Guido Lemos de Souza Filho

SINCRONISMO NA MODELAGEM E EXECUÇÃO DE APRESENTAÇÕES DE DOCUMENTOS MULTIMÍDIA

TESE DE DOUTORADO

DEPARTAMENTO DE INFORMÁTICA

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Rio de Janeiro, 29 de setembro de 1997

Guido Lemos de Souza Filho

SINCRONISMO NA MODELAGEM E EXECUÇÃO DE APRESENTAÇÕES DE DOCUMENTOS MULTIMÍDIA

Tese apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Doutor em Informática: Ciência da Computação.

Orientador: Luiz Fernando Gomes Soares

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 29 de setembro de 1997

Este trabalho é dedicado

à minha esposa e amiga, Eliauria;

aos meus filhos Caio e Luísa;

aos meus pais Guido Lemos de Souza e Margarida M. de A. L. de Souza, por todo o amor que me dedicaram.

AGRADECIMENTOS

- Ao Amigo, Professor e Orientador Luiz Fernando Gomes Soares pela dedicação, paciência e confiança investidos em minha formação.

- Aos colegas do Grêmio Recreativo Escola de Samba Futebol e Regatas Telemídia pelo ambiente de amizade, cooperação e alegria que tornou tão agradável a realização deste trabalho.

- A Fábio Costa, Débora Muchaluat, Rogério Rodrigues e Flávio Amorim, pelo empenho e trabalho fundamental para a realização desta tese.

- A toda a comunidade da Pontifícia Universidade Católica do Rio de Janeiro, mais especialmente ao Departamento de Informática através de seus professores, funcionários e alunos pelo apoio técnico e administrativo, disponibilidade, amizade e incentivo.

- Aos colegas da UFRN, em especial ao Prof. Pedro Maia, pelo apoio e confiança.

- Aos membros do grupo OLC do LAAS/CNRS, em especial ao Prof. Jean-Pierre Courtiat, pela acolhida e apoio.

- Ao CNPq que, através do projeto ProTeM HyperProp, viabilizou financeiramente a realização deste trabalho.

- À CAPES pelo apoio financeiro durante parte do período de doutoramento.

Resumo

A complexidade lógica, temporal e semântica de documentos multimídia e hipermídia torna necessário o uso de modelos conceituais que permitam sua especificação e manipulação. A expressão das idéias no documento implica em uma ordenação lógica de fragmentos de informação representada em diferentes mídias. Esta ordenação define o tempo e o espaço que deve ser ocupado pela apresentação de cada um dos componentes. Este trabalho propõe um modelo conceitual para documentos multimídia denominado Modelo de Contextos Aninhados (NCM). Quando se iniciou o trabalho, o NCM já havia sido objeto de outros estudos que concentraram-se nos aspectos estruturais dos documentos, assim, a principal contribuição deste tese foi propor estruturas que permitem capturar especificações do sincronismo temporal e espacial nas definições de documentos estruturados como composições aninhadas. A especificação do sincronismo temporal é realizada com base em duas estruturas: eventos e elos. Sendo a semântica da apresentação dos documentos definida pela ocorrência de uma coleção de eventos alinhados no tempo através dos elos. A especificação do sincronismo espacial é capturada por uma entidade denominada descritor que separa a especificação das características de apresentação, dos dados a serem apresentados. Esta facilidade torna possível definir apresentações contextualizadas para documentos ou mesmo para componentes de documentos. Com o intuito de validar o modelo elaborado, foram definidos e implementados ambientes que permitem a edição a execução de apresentações de documentos NCM.

Abstract

Logical, temporal and semantic complexity of multimedia and hypermedia documents claims for a conceptual model permitting its specification and manipulation. Expression of ideas through documents presupposes a logical ordering of information fragments represented through different media. This ordering defines the time and space occupied by the presentation of each component. This work proposes a multimedia document conceptual model called Nested Context Model (NCM). When this work started, NCM had been object of other studies focused on structural aspects of documents. So, the main contribution of this thesis was to propose entities allowing to capture specifications of temporal and spatial synchronism in order to define documents structured as nested compositions. The specification of the presentation's temporal synchronism is based in two entities: events and links. The occurrence of a collection of events aligned in time by links defines the semantic of documents' presentations. Spatial synchronism specification is captured by an entity called descriptor, which keeps the specification of the presentation characteristics separated from the data to be presented. This facility allows the definition of alternative presentations for documents as well as for its components. In order to validate the proposed model, tools allowing the edition and execution of NCM documents presentations were developed.

Índice

Figuras

Tabelas

Capítulo 1

Introdução

1.1. Motivação

O desenvolvimento de dispositivos de baixo custo para manipulação de imagem estática, áudio e vídeo, e sua incorporação à configuração dos computadores pessoais, tornaram viável a integração dessas mídias ao texto, até então componente exclusivo dos documentos. Assim, os documentos manipulados em computadores evoluíram de documentos monomídia (compostos exclusivamente por informações textuais) para documentos multimídia; isto é, composições de fragmentos de informação representada em diferentes mídias, inicialmente contendo textos e imagens estáticas e, posteriormente, contendo também trechos de áudio, vídeo etc. A composição de fragmentos de informação multimídia implica na necessidade de definir como estes fragmentos (componentes da composição) relacionam-se uns com os outros. Assim, um documento multimídia pressupõe uma estrutura que captura os relacionamentos entre seus componentes. De fato, uma definição completa de um documento multimídia deve incluir: uma descrição de cada um dos componentes do documento, uma descrição da estrutura do documento e, ainda, uma descrição da forma como os vários componentes devem ser apresentados para os seus leitores.

