. Uma vez que o modelo prevê condições e ações compostas, ambas as expressões são um conjunto de elementos básicos e operadores. As expressões são modeladas como pilhas utilizando a notação polonesa reversa. Dessa forma, um elemento da expressão de ações do elo (HYPF_ActionElement) pode ser um operador (HYPF_ActionOperator) ou um operando (HYPF_SimpleAction). São dois os operadores possíveis da expressão de ações: seqüencial e paralelo. Já a ação simples possui um tipo (HYPF_TAction), uma referência para o registro do objeto de representação (HYPF_RepObjectRegister) ao qual se aplica, e uma referência para o registro de evento contido pelo objeto (HYPF_EventRegister), onde se encontra definida a região do nó na qual a ação se aplica. No caso em que a ação é do tipo *Aguarde*, não há qualquer evento ou objeto de representação associado.

Semelhante às ações, um elemento da expressão de condições do elo (HYPF_ConditionElement) pode ser um operador (HYPF_ConditionOperator) ou um operando (HYPF_BinarySimpleCondition). Os operadores possíveis são quatro: *negação lógica*, *e lógico*, *ou lógico* e *retardo*. A condição binária simples é composta de duas comparações (HYPF_CondComparison), uma que é feita no instante imediatamente anterior e outra feita no instante da avaliação. Ambas as comparações são aplicadas a um

evento de um determinado objeto de representação. A comparação pode ser de cinco tipos (HYPF_CondComparator): igual, diferente, menor, maior, menor ou igual e maior ou igual. A comparação é feita com relação a um entre três possíveis elementos (HYPF_CondCompared): estado de um evento (HYPF_EventStatus), ao valor de um atributo de um evento (HYPF_EventAttribute), no caso o único definido é a variável ocorrido, ou ainda ao valor de um atributo do objeto de representação.

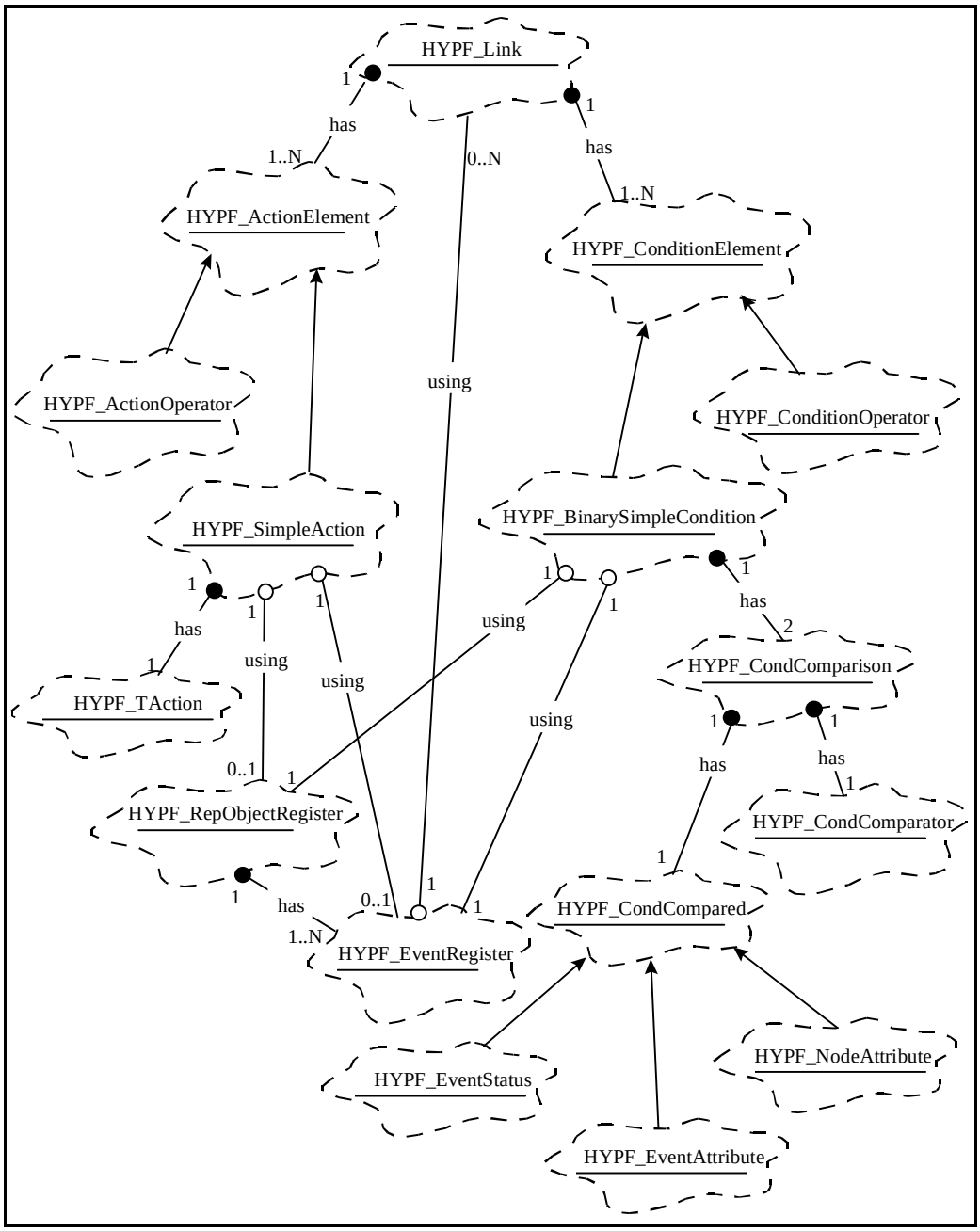


Figura 20 - Diagrama de Classes da Estrutura de Avaliação e Disparo dos Elos

Na Figura 21 encontra-se o diagrama de classes que descreve o registro de objeto de representação e, por conseqüência, o registro de evento. Todo registro de objeto de representação possui a identificação do descritor e uma perspectiva (Perspective) na apresentação do documento. Na própria perspectiva está a identificação do objeto de dados, que associado ao descritor, define o objeto de representação. O registro de um objeto de representação também possui a identificação do controlador (HYPF_Controler) responsável por gerenciar a apresentação do nó e uma lista de registros de evento. A identificação do controlador consiste de um socket e uma porta, através da qual deverá se dar toda a comunicação do executor com o controlador.

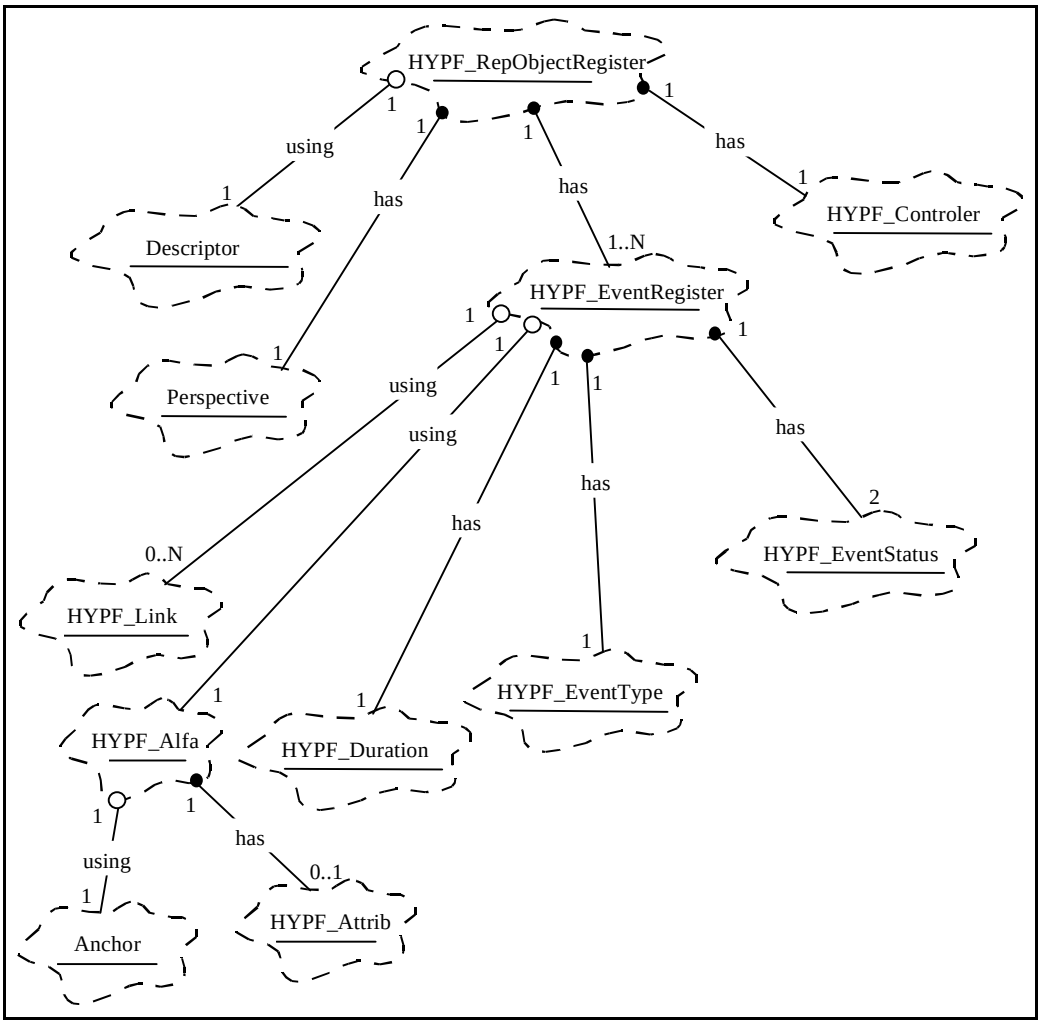


Figura 21 - Diagrama de Classes do Registro de Objeto de Representação

Todo registro de evento possui uma identificação de tipo (HYPF_EventType), que pode ser de seleção, exibição ou atribuição. O registro de evento também possui a identificação de

uma âncora (Anchor) e, no caso de ser um evento de atribuição, a identificação do atributo e um valor a ser associado (HYPF_Attrib). Se o evento for origem de algum elo em que uma condição se aplica a uma transição no estado do evento, ele possui uma referência para o elo, indicando que qualquer sinalização de mudança de estado no evento, sinalizada pelo controlador, deve disparar uma avaliação do elo. Como um mesmo evento pode ser condição de mais de um elo, mais de uma referência pode existir no registro. Todo registro de evento possui uma duração, importante para os eventos do tipo exibição, e duas variáveis que guardam o estado prévio e o estado corrente do evento.

Como mencionado na Seção 3.1.4, o formatador do sistema HyperProp é um método da base privada que, quando ativado, dispara a execução de um determinado documento que esteja na seção de trabalho do usuário. A Figura 22 mostra o diagrama de classes que salienta o relacionamento das classes do formatador com a base privada.

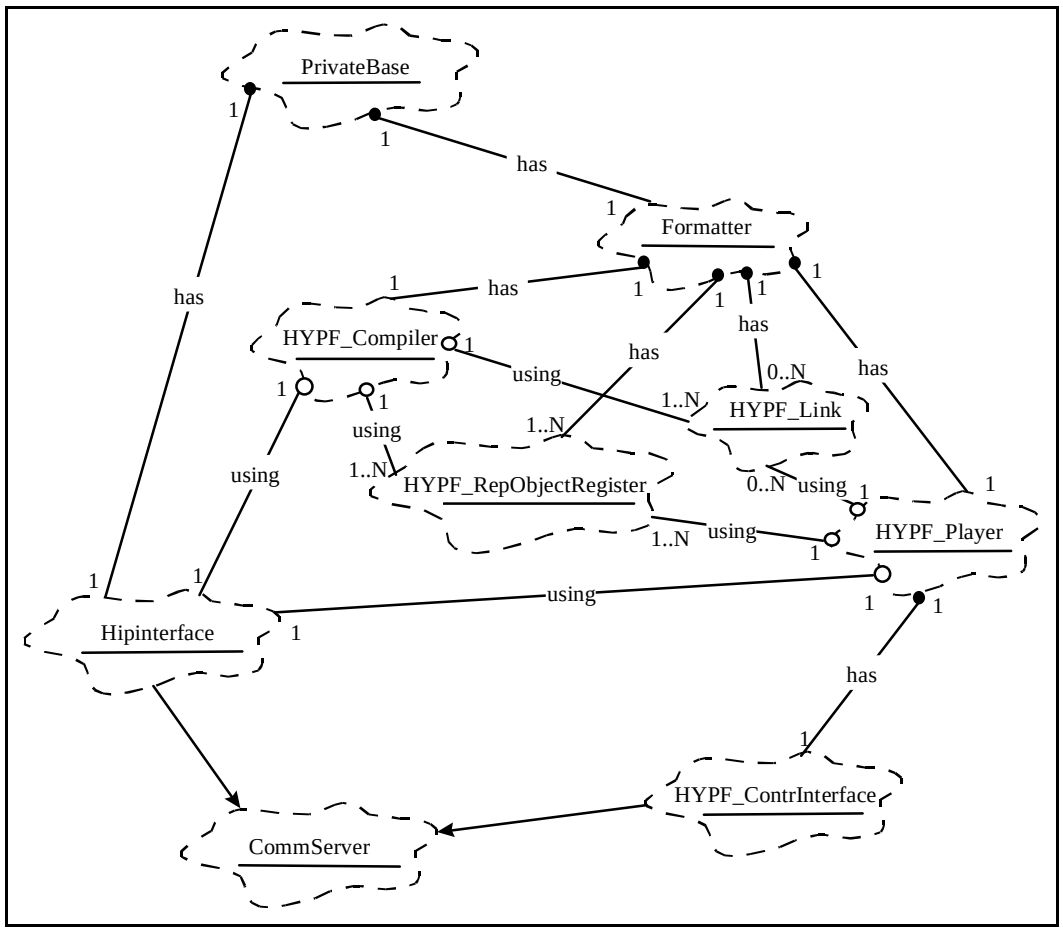


Figura 22 - Diagrama de Classes do Formatador na Base Privada

A base privada (PrivateBase) possui um formatador (Formatter), que é uma classe que auxilia na execução do método de formatação. O formatador possui um compilador (HYPF_Compiler) e um executor (HYPF_Player). Além disso, o formatador possui uma lista de elos do documento executáveis e uma tabela de registros de objetos de representação, explicados anteriormente. Essas estruturas são inicializadas pelo compilador e mantidas pelo executor. O compilador utiliza a própria classe contida na base privada, que dá suporte à comunicação com o ambiente de autoria e com o ambiente de armazenamento, para receber a requisição de apresentação do documento (HIPinterface), que havia sido desenvolvida para implementação dos editores de estrutura [Much96] e sincronismo [Cost96] do sistema. O executor possui ainda uma classe para dar suporte a comunicação com os controladores (HYPF_ContrInterface). Tanto a classe HIPinterface como a classe HYPF_ContrInterface são subclasses de uma classe genérica (CommServer) para suporte à comunicação utilizando o protocolo TCP/IP.

4.2 Principais Classes do Formatador

Esta seção descreve as principais classes do formatador e os principais métodos implementados. A Figura 23 apresenta a notação utilizada para descrição das classes.

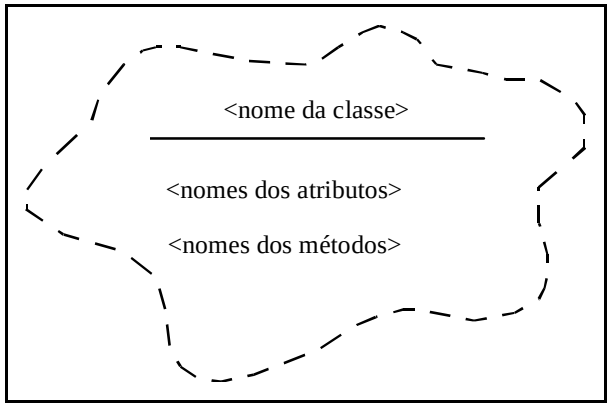


Figura 23 - Notação para descrição das classes

A principal classe para desenvolvimento do formatador é a própria classe Formatter, onde estão definidos o compilador, o executor e a estrutura de dados utilizada para controle da apresentação. A classe Formatter está ilustrada na Figura 24.