Um modelo conceitual de documentos define mecanismos que permitem descrever documentos. As referências [Hoep94, MeRT91, Karm93, HaSc90], pelos motivos citados no parágrafo anterior, advogam que a descrição dos documentos multimídia seja dividida em três partes: *estrutura lógica*, que descreve as diferentes partes lógicas dos documentos, ou componentes, e as relações lógicas entre eles; *estrutura da apresentação*, descrevendo onde (em que região de qual dispositivo) e quando os componentes do documento devem

ser apresentados; e *estrutura dos conteúdos*, onde são descritas as informações que constituem os componentes do documento. A divisão da descrição do documento nessas três estruturas apresenta uma série de vantagens [Fluc95], sendo as principais delas as seguintes:

- A divisão da descrição do documento em estrutura lógica e estrutura dos conteúdos permite que conteúdos sejam reutilizados em diferentes pontos da estrutura de um documento e em documentos distintos.
- A separação entre a estrutura da apresentação e a estrutura dos conteúdos permite que os mesmos conteúdos sejam apresentados de diferentes formas; por exemplo, um texto pode ser apresentado graficamente por um editor de textos ou como áudio produzido por um sintetizador de som.
- A separação entre a estrutura lógica e de apresentação torna possível a reutilização da estrutura lógica em apresentações diferentes do documento, modificando-se apenas a estrutura da apresentação; por exemplo, para adaptar a apresentação do documento a ambientes com dispositivos ou QoS (Qualidade de Serviço) diferentes, basta modificar a descrição da apresentação do documento.

Do exposto acima, conclui-se então que um modelo conceitual de documentos deve representar os conceitos estruturais dos dados, os eventos e os relacionamentos entre os dados, assim como definir regras de estruturação e operações sobre os dados para manipulação e atualização das estruturas.

Em 1945, Vannevar Bush, no artigo “As We May Think” [Bush45], propôs um dispositivo chamado *memex* cujo objetivo era armazenar fragmentos de informação que podiam ser associados entre si. A idéia básica era permitir que o leitor pudesse pular de um item enfocado para outro sugerido por uma associação. A essa estrutura proposta por Bush, Nelson denominou hipertexto [Nels65].

Entre várias definições, pode-se definir um hipertexto como um texto organizado de forma não linear ou não seqüencial. Os fragmentos de texto correspondem a nós, conectados entre si por elos, associados a pontos de ancoragem (regiões marcadas) dentro dos nós. O leitor navega por um hipertexto de forma não seqüencial, selecionando âncoras e seguindo os

respectivos elos de um nó para outro. Um documento com a estrutura de um hipertexto cujos nós contêm outros tipos de mídia, além de texto, é um documento hipermídia. Nesses documentos,[1] ao navegar para um nó, o usuário poderá escutar um trecho de áudio ou assistir a uma cena de um vídeo da mesma forma que vê os textos contidos em nós de hipertextos. Estas características do paradigma hipermídia, ao permitirem organizar e definir relacionamentos entre os fragmentos de informação, o qualificam para orientar a definição de modelos conceituais para documentos multimídia.

A possibilidade de representar referências arbitrárias entre partes quaisquer de um documento representa, para muitos, a característica marcante de um sistema hipertexto. No entanto, é necessário combinar esta flexibilidade com mecanismos de organização do documento, a fim de evitar o problema conhecido como "desorientação do usuário" [Hala88], com mecanismos para definição de diferentes visões de um mesmo documento, com mecanismos para organização hierárquica (linear ou não) de documentos e com mecanismos para permitir a definição das relações de estruturação do documento, independente do conteúdo de dados do documento, de forma a permitir o reuso dos dados sem a herança obrigatória das relações, como acontece na maioria dos sistemas hipermídia atuais, exemplificados pelo WWW. Os modelos com nós de composição fornecem suporte a tais mecanismos, em especial modelos que permitem o aninhamento destes nós.

Ferramentas de apoio à tarefa de edição e visualização da estrutura de documentos devem fornecer informações de contexto para que usuários desorientados possam restabelecer a noção de localização [Much96]. Em particular dois tipos de contexto são necessários: contexto espacial, que responde à questão "para onde posso ir agora?", e contexto temporal, que responde à questão "como eu cheguei aqui?". Para fornecer aos usuários a informação de contexto temporal, duas técnicas têm sido utilizadas: registrar o caminho percorrido e permitir que o usuário possa voltar atrás neste caminho, denominado trilha. Para voltar a ler o documento sem ser forçado a recriar exatamente os mesmos passos através dos elos percorridos anteriormente, pode-se navegar pela trilha criada através da história do último caminho percorrido por cada leitor. Para resolver a questão do contexto

[1] A partir deste ponto, as ocorrências da palavra documento devem ser entendidas como documento multimídia/hipermídia.

espacial, a maneira mais direta é dar aos usuários um sentido de localização através de um mapa de todo o espaço de nós do documento e mostrar-lhes a sua posição, indicando de onde vieram e para onde podem ir. Entretanto, quando o número de nós e elos do hiperdocumento é grande, a apresentação do mapa, ao invés de melhorar a orientação do usuário, pode prejudicá-la ainda mais. Como já mencionado, alguns modelos conceituais permitem agrupar nós em nós de composição e também permitem composições aninhadas. Tais modelos, ao permitirem que grupos de nós sejam organizados, hierarquicamente ou não, facilitam a construção de mapas dos documentos, ajudando a reduzir o problema da desorientação do usuário.

A descrição de um documento envolve, além da definição de sua estrutura lógica e de seus componentes, a especificação da forma como ele deverá ser apresentado para os usuários. O termo *apresentação* é usado neste texto para denotar as atividades de exibição das várias mídias que compõem um documento ao usuário. Para imagens e texto, apresentação implica em exibição na tela. Para os componentes de áudio e vídeo, apresentação indica reprodução dinâmica audível e visual do dado. As apresentações de documentos, que serão aqui discutidas, são *interativas*, isto é, seus autores podem definir pontos onde poderão ser capturadas ações dos espectadores, determinando como as apresentações devem prosseguir.

Os conceitos de nó terminal (que armazena um fragmento de informação) e nó de composição (que armazena uma coleção de nós terminais e de composição, recursivamente), adequam-se aos requisitos de modelagem da estrutura lógica dos documentos. Para definir apresentações, os componentes dos documentos devem ser posicionados no espaço definido pelos dispositivos de entrada/saída dos computadores (sincronismo espacial) e no tempo (sincronismo temporal). Em muitas situações, o posicionamento dos componentes é relativo, ou seja, depende de outro(s) componente(s). Isto ocorre porque o espaço e o tempo ocupado por alguns dos componentes podem não ser conhecidos com precisão no momento da edição da apresentação (interações com usuários, reflexos de mudanças na velocidade da exibição quando uma apresentação é executada em máquinas diferentes etc.). Uma representação adequada para as apresentações precisa fornecer um mecanismo que capture esses relacionamentos de sincronismo entre os componentes. No paradigma hipertexto, a definição de relacionamentos entre componentes

cabe aos elos que, se adequadamente definidos, podem capturar também a noção de relacionamento de sincronismo temporal e espacial.