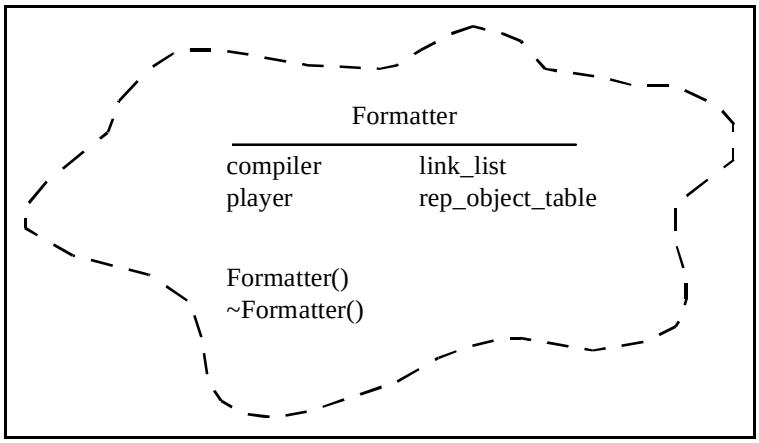


Figura 24 - Classe Formatter

A classe formatador possui como atributos um elemento compilador, um elemento executor e a estrutura para execução, composta por uma lista de elos e pela tabela de registros de objetos de representação. Como métodos, o formatador possui apenas o seu construtor e o seu destrutor. Toda vez que um determinado documento tiver que ser exibido, uma instância de formatador deve ser criada pela base privada, informando a composição que representa o documento e o elemento da base responsável por executar a comunicação com o servidor NCM. Nesse momento, o formatador cria os elementos compilador e executor. É responsabilidade da base privada, destruir a instância de formatador e criar uma nova, para um outro documento que venha a ser apresentado, ou para o mesmo documento, em caso de modificações. É no momento da destruição do formatador que é liberada a área de memória alocada para armazenar a estrutura de execução.

O primeiro passo do formatador, em sua construção, é criar o compilador, ou seja, chamar o método construtor do mesmo. A Figura 25 descreve os atributos e métodos da classe compilador (HYPF_Compiler). Da mesma forma que o formatador, o compilador apenas possui métodos construtor e destrutor. Em sua criação, o compilador recebe a composição a ser apresentada, uma referência para a lista de elos e uma referência para a tabela de registros dos objetos de representação, sendo sua função inicializar esses elementos. O compilador também recebe uma referência para o elemento da base privada responsável por gerenciar a comunicação com o ambiente de armazenamento. O algoritmo utilizado para construção da estrutura é apresentado na Figura 26.

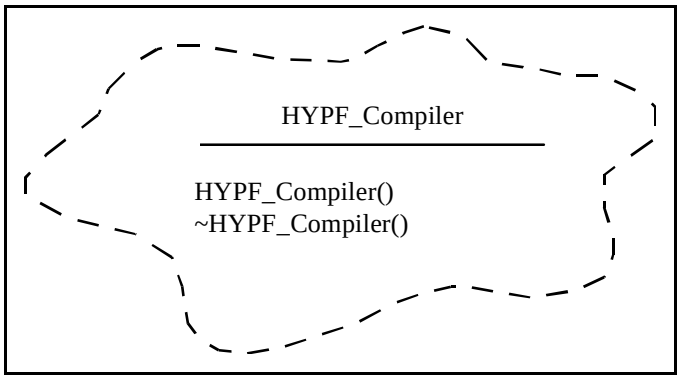


Figura 25 - Classe HYPF_Compiler

```
Compila(nó_composição composição, lista lista_elos, tabela tab_reg_OR,
perspectiva perspectiva_corrente)
{
	Para cada elo contido em composição
		Insere elo em lista_elos;

	Para cada nó contido em composição
	{
		Para cada descritor possível de exibir nó
			Insere (descritor+nó) em tab_reg_OR;
		Se nó é de composição
		{
			Insere nó em perspectiva_corrente;
			Compila(nó, lista_elos, tab_reg_OR, perspectiva_corrente);
			Remove nó de perspectiva_corrente;
		}
	}
}


HYPF_Compile(nó_composição composição, lista lista_elos, tabela tab_reg_OR)
{
	perspectiva perspectiva_corrente;

	Insere composição em perspectiva_corrente;
	Insere (descritor+composição) em tab_reg_OR;

	Compila (composição, lista_elos, tab_reg_OR, perspectiva_corrente);

	CriaInstânciasElos();
	CriaRegistrosEventos();
}
```

Figura 26 - Algoritmo de compilação

Em primeiro lugar, o algoritmo inicializa uma variável perspectiva corrente e uma variável nó corrente, com o nó de composição que representa o documento. Em seguida, é criada uma entrada na tabela de registros de objetos de representação, para a composição com o descritor do nó, ou descritor default da classe. Em seguida, é iniciado o procedimento de

compilação de forma recursiva. Dentro da função compila, que recebe a perspectiva corrente, o nó a ser compilado e a estrutura de execução (lista de elos e tabela de registros dos objetos de representação), para cada elo contido no nó sendo compilado o mesmo é colocado na lista de elos. Para cada nó contido no nó sendo compilado, todas as possíveis representações do nó são postas na tabela de registros dos objetos de representação. Finalmente, para cada nó de composição contido no nó sendo compilado, o mesmo é colocado na perspectiva corrente, e o método compila é chamado para o nó. A estrutura recursiva do algoritmo de compilação se adapta bem à opção de executar a pré-compilação em cada nó de composição onde o resultado já ficaria armazenado, conforme foi mencionado na Seção 3.2.2.

Após a primeira fase de compilação, todas as instâncias de elos são criadas, seguindo o procedimento descrito na Seção 3.2.2. Finalmente, de posse de todos os registros de objetos de representação e de todas as instâncias de elo, os registros de evento são inicializados, em cada registro de objeto de representação.

O passo final na criação do formatador é, após o carregamento do documento feito pelo compilador, iniciar a exibição. A exibição do documento é iniciada com a criação do elemento executor do formatador. A Figura 27 apresenta a classe HYPF_Player da qual o executor é um objeto.

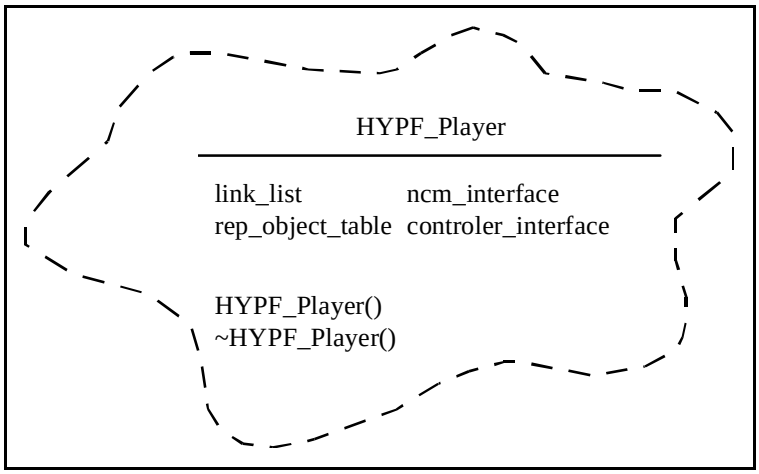


Figura 27 - Classe HYPF_Player

No método construtor da classe HYPF_Player, uma thread é iniciada, sendo responsável por gerenciar a comunicação com os controladores. Nesse momento, o controlador para o

primeiro objeto de representação a ser exibido é iniciado, como um processo independente. A partir daí, a thread de controle da execução assume o controle, recebendo a sinalização dos eventos, avaliando os elos e enviando mensagens para os controladores, ou mesmo criando novos controladores. O elemento gerenciador da interface com os controladores utiliza uma porta pré-determinada. Dessa forma, no momento em que um controlador é iniciado, ele deve enviar uma mensagem ao executor, para que, na tabela de registros de objetos de representação, o campo identificação do controlador seja inicializado pelo formatador.

A classe HYPF_Link, entre outros métodos, possui uma função que avalia a condição do elo. Esse método é chamado pelo executor, quando recebe a sinalização de um evento por parte de algum controlador. Evidentemente, apenas os elos em que o evento é condição origem são avaliados.

4.3 Testes

Para testar a implementação do formatador, devido ao fato de diversas características ainda não se encontrarem desenvolvidas no sistema HyperProp, algumas simulações tiveram que ser feitas.

Numa primeira etapa, foi desenvolvido um controlador único de objetos de representação que simulava a execução dos documentos, sem qualquer tratamento de conteúdo e de informação de apresentação (descritor). O objetivo nessa fase era testar a construção da estrutura interna de execução e a comunicação do executor com os objetos de representação.

Nesse sentido, o executor foi integrado ao editores de documentos do sistema, o Browser de Sincronismo[Cost96] e o Browser de Estrutura[Much96], que foram utilizados para definir os documentos a serem especificados. O simulador, cuja interface é apresentada na Figura 28, funcionava da seguinte forma. A partir de um documento especificado no modelo NCM, existia uma opção para requisitar a execução de uma determinada base privada. A partir daí, o simulador enviava uma mensagem de execução para a base privada, e começa a funcionar como controlador de todos os objetos de representação. Nele, ao

receber uma mensagem de Show, uma vez que não havia o tratamento do conteúdo dos nós, um quadrado era exibido, simulando a exibição do objeto de representação. Essa exibição gerava a sinalização da apresentação do nó inteiro (região λ do objeto de dados) para o executor. O simulador gerava, além de eventos de exibição, eventos de seleção (novamente, apenas da região λ do nó) através do click com o mouse no quadrado que apresentava o objeto. Com isso, foram testadas as interfaces de comunicação, a construção do documento para exibição e a avaliação dos elos no executor.

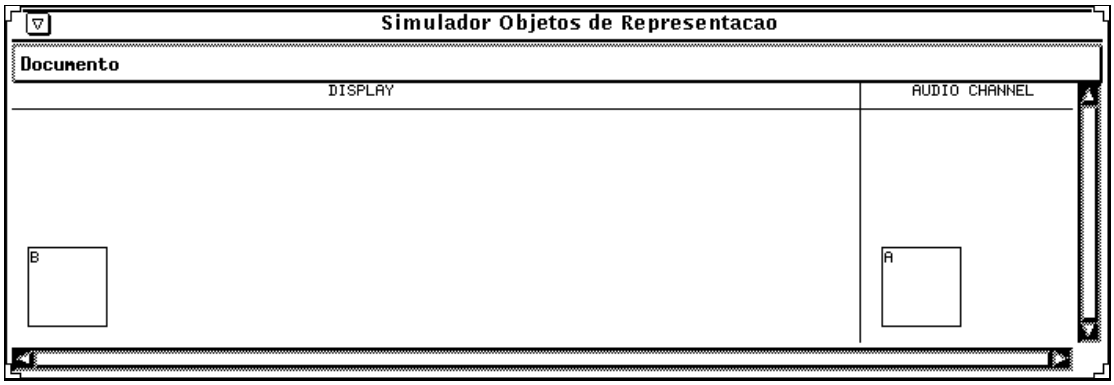


Figura 28 - Primeiro protótipo para teste do formatador

Numa segunda fase, ao invés de possuir um único controlador para todos os objetos de representação, passou a existir um controlador para cada nó. Nessa nova fase, passou a ser considerada a informação do descritor, mas por ainda não haver o tratamento de conteúdo, apenas a região λ continuou a ser considerada. Nessa nova fase, o método de execução passou a fazer parte do próprio browser de estrutura, e uma interface para suporte a comandos do usuário que permitem o controle da apresentação como um todo foi implementada (suporte a meta-diretivas).

Capítulo 5

Trabalhos Relacionados

Este capítulo é dedicado à comparação com trabalhos relacionados. São descritos alguns sistemas que apresentam a formatação temporal e espacial automática para apresentação dos documentos. As características dos diversos formatadores são comparadas com os requisitos apresentados no Capítulo 2 e, ao final do capítulo, uma tabela de comparação, resumindo as diversas características dos sistemas, é apresentada, onde também são incluídas as características do formatador HyperProp.

5.1 Firefly

O sistema Firefly [BuZe92, BuZ93a, BuZ93b] foi desenvolvido com o objetivo de fornecer facilidades para criação, edição, manutenção e apresentação de documentos multimídia/hipermídia.

A apresentação dos documentos definidos no sistema é controlada por um formatador que apresenta arquitetura idêntica à ilustrada na Figura 4, exceto pelo fato de não possuir uma fase de pré-compilação e por não separar os controladores dos exibidores. O compilador do formatador, denominado *scheduler*, constrói uma cadeia temporal de apresentação principal e zero ou mais cadeias temporais auxiliares. A cadeia temporal principal possui todos os pontos de sincronização, no sistema denominados eventos, previsíveis antes do início da exibição, que possuam algum encadeamento temporal. As cadeias temporais auxiliares são

encadeamentos de pontos de sincronização previsíveis, a partir de um ponto de sincronização imprevisível (por exemplo, interação do usuário).

O compilador Firefly recebe a especificação de apresentação de um documento e, em primeiro lugar, obtém a duração entre cada dois pontos de sincronização temporais adjacentes em cada segmento de mídia (correspondente ao conteúdo de um nó terminal no modelo NCM) a ser apresentado. Cada intervalo desses é composto por um valor mínimo, um valor ótimo e um valor máximo, bem como uma função de custo para diminuir ou aumentar a duração em relação ao valor ótimo.

Além das durações entre pontos de sincronização de um mesmo segmento de mídia, o compilador recebe a especificação das restrições temporais de apresentação do documento, que são definidas através de relações de ordem entre dois pontos de sincronização de segmentos de mídia distintos. O compilador também recebe uma lista de operações, que especificam mudanças de comportamento aplicadas a cada ponto de sincronização. O sistema Firefly não permite a definição de composições.

Para construir as cadeias temporais, o compilador aplica um algoritmo para obter as componentes conexas do documento. Inicialmente, cada ponto de sincronização é colocado como uma componente conexa separada. Dois pontos de sincronização são identificados como de uma mesma componente se existe uma duração previsível ou uma restrição temporal entre os mesmos. Ao final, as componentes conexas que não possuem qualquer ponto de sincronização imprevisível são ditas previsíveis. As demais componentes conexas são denominadas imprevisíveis.

O passo seguinte do compilador é calcular os tempos de ocorrência esperados para cada um dos pontos de sincronização. Esse cálculo é feito de forma independente para cada uma das componentes conexas geradas (cadeias temporais). Para o cálculo é utilizado um algoritmo de programação linear (método simplex) que minimiza o custo total de diminuir ou aumentar os intervalos de duração. Os tempos esperados são calculados de forma relativa ao instante de ocorrência do primeiro ponto de sincronização de cada componente conexa. Em fase de compilação, algumas inconsistências temporais são verificadas pelo formatador e reportadas ao autor, no entanto, de maneira rudimentar. Entre os erros reportados estão:

a lista de pontos de sincronização aos quais o algoritmo não conseguiu atribuir um instante esperado e os conflitos de pontos de sincronização (pontos de sincronização de um mesmo segmento de mídia, que receberam o mesmo instante esperado de ocorrência).

Por fim, o compilador, em cada componente conexa, visita os pontos de sincronização em ordem temporal, gerando, em cada um, no máximo uma ação. As ações geradas em componentes conexas imprevisíveis são gravadas na cadeia temporal auxiliar respectiva, enquanto as demais são colocadas na cadeia temporal principal. As ações serão os elementos responsáveis por alterar a velocidade de apresentação dos segmentos de mídia que tiveram suas durações calculadas de forma diferente do valor ótimo.

Geradas e colocadas todas as ações nas componentes conexas previsíveis, o resultado é a cadeia temporal principal (uma estrutura em timeline), que é entregue ao executor do sistema, composto por três elementos básicos: o sistema de apresentação, o tratador de eventos (pontos de sincronização) e os segmentos de mídia. O sistema de apresentação é responsável por gerenciar a cadeia temporal de apresentação do documento, unindo cadeias temporais auxiliares à cadeia principal e enviando comandos com as ações gravadas na cadeia temporal principal. Os segmentos de mídia apresentam os conteúdos além de executar as ações e reconhecer a ocorrência de pontos de sincronização imprevisíveis, que são sinalizados ao tratador. A Figura 29 ilustra a arquitetura do executor do formatador Firefly.

O sistema de apresentação inicia a exibição do documento quando recebe do compilador a cadeia temporal principal e as auxiliares, disparando um relógio do documento. O relógio é incrementado até que a exibição termine naturalmente, ou até que um comando de pausa ou de término forçado seja enviado pelo usuário. Para os documentos suspensos, o tempo do relógio permanece constante, até que o usuário reassuma a exibição. O sistema de apresentação envia comandos com as ações especificadas na cadeia principal, quando o relógio atinge o tempo calculado para os pontos de sincronização previsíveis.

O sistema também permite que qualquer ponto de sincronização imprevisível em um segmento de mídia seja habilitado ou inibido. Quando um segmento de mídia detecta a ocorrência de um ponto de sincronização imprevisível, ele o notifica ao tratador de pontos

de sincronização, que mantém uma lista dos pontos habilitados e inibidos. Se o ponto notificado estiver habilitado, ele é passado ao sistema de apresentação, para que a cadeia temporal auxiliar com início nesse ponto de sincronização seja unida à cadeia principal. Antes da união, o sistema de apresentação calcula tempos absolutos para os pontos de sincronização da cadeia auxiliar, que originalmente possuíam valores relativos.