1.2. Contribuições

Dentre os principais objetivos do projeto HyperProp [Soar95], em cujo contexto está inserida esta tese, estão a definição de um modelo conceitual de documento que atenda aos requisitos listados na seção anterior, modelo este denominado Modelo de Contextos Aninhados — NCM (Nested Context Model), e mostrar sua viabilidade através da construção de um sistema com suporte para edição, exibição e armazenamento de documentos NCM [SoCC93]. As primeiras descrições do NCM se ativeram mais aos aspectos da estrutura de dados do modelo [Casa91] e das regras de estruturação lógica. As estruturas e operações para controle de versões foram assunto de uma especificação posterior [SoCR95].

A principal contribuição desta tese foi estender a versão do NCM descrita em [SoCR95], dotando o modelo de mecanismos para especificação da apresentação dos documentos [SoSo95, SoSC95, SoCS96a, SoCS96b]. Mais especificamente, as principais contribuições ao modelo neste aspecto foram:

- Definição da entidade descritor, cujo objetivo é encapsular as informações necessárias para apresentar um nó: procedimentos para iniciação da apresentação, operações para exibição e edição, e modificações no comportamento da apresentação a serem aplicadas em tempo de exibição. A separação das informações relativas à apresentação do conteúdo dos nós, como já mencionado, permite contextualizar a apresentação dos documentos.
- Definição de um esquema de associação dinâmica de descritores a nós dependente do caminho percorrido pelo usuário ao navegar no documento para atingir um nó, facilidade que torna possível a definição de formas alternativas para a apresentação de um componente do documento dependendo da forma como o usuário navega na estrutura do documento. Esta facilidade do modelo proposto define uma alternativa de solução para um dos problemas em aberto na área, conforme cita Zellweger [Zell95].

- Introdução no modelo do conceito de evento, cuja ocorrência é definida pela seleção ou apresentação de uma âncora de um nó (eventos de seleção e apresentação), ou pela modificação do valor de um dos atributos do nó (evento de atribuição). O evento, como será visto no Capítulo 3, é o mecanismo básico para a especificação dos relacionamentos de sincronização temporal e espacial no NCM.
- Redefinição do conceito de elo, permitindo seu emprego na definição de relacionamentos entre *n* eventos de origem e *m* eventos de destino. A nova definição não apenas preservou a capacidade dos elos de exprimir relacionamentos de navegação baseada em seleção de extremidades de origem, como também permitiu seu emprego na especificação de alternativas de navegação automática através da estrutura dos documentos.
- Definição de uma ferramenta integrada para edição e visualização da estrutura e sincronismo de documentos baseados no modelo conceitual proposto. A idéia básica da ferramenta foi tratar os documentos multimídia como estruturas multidimensionais que, como tal, são visualizadas através de três perspectivas integradas: estrutural, temporal e espacial.
- Especificação das entidades do NCM que capturam a especificação do sincronismo em RT-LOTOS [CoOl94]. A especificação permitiu o uso da ferramenta RTL (RT-LOTOS Laboratory) para simular a apresentação de documentos NCM. A ferramenta gera, a partir da especificação RT-LOTOS, um DTA (Dynamic Timed Automaton) que é a estrutura base da simulação. Esta estrutura, entretanto, permite o emprego de técnicas de análise de alcançabilidade [CoOl95] para verificar a consistência da especificação do sincronismo em documentos NCM.

A definição de novas entidades necessárias para especificação da apresentação dos documentos NCM implicou em algumas mudanças significativas ao modelo. Assim, entre outras contribuições desta tese ao modelo, destacam-se também:

- Atualização da definição dos conceitos de perspectiva, elo, elo visível e entidade virtual.

- Redefinição das operações de controle de versões, levando em conta o requisito de várias apresentações distintas (representações) poderem ser derivadas de um único componente.
- Definição dos métodos associados às várias entidades do modelo.

A referência [SoCC93] apresenta a arquitetura do sistema HyperProp. Em sua versão distribuída, a arquitetura propõe uma separação entre clientes, responsáveis pela interação usuário-sistema para manipulação de documentos, e servidores, cuja função principal é o armazenamento de documentos. Com relação ao sistema HyperProp, a principal contribuição do trabalho reportado nesta tese foi o refinamento da arquitetura dos clientes, definindo um ambiente para edição e exibição de documentos [SoSo96]. O ambiente proposto é composto pelos módulos de edição, execução e intercâmbio, como mostra a Figura 1. A implementação dos vários módulos foram assunto de outras dissertações, citadas nesta introdução para dar uma visão completa da definição e desenvolvimento do sistema HyperProp.

O módulo de edição oferece ferramentas que permitem a criação dos documentos através da edição de seus componentes (editores de componentes[2]), da especificação dos relacionamentos entre os mesmos (editor de estrutura [Much96]) e da descrição do comportamento que é esperado quando das exibições desses componentes (editor de sincronismo temporal e espacial [Cost96, CoSS96]). Observando a Figura 1, percebe-se que o objetivo buscado é fornecer ao autor um ambiente de edição integrado, ou seja, a idéia é que o usuário possa ver e editar todos os aspectos do documento que está criando de forma integrada e simultânea [CMSS96, MSCS97, Souz93]. O resultado do processo de edição é uma coleção de objetos NCM que, por sua vez, servirão de entrada para o ambiente de execução, para que a apresentação do documento, que é o resultado final do processo, ocorra conforme especificado pelo autor do documento. O ambiente de edição de documentos NCM é discutido no Capítulo 4 deste texto.

[2] Um dos requisitos básicos do sistema HyperProp é ser um sistema aberto, no caso específico dos editores de componentes optou-se por usar editores já existentes ao invés de desenvolver editores de componentes próprios. Cabe ressaltar que isto implicou no estudo, definição e uso de interfaces abertas implementadas para tais editores, como será detalhado no Capítulo 5.

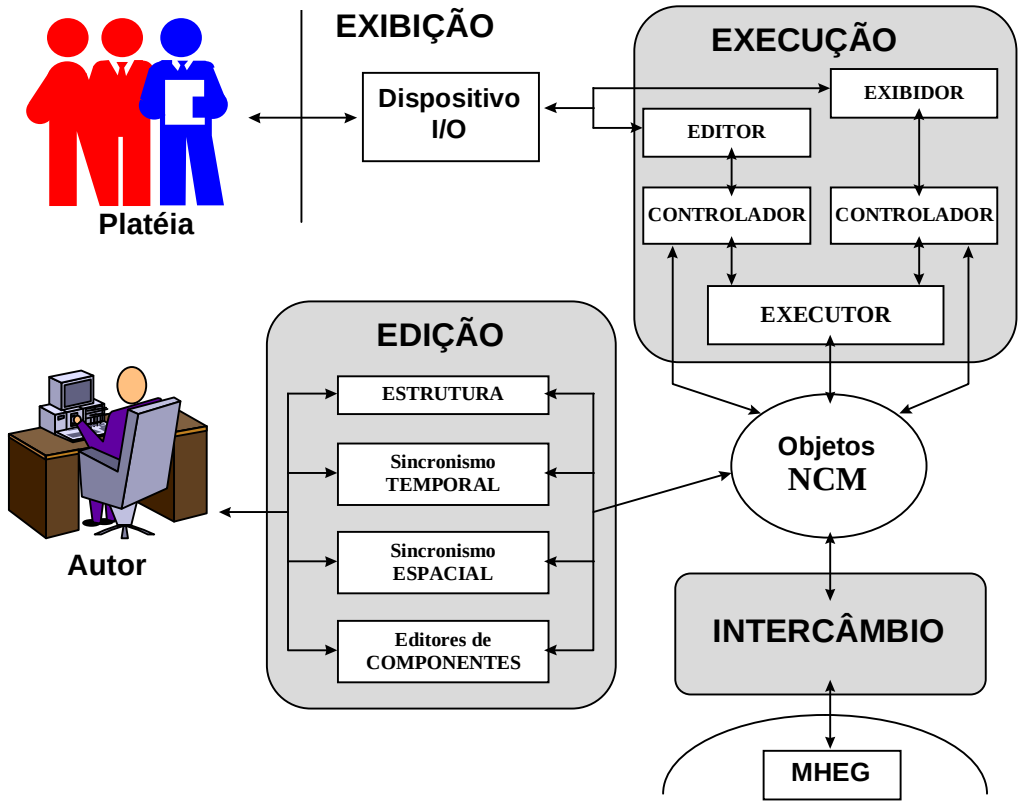


Figura 1: Arquitetura do cliente HyperProp.

A Figura 1 mostra também, de forma simplificada, a arquitetura proposta para o ambiente responsável pela execução das apresentações de documentos NCM [RoSS97a]. A idéia básica da proposta é a separação do controle do sincronismo entre os componentes (um módulo denominado executor controla o disparo dos elos) do controle e monitoramento da exibição dos nós (módulos controladores de exibição e exibidores gerenciam a exibição e monitoram a ocorrência de eventos ao longo das exibições dos nós). A principal vantagem de se adotar esta arquitetura é que ela simplifica a integração de editores de mídia quaisquer ao sistema HyperProp. A arquitetura do ambiente de execução é apresentada no Capítulo 5. O estudo detalhado e a implementação desse ambiente são apresentados em [Rodr97].

Nos clientes HyperProp, o módulo de intercâmbio é responsável pela interface com o subsistema de armazenamento do sistema. Este módulo oferece interfaces que permitem solicitar ao subsistema de armazenamento que grave ou recupere objetos multimídia, deixando transparente a localização dos mesmos, em um sistema de informação distribuído. Cabe ao módulo de intercâmbio, em conjunto com o subsistema de armazenamento, oferecer uma qualidade de serviço (QoS) adequada para recuperação das diversas mídias,

em especial as mídias contínuas, como áudio e vídeo. A referência [CSCS96] apresenta uma interface elaborada com esse objetivo. No módulo de intercâmbio, a representação NCM dos documentos é traduzida para uma representação MHEG [MHEG95], formato de codificação adotado para o intercâmbio dos documentos. Note que todas as facilidades do ambiente se aplicam a qualquer modelo em conformidade com a proposta de padrão para intercâmbio de documentos MHEG.

1.3. Organização da Tese

Esta tese está organizada em mais seis capítulos, além desta introdução.

O Capítulo 2 apresenta características de alguns dos principais paradigmas existentes para especificação do sincronismo na apresentação de documentos multimídia. Através da análise do que já foi feito nesta área de pesquisa, foram identificados os recursos mais eficazes oferecidos por esses paradigmas e sistemas, para incorporá-los na proposta apresentada nesta tese.

Após estudar e analisar os recursos para especificação do sincronismo oferecidos pelos principais paradigmas e sistemas hipermídia existentes, apresenta-se a versão corrente do Modelo de Contextos Aninhados, salientando as contribuições dessa tese em relação às suas versões anteriores. Assim, no Capítulo 3 as extensões propostas são discutidas com base no sistema HyperProp, se constituindo em um dos focos principais deste trabalho.

O Capítulo 4 apresenta uma ferramenta para edição integrada da estrutura e do sincronismo dos documentos, desenvolvida com o intuito de permitir a edição de documentos baseados no modelo NCM.

O Capítulo 5 é dedicado às considerações sobre a execução das apresentações de documentos NCM no sistema HyperProp. Este capítulo inclui uma descrição das entidades NCM que capturam a especificação da apresentação dos documentos feita com o emprego da Técnica de Descrição Formal RT-LOTOS [CoOl94]. Esta descrição permitiu a realização de simulações de apresentações de documentos NCM, também discutidas no Capítulo 5.

O Capítulo 6 descreve alguns dos principais trabalhos relacionados e apresenta uma comparação do modelo proposto com os trabalhos descritos.

Finalizando, é feita uma avaliação crítica dos resultados obtidos, ressaltando as contribuições deste trabalho e indicando quais serão os principais tópicos a serem abordados como trabalhos futuros. Estas conclusões serão apresentadas no Capítulo 7.

Capítulo 2

Especificação de Sincronismo na Apresentação de Documentos

Como mencionado no capítulo anterior, a principal contribuição do presente trabalho foi a definição de mecanismos que permitem a descrição da estrutura de apresentação dos documentos baseados no Modelo de Contextos Aninhados [SoCr95].