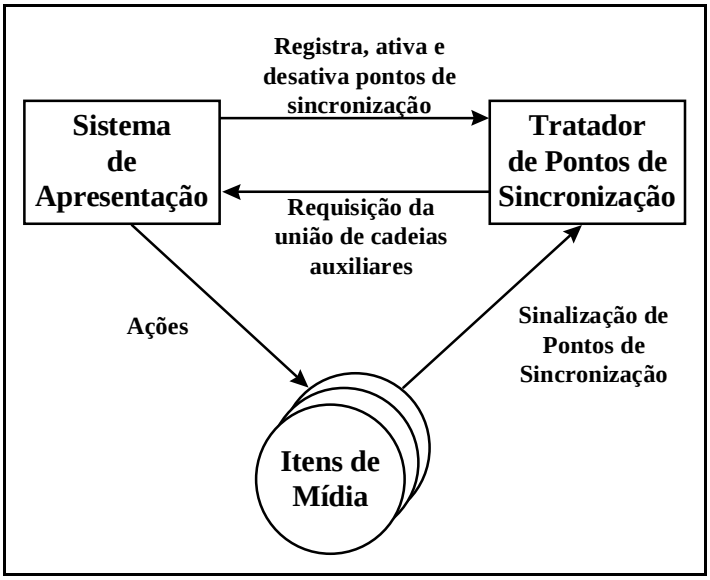


Figura 29 - Executor do Formatador Firefly

A solução adotada para atribuição dos tempos esperados no compilador do sistema Firefly é limitada, uma vez que só é computada uma solução (de menor custo) para um conjunto de restrições. Para encontrar uma outra solução, que pode existir, é preciso que os custos de elasticidade sejam alterados e o problema seja novamente resolvido. Como é o compilador, e não o executor, que realiza o algoritmo de elasticidade, não há como corrigir assincronismos devido a eventos imprevisíveis durante a exibição.

5.2 CMIFed

CMIFed [RJMB93, HaRB93] é um editor/exibidor desenvolvido para autoria e apresentação de documentos multimídia/hipermídia em ambientes distribuídos. Os documentos tratados pelo sistema são definidos segundo o modelo hipermídia CMIF (CWI Multimedia Interchange Format) [BuRL91].

O CMIF define eventos como a exibição do nó. Os eventos têm sempre duração previsível e determinada, não podendo ser especificada por um intervalo. Porque os eventos são definidos como a apresentação do nó por inteiro, o sistema apresenta granulosidade grossa, não sendo possível especificar pontos de sincronização que não sejam o início ou fim de apresentação do nó, nas restrições temporais definidas pelas composições, como será explicado mais adiante. No entanto, um evento pode possuir marcações em seu interior, onde alguns tipos de elos podem ser ancorados, permitindo, em algumas especificações, uma granulosidade mais fina.

O CMIF apresenta a idéia de composições na definição das apresentações. O sistema define uma *apresentação atômica* como um conjunto de eventos, e uma composição como sendo composta por uma ou mais apresentações atômicas e, opcionalmente, outras composições, podendo esse aninhamento ser feito em qualquer nível de profundidade. As composições definem o relacionamento temporal de seus componentes, podendo ser de dois tipos: paralelas, onde todos os componentes começam a ser exibidos juntos, ou seqüenciais, onde um componente é exibido após o outro. O principal problema na definição das composições feitas pelo CMIF é que as mesmas são sempre hierárquicas, especificando relacionamentos apenas entre eventos de exibição. Além disso, o sistema só permite a especificação dos dois tipos de relacionamentos citados (*meet*- seqüencial e *start*- paralelo).

O modelo também define elos para descrição da apresentação. Os elos são sempre 1:1, definidos entre marcações nos eventos (âncora), e podem ser de dois tipos: hiper-elos ou sinc-elos. Os hiper-elos definem os relacionamentos hipermídia tradicionais, possibilitando a interatividade do usuário. Os elos de sincronização (denominados sync arcs) são uma outra forma, além das composições, de especificação da sincronização temporal da apresentação. Eles são sempre definidos entre duas marcações contidas em eventos de uma mesma apresentação atômica e especificam um retardo, além de um desvio permissível para o retardo. Um tipo especial de arco, denominado *continuous sync arc*, determina que a variação de retardo especificada deve ser mantida durante toda a apresentação concorrente dos eventos. Esse tipo especial de arco é definido a fim de permitir sincronizações do tipo lip-sync, impossíveis de serem realizadas apenas com os sync arcs (a não ser que o usuário

coloque uma marcação em cada unidade de exibição dos eventos a serem sincronizados, e defina um sync arc para cada par de marcações).

Como trabalhos futuros, CMIF pretende colocar tempo máximo e mínimo nas restrições definidas por seus relacionamentos, o que não deve ser difícil para relacionamentos definidos em seus arcos, mas deve complicar nos relacionamentos definidos em suas composições. Note que, ao contrário do NCM, no CMIF existem relacionamentos definidos nas composições e outros definidos nos arcos. Quando definidos em arcos, não permitem um reuso estruturado.

O sincronismo espacial no CMIF pode ser especificado de maneira conjunta, onde um objeto, denominado *canal*, é definido, podendo ser compartilhado por mais de um evento. É como se diferentes objetos de representação utilizassem o mesmo descritor. O canal é uma abstração de um conjunto de propriedades compartilhadas por diversos eventos do mesmo tipo de mídia.

O formatador CMIFed, denominado *player*, é o elemento responsável pelo controle da apresentação do documento. A Figura 30 ilustra a interface de um documento apresentado no CMIFed e controlada pelo player. À esquerda, na janela inferior (Player - time-mgmt), é permitido o controle por parte do usuário da apresentação como um todo. Quando um usuário requisita a exibição de um documento, o formatador entra em uma fase de compilação, onde um grafo de dependências temporais é construído. No grafo, os nós representam pontos de sincronização e as arestas são relações temporais entre os pontos de sincronização. Inicialmente, o grafo é construído percorrendo a hierarquia do documento em profundidade, utilizando apenas as restrições temporais especificadas pelas composições. Num passo seguinte, arestas representando os relacionamentos especificados por sync arcs são acrescidos. Quando um arco de sincronismo é definido utilizando uma marcação que não seja o início ou fim de um evento, um novo nó e novas arestas ligando a marcação a outras marcações do mesmo evento são acrescidos ao grafo. Finalmente, dois nós especiais são definidos, o *start node* e o *end node*. O start node é conectado através de uma aresta, com duração nula, ao nó representando o início do primeiro evento a ser

exibido no documento. Da mesma forma, uma aresta também com duração nula liga o nó que define o fim do último evento ao end node.

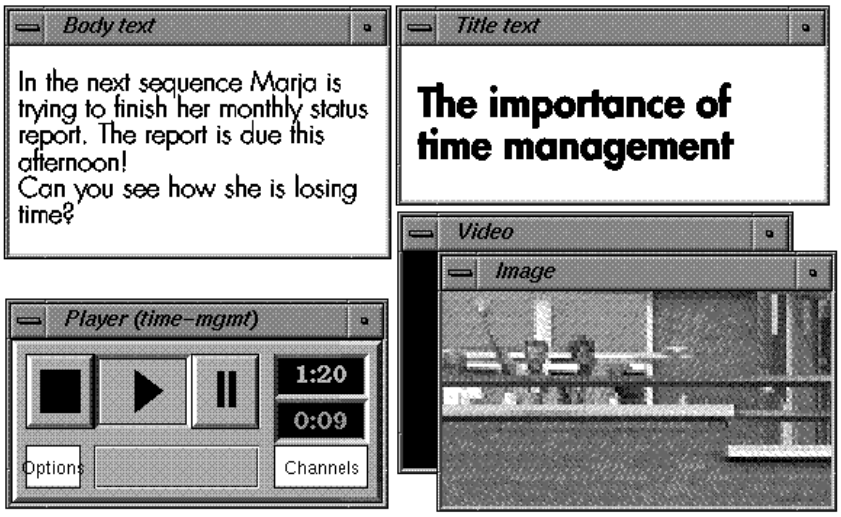


Figura 30 - Interface do executor CMIF

Construído o grafo, o formatador entra na fase de execução. Um relógio interno é disparado e a execução feita utilizando um algoritmo que percorre o grafo criado. Inicialmente, o start node é marcado. Quando um nó é marcado, todas as arestas que partem desse nó recebem o valor do relógio de execução no momento da marcação. Quando a duração especificada na aresta é completada, a aresta é marcada. Quando todas as arestas que chegam em um nó são marcadas, o nó é marcado. A execução termina quando o end node é marcado. Esse algoritmo não permite a implementação dos continuous sync arcs, que exigem um tratamento a parte, não sendo suportados pelo formatador na implementação atual. As características do grafo de dependências temporais são similares às de uma rede de Petri.