Neste capítulo, são definidas de forma mais precisa as noções de sincronização temporal e espacial e são apresentados os principais modelos para especificação de sincronismo encontrados na literatura. Para tanto, inicialmente são definidos alguns conceitos básicos, não só para o presente capítulo, mas para toda a tese. Em seguida, são listados alguns dos principais requisitos relativos à especificação do sincronismo que um modelo de documentos deve satisfazer e é definido um esquema para avaliar suas características. O capítulo se encerra com a apresentação das principais abordagens adotadas por diferentes modelos de documentos para especificação do sincronismo. No Capítulo 6, após a apresentação detalhada do modelo proposto (Capítulo 3) e dos ambientes para edição (Capítulo 4) e execução (Capítulo 5) de documentos NCM, são descritos outros modelos de documentos que, em seguida, são comparados ao proposto nesta tese com base no esquema de avaliação apresentado na Seção 2.2.2.

2.1. Conceitos Básicos

Um *documento multimídia* é uma composição de fragmentos de informação representada em diferentes mídias. Considerando sua característica temporal, as mídias que compõem um documento multimídia são classificadas em dois tipos [StNa95]: mídias dinâmicas (ou contínuas) e mídias estáticas (ou discretas). Mídias *dinâmicas* possuem uma dimensão

temporal inerente, isto é, devem ser apresentadas segundo uma taxa particular por um intervalo de tempo específico, caso contrário a integridade da informação é perdida. Exemplos de mídias contínuas são áudio, vídeo e animações em geral. Mídias *estáticas*, como os textos, os gráficos e as imagens paradas, têm a exibição não dependente do tempo, muito embora, ao compor um documento, um autor possa associar um comportamento temporal (duração de exibição do texto, ou imagem etc.) a uma mídia estática.

Um *documento hipermídia* é um documento multimídia onde o relacionamento entre os componentes, isto é, sua estrutura lógica e de apresentação, é definida com base no paradigma hipertexto, com a ressalva de que, no caso dos documentos hipermídia, os nós contêm informações representadas em diferentes mídias. A incorporação de mídias dinâmicas introduz o aspecto temporal na especificação dos documentos hipermídia e exige a adequação do modelo original de documentos hipertexto no sentido de permitir a combinação, em um único documento, de diversas mídias estáticas e dinâmicas. Coloca-se então como requisito para modelos de documentos dessa natureza, a disponibilização de mecanismos (ou métodos) que permitam descrever, além do mecanismo clássico de navegação por elos baseado na seleção de âncoras, mecanismos capazes de expressar relacionamentos temporais e espaciais entre os componentes do documento, que definem o sincronismo em sua apresentação.

A sincronização nas apresentações de documentos multimídia/hipermídia [STCN92] trata do posicionamento no espaço dos componentes que serão exibidos, ao que chamamos sincronização espacial, e do posicionamento no tempo, ou sincronização temporal.

A noção de sincronização espacial depende da mídia a ser apresentada e baseia-se em operadores que definem como combinar objetos a serem apresentados em um dispositivo de saída, em um dado instante do tempo. Por exemplo, a sincronização espacial de objetos contendo imagens, gráficos ou textos define como posicionar os objetos para apresentação em uma janela da aplicação. A sincronização, nesse caso, envolve operações como mudanças de escala, cortes, conversões de cores e posicionamento dentro da janela da aplicação. Para objetos áudio, a sincronização espacial envolve, por exemplo, a mixagem de canais de áudio em um dispositivo de saída de áudio, com ajustes de volume etc. No Capítulo 3, o sincronismo espacial será novamente discutido quando forem introduzidas as

entidades do modelo NCM que capturam sua especificação. O assunto será também abordado no Capítulo 4, que apresenta uma interface gráfica que dá suporte à especificação do sincronismo espacial em documentos multimídia. No presente capítulo, a discussão será centrada na especificação do sincronismo temporal.

A noção de sincronização temporal baseia-se em mecanismos que definem como a apresentação dos componentes de um documento é escalonada nos dispositivos de saída. O escalonamento pode ser natural, como no caso da exibição simultânea de áudio e vídeo capturados como um sinal único, ou sintética, como em documentos que possuem anotações de áudio.

Distinguem-se, ainda, as relações temporais internas a um componente (sincronização intra-objeto) das relações temporais entre componentes (sincronização inter-objetos) [BlSt96].

A *sincronização intra-objeto* refere-se aos relacionamentos temporais entre as várias unidades de apresentação de um objeto que contém uma mídia contínua. A especificação do sincronismo intra-objeto faz parte da descrição da estrutura dos componentes do documento. A apresentação de um objeto contínuo consiste na apresentação de uma seqüência ordenada no tempo de *unidades de informação* (UI). Um exemplo desse tipo de sincronização é o relacionamento temporal entre os quadros, UIs, de uma seqüência de vídeo. Para um vídeo exibido a uma taxa de 25 quadros por segundo, cada um dos quadros é exibido em 40 ms. São comuns casos onde coexistem unidades de informação com diferentes níveis de granularidade. Por exemplo, em um objeto contendo um filme, as unidades de informação podem ser quadros individuais do vídeo, os conjuntos de quadros de cada uma das cenas do filme, ou o conjunto formado por todos os quadros do filme.

Um ambiente de autoria deve oferecer ferramentas que permitam a criação dos documentos, através da edição de seus objetos componentes, isto é, de seus conteúdos, e da especificação de sua granularidade, que define quais conjuntos de suas unidades de informação podem ser usados na definição de eventos (definidos a seguir). Diz-se que a granularidade é grossa quando o menor conjunto é todo o segmento de mídia. A criação (definição) do conteúdo dos componentes e a definição de sua granularidade fogem do escopo deste trabalho. Para efeito de discussão no texto, são chamados *objetos de dados* os

objetos que contêm essas definições. Normalmente, a criação de tais objetos é de responsabilidade dos editores específicos da mídia. Um ambiente de autoria deve, no entanto, permitir a descrição do comportamento esperado de cada componente, quando for exibido, e a especificação dos relacionamentos temporais entre os componentes, denominada *sincronização inter-objetos*. A especificação do sincronismo inter-objetos é parte da descrição da estrutura da apresentação do documento. Em uma especificação de sincronização inter-objetos, são definidos relacionamentos temporais entre eventos que ocorrem ao longo da apresentação dos componentes de um documento.

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*. O início ou o fim da exibição de um conjunto de unidades de informação define também um evento denominado *ponto de sincronização*.

O instante de ocorrência de um ponto de sincronização pode ser *previsível*, como o fim da exibição de um trecho 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 em relação à ocorrência de outro evento. Obviamente, apenas os pontos de sincronização previsíveis podem, antes do início da apresentação do documento, ter um instante de ocorrência associado.

Cumprir uma especificação de sincronismo é garantir que os eventos relevantes para a especificação ocorrerão de forma ordenada no tempo.

2.2. Esquema para Classificação de Modelos de Sincronização

Buchanan e Zellweger propuseram em [BuZe93] um esquema para classificar sistemas multimídia considerando três pontos: os recursos fornecidos pelo sistema para o suporte aos segmentos de mídia, os métodos usados para posicionar os segmentos ao longo da apresentação do documento e o algoritmo de formatação usado no sistema. Neste capítulo, não serão discutidos os algoritmos utilizados para formatar a apresentação de documentos hipermídia no espaço e no tempo, tema do Capítulo 5, mas sim os dois primeiros pontos do

esquema de classificação, que são capturados pelos modelos dos documentos. Cabe salientar que foi acrescentada ao esquema original, proposto por Buchanan e Zellweger, a avaliação dos aspectos estruturais dos documentos (Seção 2.2.2.3). Nas seções que seguem são definidos os parâmetros dos modelos de documentos considerados relevantes para avaliar o suporte que eles oferecem à especificação do sincronismo.

2.2.1. Especificação da Apresentação de Componentes

O primeiro ponto do esquema de classificação utilizado analisa o modo como os modelos especificam a *apresentação de segmentos monomídia*. São avaliadas quatro características dos modelos: granularidade, duração, flexibilidade e métricas de qualidade de serviço — QoS (Quality of Service).

2.2.1.1. Granularidade

A *granularidade* na especificação da apresentação de um segmento monomídia determina a quantidade de estrutura interna dos segmentos que o modelo torna acessível aos autores para que eles possam especificar a sincronização. Essa característica classifica os modelos como possuindo *granularidade grossa* quando só permitem o acesso às extremidades, isto é, ao início e ao final da apresentação dos segmentos; ou como apresentando *granularidade fina* quando permitem a manipulação de pontos internos aos segmentos.

2.2.1.2. Duração

A característica *duração* indica se os modelos avaliados permitem que sejam definidos os intervalos de tempo necessários para preparar e apresentar os segmentos. Nesse aspecto, os modelos podem permitir a especificação de durações *previsíveis*, cujo valor pode ser determinado quando os segmentos são editados, ou *imprevisíveis*, onde o valor do intervalo de tempo associado à apresentação do segmento só pode ser determinada em tempo de execução da apresentação.

2.2.1.3. Flexibilidade

O aspecto *flexibilidade* lida com a especificação de parâmetros para manipular a duração e a forma da apresentação dos segmentos. Os modelos classificados como *de ajuste contínuo*

permitem que seja especificada uma faixa de valores dentro da qual a duração da apresentação de um segmento pode ser selecionada. Por exemplo, a duração da apresentação de um segmento de áudio pode variar na faixa de 10 a 12 segundos, considerando a diminuição ou não da apresentação dos intervalos de silêncio dos trechos do áudio. Outra forma de especificar um certo nível de flexibilidade na apresentação de segmentos é permitir que o autor defina representações alternativas distintas de um mesmo segmento, as quais poderão ter durações diferentes. Um exemplo deste tipo de flexibilidade seria permitir que um texto fosse exibido graficamente em um monitor ou como áudio com a ajuda de um sintetizador de som. Os modelos que fornecem este último tipo de flexibilidade são classificados como *de ajuste discreto*.

2.2.1.4. Métricas de QoS

As *métricas de QoS* fornecem uma medida objetiva da qualidade da apresentação de um segmento particular. Alguns exemplos de parâmetros de QoS são: a variação de retardo intra-mídia (jitter), a banda passante necessária, a resolução e o número de cores em monitores de vídeo, o número e tipo de canais de áudio etc.

Essa informação, combinada com os atributos que especificam a flexibilidade da apresentação, pode ser usada para selecionar a melhor representação possível para cada segmento na apresentação de um documento em um determinado ambiente.

2.2.2. Relacionamento Temporal entre Componentes

O segundo ponto do esquema de classificação de modelos trata dos *relacionamentos temporais* que descrevem como os segmentos podem ser combinados no tempo para produzir a apresentação de um documento hipermídia. Os relacionamentos temporais possuem quatro atributos classificados como relevantes: granularidade, tipo da relação temporal, flexibilidade e métricas da QoS.

2.2.2.1. Granularidade

A *granularidade* define onde os relacionamentos temporais podem ser posicionados: em um ponto, em um ponto composto, em um intervalo ou em um intervalo composto. Um *ponto* pode ser: um valor absoluto de tempo, um valor de tempo relativo a algum ponto da

apresentação do documento (cinco segundos após o início da apresentação de um determinado segmento), um evento previsível ou imprevisível que ocorre ao longo da apresentação de um segmento, ou um evento externo que pode ser reportado ao sistema (uma ação realizada por um usuário, por exemplo, um click em um botão do mouse). Um *ponto composto* é definido como sendo qualquer relacionamento temporal que identifica um ponto específico do tempo, por exemplo, quando terminarem as exibições dos segmentos α e β. Um *intervalo* pode ser definido pela apresentação completa de um segmento ou pela apresentação de uma parte de um segmento. Um *intervalo composto* é definido como sendo um intervalo de tempo produzido por um relacionamento temporal entre intervalos, por exemplo, enquanto os segmentos α e β estiverem sendo apresentados.