Ao simular uma execução e percorrer o grafo de dependências temporais o sistema é capaz de detectar erros na especificação temporal, através da detecção de caminhos fechados, e também conflitos por recursos. O grafo também permite que o formatador faça pré-busca do conteúdo dos objetos. Na implementação atual essa pré-busca é feita de maneira simplificada, utilizando os momentos em que o formatador encontra-se ocioso. A utilização de threads e heurísticas de busca para eventos gerados por pontos de sincronização

imprevisíveis, bem como negociações de QoS para busca dos conteúdos são melhorias que poderiam ser aplicadas ao algoritmo do sistema.

5.3 MODE

O sistema MODE (Multimedia Objects in a Distributed Environment) [BlHl92] também foi desenvolvido objetivando solucionar o problema de apresentação de documentos multimídia em ambientes distribuídos heterogêneos.

O sistema suporta nós de conteúdo, denominados objetos, porém não permite a definição de composições. Pontos de sincronização são denominados pontos de referência e podem ser definidos não só nas extremidades dos objetos (início e fim), mas também no interior dos mesmos. Na especificação, o sistema permite a definição de mais de uma alternativa de apresentação, associando uma qualidade a cada alternativa. Prioridades também podem ser definidas com o objetivo de determinar a sensibilidade do objeto a retardos que venham a ocorrer durante a exibição, para que ajustes possam ser aplicados. O sistema suporta a definição de relacionamentos temporais, onde dois ou mais pontos de sincronização podem estar associados a uma espécie de ponto de encontro (o sistema define esse elemento como ponto de sincronização). Esse ponto de encontro funciona como a tradicional transição na rede de Petri. Nenhum objeto pode continuar a sua exibição, até que todos os objetos associados ao ponto de encontro atinjam os seus respectivos pontos de sincronização.

O formatador do sistema é baseado em duas fases, uma de compilação e outra de execução. O compilador, denominado *optimizer*, recebe um layout de apresentação manualmente construído e uma especificação da plataforma de exibição. Não existe nenhuma interferência do formatador na construção do plano de exibição, apenas o compilador utiliza a especificação da plataforma a fim de escolher uma qualidade de apresentação para ser utilizada na exibição do documento. O executor, denominado *synchronizer*, controla as apresentações dos objetos, permitindo que os mesmos ajustem em tempo de exibição as suas durações, a fim de atingir de forma mais acelerada o ponto de encontro. O executor também implementa meta-diretivas geradas pela interação do usuário com os objetos.

O sistema não suporta qualquer estruturação lógica do documento (não suporta a definição de composições) e não permite a definição de relacionamentos baseados no conceito hipermídia (interatividade do usuário de maneira completamente imprevisível). Como principal ponto crítico na formatação, está a não disponibilidade de mecanismos que implementem a construção automática do plano de execução, o que exige uma construção manual por parte do autor. Em compensação, a idéia de pontos de encontro aliados à flexibilidade na especificação permitem um ajuste da taxa de exibição dos objetos em fase de execução.

5.4 I-HTSPN

I-HTSPN [WSSD96] é um modelo conceitual baseado em redes de Petri para definição de documentos hipermídia. O modelo é uma extensão a outro modelo também baseado em redes Petri, denominado HTSPN [SéSW95], que, por sua vez, é uma extensão ao modelo TSPN, explicado na Seção 3.3. HTSPN (Hierarchical Time Stream Petri Net) é definido com o objetivo de melhorar a estruturação durante a fase de autoria dos documentos hipermídia. A idéia é definir lugares especiais que podem ser explodidos em outras redes, criando uma estrutura semelhante às das composições, a fim de diminuir a quantidade de lugares e transições visíveis durante a fase de especificação do autor. Um lugar de composição e sua subrede relacionada devem ser não apenas estruturalmente equivalentes (isto é, a subrede deve ter um lugar de entrada e um lugar de saída), mas também temporalmente equivalentes. No entanto, apesar de relacionamentos definidos no modelo poderem referenciar composições, não é possível referenciar elementos que estejam contidos nessas composições.

I-HTSPN (Interpreted version of HTSPN) é uma extensão ao modelo HTSPN, onde são definidos elementos que permitem a definição da sincronização espacial e de parâmetros para acesso à informação armazenada, questões que não eram suportadas pelos modelos anteriores.

Conforme apresentado na Seção 3.3, na TSPN um evento é representado por um lugar na rede e pela associação de uma tupla [t_{min}, t_{ot}, t_{max}] ao arco que sai do lugar, onde a tupla

$[t_{min}, t_{max}]$ especifica o intervalo possível de duração do evento, e t_{ot} o tempo de duração com a melhor qualidade de apresentação. Na TSPN, retardos podem ser introduzidos através da adição de lugares com arcos de partida associados a um intervalo de possível ocorrência de retardo. Um hiper-elo (elo hipermídia) é representado por um novo tipo de lugar, tendo a tupla $[t_{min}, *, t_{max}]$ associada ao arco de saída do lugar o significado usual, representando o intervalo possível de disparo do hiper-elo (note que t_{max} pode ser ∞), onde o instante do disparo é indeterminado (símbolo *).

[WSSD96] descreve uma ferramenta que oferece um ambiente para autoria dos documentos em rede de Petri. O sistema possui um pré-compilador, denominado *analyser*, que realiza a análise temporal do documento e detecta conflitos na utilização de recursos durante a especificação. Um outro módulo, denominado *MHEG translator*, gera a partir da especificação I-HTSPN uma representação MHEG. Se forem detectados erros na especificação que impossibilitem a tradução para o modelo MHEG, estes são reportados ao editor I-HTSPN. Uma máquina MHEG será a responsável por controlar a apresentação do documento. Sendo assim, o formatador I-HTSPN apenas apresenta as fases de pré-compilação e compilação, sendo a máquina MHEG a responsável por implementar a fase de execução. A Figura 31 apresenta o esquema de funcionamento da ferramenta I-HTSPN.

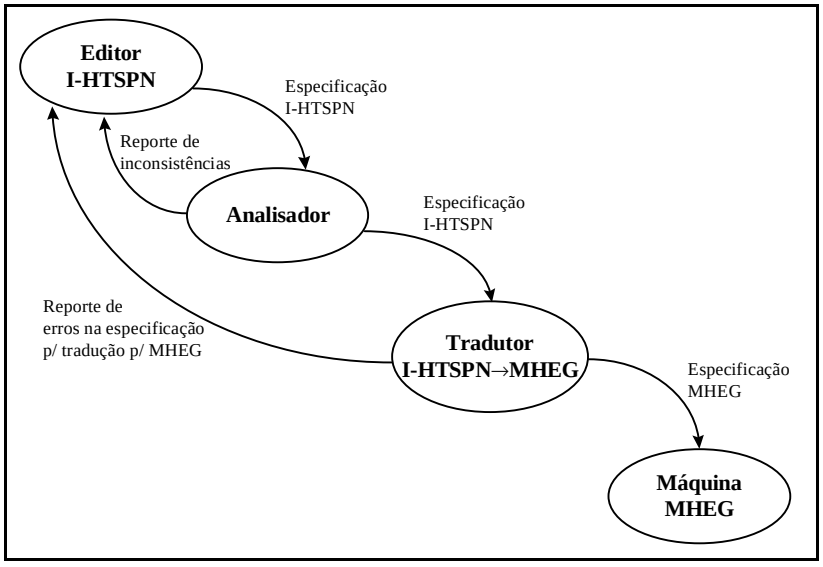


Figura 31 - Esquema de funcionamento da ferramenta I-HTSPN

Ao especificar um documento em I-HTSPN e depois traduzi-lo para o modelo orientado a objetos do padrão MHEG, pode ser que a máquina MHEG gere uma outra estrutura de dados, talvez mesmo, baseada em redes de Petri, voltando a uma estrutura próxima à primeira.

5.5 Comparação Final

Esta seção apresenta um quadro comparativo entre os sistemas descritos neste capítulo e o próprio sistema HyperProp. Os sistemas são comparados baseados nas características apresentadas no Capítulo 2, sendo divididos em três tabelas, uma para cada fase de formatação (pré-compilação, compilação e execução). As características dos sistemas apresentados nesta seção são baseadas não somente em elementos implementados, mas também nas potencialidades de implementação futura. Apenas no sistema HyperProp, as características presentes na atual implementação e previstas para uma próxima versão são separadas em duas colunas. Para os sistemas que não apresentarem a fase de formatação correspondente à tabela, a coluna respectiva ao sistema encontra-se sombreada.