2.2.2.2. Tipo das Relações Temporais

O *tipo das relações temporais* é definido de acordo com a forma como elas controlam a ordem da apresentação dos segmentos de um documento. *Relações de ordenação* são relações binárias que especificam a ordem de ocorrência de pontos ou intervalos durante a apresentação de um documento. Allen [Alle83] enumerou treze relações de ordem básicas entre dois intervalos: antes, depois, sobrepondo, durante etc. *Relações entre durações* especificam relacionamentos entre a duração de intervalos de tempo, por exemplo, a duração do segmento α deve ser igual a duração do segmento β. O esquema de classificação denomina de *relações complexas* as que requerem facilidades de programação para sua realização. As relações complexas são ainda divididas em: relações de agrupamento, relações de iteração e relações condicionais. *Relações de agrupamento* permitem ao autor definir relacionamentos temporais entre pontos ou intervalos de modo que seja possível a utilização do relacionamento entre o grupo de pontos ou intervalos como uma entidade única em outro relacionamento temporal. Por exemplo, começar a apresentação do segmento α quando terminar a apresentação dos segmentos β, δ e ϕ. *Relações de iteração* são usadas para especificar que a apresentação de um determinado intervalo seja repetida por um número de vezes qualquer, ou enquanto (ou até que) uma certa condição seja satisfeita. *Relações condicionais* são as que dependem do documento ou do sistema estar em um estado particular para serem satisfeitas, como, por exemplo,

apresentar o segmento de áudio ϕ se o dispositivo de saída de áudio da estação de trabalho estiver disponível.

2.2.2.3. Flexibilidade

Segundo o esquema de classificação adotado neste trabalho, os modelos de apresentação podem suportar duas formas de *flexibilidade* na especificação dos relacionamentos temporais. Na primeira, o modelo permite aos autores especificar que um relacionamento temporal é *obrigatório*, isto é, tem que ser satisfeito durante a apresentação do documento, ou *opcional*, caso em que o relacionamento temporal é desejável porém pode ser ignorado. Este tipo de flexibilidade é denominada flexibilidade *discreta*. O segundo tipo de flexibilidade nos relacionamentos temporais baseia-se na especificação de uma *faixa* de valores de tempo dentro da qual um relacionamento temporal é considerado satisfeito. Por exemplo, o autor de um documento pode usar este recurso para definir que a apresentação dos segmentos ϕ e γ deve acabar no mesmo instante com uma tolerância de mais ou menos dois segundos.

2.2.2.4. Métricas de QoS

Os modelos que permitem a especificação de *métricas de QoS* fornecem uma maneira de medir a degradação da qualidade de uma apresentação devido à desconsideração de relacionamentos temporais opcionais ou à variação do momento da ocorrência de um relacionamento temporal dentro de uma faixa de valores previamente especificada.

2.2.3. Composições na Estruturação de Documentos

A definição estruturada de documentos é desejável por trazer embutidos os conceitos de modularidade, encapsulamento e mecanismos de abstração. A estruturação lógica do documento é realizada pela introdução do conceito de composição como um agrupamento de componentes do documento e de suas relações [RoSS97b]. É desejável que os modelos que utilizam o conceito de composição para estruturar seus documentos apresentem as seguintes propriedades:

- Aninhamento de composições, isto é, permitam que composições contenham outras composições.

- Agrupamento dos componentes de um documento e dos relacionamentos entre eles independente de seus tipos (relacionamentos de sincronismo para apresentação, relacionamentos de seleção para a navegação usual através de *hyperlinks*, etc.)
- Uso da composição como um novo tipo de componente, em todos os sentidos, permitindo que:
    - ◊ A composição possa ser exibida[3] — é importante a apresentação da estrutura dos componentes de uma composição e não apenas o conteúdo dos seus componentes. Por exemplo, ao acessar uma composição representando o capítulo de um livro, pode-se querer não apenas a visualização do conteúdo de dados do capítulo, mas também de sua estruturação em seções.
    - ◊ Possam ser definidos diferentes pontos de entrada (âncoras) em uma composição. O que se deseja é que uma composição possa ter apresentações diferentes dependendo do ponto de entrada. Como consequência, a duração da apresentação de uma composição (duração da exibição de seus componentes) irá depender não só da duração dos componentes, mas também do ponto de entrada.
    - ◊ Possam ser definidos relacionamentos entre composições.
- *Herança* na definição das composições, no sentido de que relações possam ser definidas em uma composição *C* com base na herança (reuso) de componentes e estruturas de outras composições. Como exemplo tome uma composição representando um capítulo de um livro. Para um livro (outra composição L_1) entregue a um leitor específico, poderia ser desejável a introdução de uma relação entre duas seções do capítulo, por exemplo para indicar um complemento de informações, que não seria necessário a um leitor de maior conhecimento, a quem pode ser entregue outro livro (outra composição L_2), sem a relação. Note que L_1

[3] A apresentação de um nó de composição é diferente da apresentação de seus componentes. A apresentação de um nó de composição é a exibição da estrutura definida na composição e não a exibição de cada um dos seus componentes.

poderia ser definida simplesmente como uma composição contendo L_2 e a relação introduzida, reusando toda a estruturação de L_2.

- Auxílio ao leitor para navegação sobre o documento. As composições podem ser usadas para agrupar nós com base em algum relacionamento semântico facilitando a leitura do documento, pois o leitor, nesse caso, não vê todos os nós do documento de uma só vez. Esta propriedade vai facilitar a implementação de algoritmos de filtragem, tal como o olho de peixe estendido [Much96], quando da exibição da estrutura, de forma a diminuir o problema de desorientação do leitor.

Assim, com relação ao aspecto estruturação dos documentos, são avaliadas quatro características dos modelos: granularidade, tipo dos relacionamentos agrupados, aninhamento e herança.

2.2.3.1. Granularidade

Modelos que permitem que se tenha vários pontos de entrada em uma composição são classificados como de *granularidade fina*. Nestes modelos, é possível ao autor especificar exibições diferentes para os componentes de uma composição dependendo do ponto de entrada na composição. Note que as UIs (unidades de informação) das composições são seus componentes: nós e elos. A duração de um nó de composição vai, assim, depender do ponto de entrada e não apenas da duração de seus componentes. Uma outra característica dos modelos com granularidade fina nas composições é que eles permitem que sejam definidos relacionamentos entre componentes internos de composições distintas. Modelos que apresentam *granularidade grossa* são os que só permitem referenciar a composição como um todo, não permitindo acesso via navegação aos componentes internos da composição. Esse tipo de modelo só permite o acesso às extremidades, isto é, ao início e ao final da apresentação das composições.