Fase de Pré-Compilação						
Características	**Firefly**	**CMIFed**	**MODE**	**I-HTSPN**	**HyperProp**	
					atual	**futura**
Flexibilidade						
inflexível					●	
ajustável				●		●
seleciona alternativas				●	●	●
Mudanças de Comportamento						
Especificações de autoria				●	●	●
Inconsistência						
Temporal				●	●	●
Plataforma						
Conflito				●	●	●

Tabela 9 - Comparação entre as características da fase de pré-compilação dos formatadores analisados

Fase de Compilação						
Características	Firefly	CMIFed	MODE	I-HTSPN	HyperProp	
					atual	futura
Flexibilidade						
inflexível		●			●	
ajustável	●			●		●
seleciona alternativas			●	●	●	●
Mudanças de Comportamento						
Especificações de autoria	●			●	●	●
Inconsistência						
Temporal	●	●		●		●
Plataforma			●	●		●
Conflito			●	●		●

Tabela 10 - Comparação entre as características da fase de compilação dos formatadores analisados

Fase de Execução						
Características	Firefly	CMIFed	MODE	I-HTSPN	HyperProp	
					atual	futura
Flexibilidade						
inflexível	●	●			●	
ajustável			●			●
seleciona alternativas					●	●
Relações Temporais						
previsíveis	●	●	●		●	●
imprevisíveis	●	●			●	●
Mudanças de Comportamento						
Especificações de autoria	●				●	●
Meta-Diretivas	●	●	●		●	●
Inconsistência						
Temporal						●
Plataforma						●
Conflito						●
Variações Imprevisíveis						
não suporta	●	●			●	
suporta			●			●

Tabela 11 - Comparação entre as características da fase de execução dos formatadores analisados

Nas composições reside uma das diferenças mais fundamentais entre o modelo NCM e os modelos CMIF e HTSPN. Esses últimos partem do paradigma que as composições são utilizadas apenas para a estruturação da apresentação dos documentos, ou que essa estruturação se confunde com a estruturação lógica do documento, o que nem sempre é verdade. Nestes modelos não é possível, por exemplo, definir uma composição *capítulo A*, que pode, sob a intervenção do leitor pelo disparo de um evento de seleção, acionar outra composição *capítulo B*, que não está contida em A. Para tanto, componentes dos *capítulos A e B* deveriam estar definidos em uma mesma composição, perdendo assim a estruturação lógica. No NCM as composições representam, de fato, uma estruturação lógica do documento.

Por ser uma abstração de mais alto nível que os grafos de dependências temporais e redes de Petri, os modelos orientados a objetos permitem uma interface com o usuário muito mais amigável. Ao contrário do que afirma Willrich [WSSD96], não é irreal descrever documentos hipermídia usando esses modelos, como o apresentado no padrão MHEG. Basta que seja construída uma boa interface com o usuário. Se assim for feito, a expressividade muito maior de tais modelos pode ser passada de modo fácil e compreensível. Além de tudo, outras facilidades podem ser facilmente agregadas ao modelo, como controle de versões, facilidades para trabalho cooperativo, entre outras [SoCR95], que seriam de difícil incorporação nos outros modelos. Modelos orientados a objetos, tais como o utilizado no HyperProp, podem também ser mais facilmente traduzidos para o padrão de intercâmbio de objetos multimídia/hipermídia definido no MHEG, e explorando toda sua potencialidade. Em outras palavras, advoga-se aqui a utilização de um modelo orientado a objetos como o resumidamente descrito para o sistema HyperProp, para a especificação do documento. Deste modelo, será extraída a estrutura que alimentará o formatador temporal e, aí sim, por estar mais perto da máquina, e não do usuário, um modelo de mais baixo nível para máquinas de estado paralelas pode ser o mais adequado.

Capítulo 6

Conclusão

A tarefa de controle da apresentação de documentos multimídia/hipermídia é complexa, principalmente devido às características não convencionais de alguns de seus componentes, como áudio e vídeo. Como conseqüência, torna-se necessária a construção de um ambiente de execução capaz de gerenciar a exibição, garantindo que a especificação do autor, sempre que possível, seja respeitada.

Para tornar possível o controle da apresentação dos documentos, o ambiente de execução deve possuir um plano de execução que lhe permita saber que ações executar quando da ocorrência de eventos. A criação desse plano de execução e o controle da apresentação consistem no posicionamento dos objetos constituintes do documento no tempo e no espaço, exigindo a implementação de algoritmos para que essa formatação temporal e espacial seja feita de maneira automática. Por esse motivo, o ambiente de execução é também denominado formatador temporal e espacial, e seus algoritmos de formatação devem ser desenvolvidos levando em consideração os não determinismos, permitindo a correção da apresentação no caso de ocorrência de eventos imprevisíveis (por exemplo, retardos variáveis na rede, diretivas do usuário etc.), e tirando proveito das definições das qualidades de serviço exigidas por cada componente do documento, em conjunto com as características da plataforma de exibição (por exemplo, realização de pré-busca do conteúdo dos objetos).

Nos formatadores de sistemas multimídia/hipermídia algumas características podem ser analisadas, sendo elas: o momento em que a construção do plano de execução é feita, a flexibilidade na construção, os tipos de relacionamentos temporais existentes, as mudanças de comportamento previstas, os tipos de inconsistência verificados e as alterações imprevisíveis suportadas. Essas características definem para a arquitetura do formatador quatro elementos básicos: o pré-compilador, o compilador, o executor e os controladores de exibição. De acordo com as características e os elementos que um determinado formatador implementa, a especificação de apresentação torna-se mais ou menos expressiva.

Pela exposição é possível observar que as tarefas do formatador temporal e espacial são bem próximas das tarefas de um sistema operacional de tempo real e distribuído. Pode ser assim uma tendência que, um dia, o formatador dos sistemas multimídia/hipermídia seja parte integrante dos sistemas operacionais.

6.1 Contribuições da Dissertação

As principais contribuições desta dissertação foram: definir as características de um formatador temporal e espacial para apresentação de documentos multimídia/hipermídia, especificar o formatador temporal e espacial para o sistema HyperProp, e implementar o primeiro protótipo do formatador deste sistema.

A definição de formatadores temporais e espaciais para apresentação de documentos multimídia/hipermídia englobou a descrição dos requisitos de formatação para apresentação de documentos multimídia/hipermídia com restrições temporais, a descrição das características que podem estar presentes em formatadores, com suas possíveis formas de implementação, e o detalhamento de uma arquitetura genérica para os formatadores.

A especificação do formatador temporal e espacial do sistema HyperProp definiu a arquitetura interna do formatador, com a descrição de cada um dos elementos componentes (pré-compilador, compilador, executor e controladores de exibição) a partir de suas funções básicas. Foram caracterizadas as informações a serem trocadas internamente entre os módulos do formatador, e entre o formatador e os demais elementos do sistema. A interface

entre o executor e os controladores de exibição, assim como a interface entre compilador/executor e o ambiente de armazenamento (servidor de dados NCM) foram completamente especificadas. Ainda como parte da definição do formatador do sistema HyperProp, foi feita a modelagem da estrutura de dados interna para controle da apresentação dos documentos.

A implementação do protótipo do formatador para o sistema HyperProp foi realizada sobre plataforma UNIX utilizando linguagem C++. Foram desenvolvidos os módulos compilador e executor do formatador, o suporte para troca de mensagens entre o executor e os controladores de exibição, e entre o compilador/executor e o servidor de dados NCM. Foi feita a integração entre o formatador e o subsistema de autoria, composto pelo editor de sincronismo [Cost96] e o editor de estrutura [Much96]. Para efeito de testes, foi desenvolvido um controlador de exibição para um exibidor simplificado, a fim de simular a apresentação dos documentos multimídia/hipermídia criados no sistema HyperProp.

6.2 Trabalhos Futuros

Dentre as principais características a serem incorporadas, em trabalhos futuros, ao formatador temporal e espacial do sistema HyperProp estão: a gravação das cadeias temporais geradas pelo pré-compilador, estudo de algoritmos de elasticidade para as fases de pré-compilação, compilação e execução, o estudo de heurísticas de carregamento prévio do conteúdo dos objetos, e o desenvolvimento de mecanismos para testes de consistência temporal e espacial nas fases de compilação e execução.

Para implementar a gravação das cadeias temporais parciais calculadas na fase de pré-compilação, é preciso decidir como armazená-las. Uma opção seria colocá-las dentro dos respectivos contextos que possuem os elos cujos eventos geraram a cadeia. A outra opção seria definir as cadeias como uma estrutura a parte do modelo, estando associada a um determinado contexto.

O estudo de algoritmos de elasticidade para as fases de pré-compilação, compilação e execução está relacionado com as características de flexibilidade do formatador, correspondendo ao algoritmo utilizado na escolha dos tempos esperados para os pontos de

sincronização, e nos possíveis ajustes feitos nas taxas de apresentação das mídias. Algumas opções citadas na literatura englobam programação linear, programação dinâmica, ordenação topológica, caminho mais curto, entre outras.

O pré-carregamento é uma tarefa importante de ser executada pelo formatador, a fim de minimizar a probabilidade de atrasos na apresentação das mídias. Cabe estudar heurísticas para que essa busca seja feita de maneira automática, bem como determinar se essa tarefa será desempenhada pelo executor ou pelos próprios controladores de exibição.

Por fim, é importante que assim como na fase de pré-compilação o formatador disponha de algoritmos eficientes para verificar a consistência temporal e espacial do documento, em tempo de compilação e execução. O estudo e implementação desses algoritmos são também colocados como pesquisas futuras.

As Tabelas 9, 10 e 11, apresentadas no Capítulo 5, resumem, respectivamente, as características implementadas nas fases de pré-compilação, compilação e execução da versão atual do sistema HyperProp, comparando com as características previstas de serem suportadas pelo sistema em uma implementação futura.

Referências Bibliográficas

[Alle83] Allen, J.F.; Maintaining Knowledge About Temporal Intervals, *Communications of the ACM*, vol. 26, No 11, Novembro de 1983, pp. 832-843.

[BlHL92] Blakowski, G.; Hubel, J.; Langrehr, U.; Tool Support for the Synchronization and Presentation of Distributed Multimedia, *Computer Communications*, vol. 15, No 10, Dezembro 1992, pp. 611-618.

[Bolo90] Bolognesi, T.; From Timed Petri Nets to Timed LOTOS, *Proceedings of the IFIP Conference on Protocol Specification*, Testing and Verification, Ottawa, North Holland, Junho 1990.

[Booc94] Booch, G.; Object Oriented Analisys and Design with Applications; second edition; Addison Wesley, 1994.

[BuRL91] Bulterman, D.; van Rossum, G.; Liere, R.; A Structure for Transportable Dynamic Multimedia Documents, *USENIX conference*, Nashville, Junha 1995, pp. 137-155.

[BuZe92] Buchanan, M.C.; Zellweger, P.T.; Specifying Temporal Behavior in Hypermedia Documents, *Proceedings of European Conference on Hypertext*, ECHT'92, Milano, Dezembro 1992.

[BuZe93a] Buchanan, M.C.; Zellweger, P.T.; Automatically Generating Consistent Schedules for Multimedia Documents, *Multimedia Systems*, Springer-Verlag, Abril 1993.

[BuZe93b] Buchanan, M.C.; Zellweger, P.T.; Automatic Temporal Layout Mechanisms, *Proceedings of ACM Multimedia'93*, Anaheim, California, Agosto 1993. pp. 341-350.

[CMSS96] Costa, F.R.; Muchaluat, D.C.; Soares, L.F.G.; Souza, G.L.; Editor Gráfico para Estrutura e Sincronismo de Documentos Multimídia, *Anais do IX SIBGRAPI*, Caxambu, Outubro 1996, pp. 289-296.

[CSCS96] Colcher, S.; Soares, L.F.G.; Casanova, M.A.; Souza, G.L.; Modelo de Objetos Baseado no CORBA para Sistemas Hipermídia Abertos com Garantias de Sincronização, *Anais do XIV SBRC*, Fortaleza, Maio 1996. pp. 18-37.

[Cost96] Costa, F.R.; Um Editor Gráfico para Definição e Exibição do Sincronismo de Documentos Multimídia/Hipermídia, *Dissertação de Mestrado*, Departamento de Informática PUC-Rio, Rio de Janeiro, Agosto 1996.

[DiSé94] Diaz, M.; Sénac, P.; Time Stream Petri Nets, a Model for Timed Multimedia Information, *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, Zaragoza, 1994, pp. 219-238.

[DrGr93] Drapeau, G.D.; Greenfield, H.; Synchronization in the MAEstro Multimedia Authoring Environment, *Parallax Graphics, Inc.*, Santa Clara, California, 1993.

[FiTD87] Fiume, E.; Tsichritzis, D.; Dami, L.; A Temporal Scripting Language for Object-Oriented Animation, *Proceedings Eurographics'87*, Elsevier Science Publishers, North-Holland Publishing Company, Amsterdam, 1987.

[HaRB93] Hardman, L.; van Rossum, G.; Bulterman, D.C.A.; Structured Multimedia Authoring, *Proceedings of ACM Multimedia'93*, Anaheim, California, Agosto 1993, pp. 283-289.

[HaSc90] Halasz, F.; Schwartz, M.; The Dexter Hypermedia Reference Model, *NIST, Hypertext Standardization Workshop*, Gaithersburg, MD, Janeiro 1990.

[KiSo95] Kim, M.Y.; Song J.; Multimedia Documents with Elastic Time, *Proccedings of ACM Multimedia'95*, San Francisco, California, Novembro 1995, pp. 143-154.

[LiGh90] Little, T.; Ghafoor, A.; Synchronization and Storage Models for Multimedia Objects, *IEEE J. Selected Areas of Commun.*, vol. 8, No. 3, Abril 1990, pp. 413-427.

[Macr89] MacroMind Director: Overview Manual, *MacroMind Inc.*, Março 1989.

[MHEG95] Multimedia and Hypermedia Information Coding Expert Group (MHEG), ISO/IEC DIS 13522-1 - coded Representation of Multimedia and Hypermedia Information (MHEG), Part I: MHEG Object Representation, Base Notation (ASN.1), Setembro 1995.

[Much96] Muchaluat, D.C.; Browsers e Trilhas para Documentos Hipermídia Baseados em Modelos com Composições Aninhadas, *Dissertação de Mestrado*, Departamento de Informática PUC-Rio, Rio de Janeiro, Março 1996.

[RJMB93] van Rossum, G.; Jansen, J.; Mullender, K.S., Bulterman D.; CMIFed: A Presentation Environment for Portable Hypermedia Documents, *Proceedings of ACM Multimedia'93*. Anaheim, California, Agosto 1993.

[SéSW95] Sénac, P.; Saqui-Sannes, P.; Willrich R.; Hierarchical Time Stream Petri Net: a Model for Hypermedia Systems, *Application and Theory of Petri Nets*, 1995, pp. 451-470.

[SoCC93] Soares, L.F.G.; Casanova, M.A.; Colcher, S.; An Architecture for Hypermedia Systems using MHEG Standards Objects Interchange, *Information Services & Use*, vol.13, No.2, IOS Press. Amsterdam, The Netherlands. 1993; pp. 131-139.

[SoCR95] Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.L.R.; Nested Composite Nodes and Version Control in an Open Hypermedia System, *IEEE Transaction on Information Systems*, vol. 20, No. 6. Elsevier Science Ltd, Great Britain. 1995; pp. 501-519.

[SoRo97] Soares, L.F.G.; Rodrigues, R.F.; Autoria e Formatação Estruturada de Documentos Hipermídia com Restrições Temporais, *III Workshop em Sistemas Multimídia e Hipermídia*, São Carlos, Maio de 1997.

[SoSC95] Souza, G.L.; Soares, L.F.G.; Casanova M.A.; Synchronization Aspects of an Hypermedia Presentation Model with Composite Nodes, *Proceedings of the ACM Worshop on Effective Abstractions in Multimedia, in connection with ACM Multimedia 95*, São Francisco, EUA, Novembro 1995.

[SoSo97] Souza, G.L.; Soares, L.F.G.; Modelo de Contextos Aninhados Segunda Versão, *Relatório Técnico do Laboratório TeleMídia*, Departamento de Informática da PUC-Rio, Rio de Janeiro, Abril 1997.

[StNa95] Steinmetz, R. and Nahrstedt, K.; Multimedia: Computing, Communications and Applications, *Prentice Hall*, Inc. 1995.

[Walt83] Walter, B.; Timed Petri Nets for Modeling and Analysing Protocols with Real-Time Characteristics, *Proceedings of the 3rd IFIP Conference on Protocol Specification, Testing and Verification*, North Holland, 1983.

[WoQG94] Woo, M.; Qazi, N.; Ghafoor, A.; A Synchronization Framework for Communication of Pre-Orchestrated Multimedia Information, *IEEE Network*, 1994.

[WSSD96] Willrich, R.; Sénac, P.; Saqui-Sannes, P.; Diaz, M.; Towards Hypermedia Documents Design, *Anais do XIV SBRC*, Fortaleza, Maio 1996, pp. 473-491.