2.2.3.2. Tipo dos Componentes e Relacionamentos Agrupados

Com relação ao tipo dos relacionamentos agrupados, as composições podem: agrupar componentes e relacionamentos *previsíveis* (cujo tempo de ocorrência pode ser calculado a priori) ou agrupar componentes e relacionamentos *imprevisíveis* (por exemplo, interação com usuário).

2.2.3.3. Aninhamento

Esta característica permite identificar os modelos que permitem o *aninhamento* das composições, isto é, uma composição conter outra composição.

2.2.3.4. Herança

Esta característica está associada à capacidade de um modelo de documentos permitir a *herança* na definição das composições, no sentido de que relações definidas em uma composição possam ser reusadas na definição de outras composições.

2.2.4. Especificação do Ambiente onde se dará a Apresentação do Documento

Neste item do esquema de classificação de modelos são consideradas as facilidades disponíveis nos modelos para que os autores possam *especificar o ambiente* onde se dará a apresentação do documento. Esses parâmetros são: especificação de comportamento, especificação do posicionamento no espaço definido pelos dispositivos de saída, especificação da estação de trabalho e especificação do suporte de comunicação utilizado.

2.2.4.1. Especificação de Comportamento

Os modelos que possuem facilidades para *especificação de comportamento* permitem que os autores possam definir que a apresentação de um segmento particular assuma um determinado comportamento a partir de um ponto qualquer ao longo de sua apresentação. Um exemplo desse tipo de especificação é a possibilidade de programar a modificação do volume de um segmento de áudio quando a sua apresentação atingir 50% de sua duração.

2.2.4.2. Especificação Espacial

A *especificação espacial* permite aos autores controlar onde e como os segmentos deverão aparecer fisicamente nos dispositivos de saída. Um exemplo de um parâmetro que faz parte deste grupo é a definição da posição e do tamanho da área do display onde um segmento contendo uma imagem deve ser apresentado.

2.2.4.3. Especificações da Estação de Trabalho

As *especificações da estação de trabalho* descrevem os recursos da estação de trabalho na qual um documento deverá ser apresentado (os dispositivos de hardware e as ferramentas de software).

2.2.4.4. Especificações do Suporte de Comunicação

Por fim, as *especificações do suporte de comunicação* definem as características do sistema de comunicação que irá suportar a transmissão dos segmentos do documento.

2.3. Modelos para Especificação do Sincronismo Temporal

Nesta seção são descritos os modelos básicos [BlSt96] usados para especificação do sincronismo temporal de documentos multimídia/hipermídia: baseados em linhas de tempo, hierárquicos, scripts, baseados em redes de Petri, baseados em pontos de referência e baseados em eventos. A descrição introdutória das abordagens básicas para especificação do sincronismo, feita nesta seção, visa facilitar a compreensão do método adotado no modelo NCM descrito em detalhes no Capítulo 3.

Para facilitar o entendimento e a comparação, será feita uma descrição da apresentação de um documento exemplo segundo cada um dos modelos descritos. No documento um locutor descreve em inglês uma seqüência de três imagens, que devem ser apresentadas junto com os trechos do áudio que descrevem cada uma delas, respectivamente. Adicionalmente, são exibidas legendas em português (texto) com traduções da descrição em inglês do áudio. As legendas são distribuídas em três componentes com a descrição relativa a cada uma das imagens, e devem ser mostradas junto com as respectivas imagens. Se o usuário selecionar a palavra “música” que aparece no texto da primeira legenda, um componente de áudio contendo uma música deve ser adicionalmente exibido.

2.3.1. Sincronização Hierárquica

Na sincronização hierárquica [Pogg85, ISO89, SaSh90], a descrição do sincronismo de uma apresentação é feita com base em duas operações de sincronização: sincronização serial de ações e sincronização paralela de ações.

Uma ação pode ser atômica ou composta. Uma ação atômica define a exibição de um componente de um documento (texto, áudio etc.), de uma interação com o usuário, ou de um retardo.[4] Uma ação composta é uma combinação de operações de sincronização e ações, na forma de uma árvore onde os nós terminais são ações atômicas e os nós não terminais são operações de sincronização.

A Figura 2 mostra duas árvores descrevendo o sincronismo da apresentação exemplo. Na árvore A o sincronismo foi definido sem o uso de ações retardo, e na B foi usado o método hierárquico com o uso de retardo. Os símbolos >> e & correspondem aos operadores seqüencial e paralelo, respectivamente. O componente *áudio* na Figura 2-B corresponde aos componentes *Áudio1*, *Áudio2*, e *Áudio3* na Figura 2-A. As durações dos componentes *Retardo1* e *Retardo2* na Figura 2-B são equivalentes àquelas dos componentes *Áudio1* e *Áudio2* na Figura 2-A.

As estruturas hierárquicas são fáceis de manipular e muito utilizadas. A principal restrição deste modelo é que as ações (que comandam a exibição dos componentes) só podem ser sincronizadas explicitamente no início ou no final de sua exibição.

No exemplo mostrado na Figura 2-A, todos os pontos de sincronização foram definidos explicitamente, ou seja, através da aplicação direta de operações de sincronização.

Por outro lado, na Figura 2-B alguns pontos de sincronização foram definidos implicitamente (indiretamente). Um exemplo de ponto de sincronização implícito é o que relaciona o início da apresentação do trecho do componente *áudio* onde é descrita a segunda imagem (evento 1) com o início da apresentação do componente *imagem2* (evento 2). O ponto de sincronização que relaciona estes eventos é implícito porque na árvore que define o sincronismo da apresentação, Figura 2-B, nenhuma operação de sincronização é aplicada diretamente aos eventos que o definem. Entretanto, a introdução da ação *retardo1* (com duração igual ao tempo gasto no áudio com a descrição da primeira imagem), que começa a ser exibida junto com a ação *áudio*, faz com que os eventos 1 e 2 ocorram

[4] A introdução de um retardo como uma possível ação [LiGh90], torna possível a modelagem de comportamentos de sincronização adicionais como retardos em apresentações seriais e apresentação retardada de objetos em uma sincronização paralela.

simultaneamente. Portanto, a colocação da ação *retardo1* caracterizou a definição de um ponto de sincronização implícito relacionando indiretamente os eventos 1 e 2.

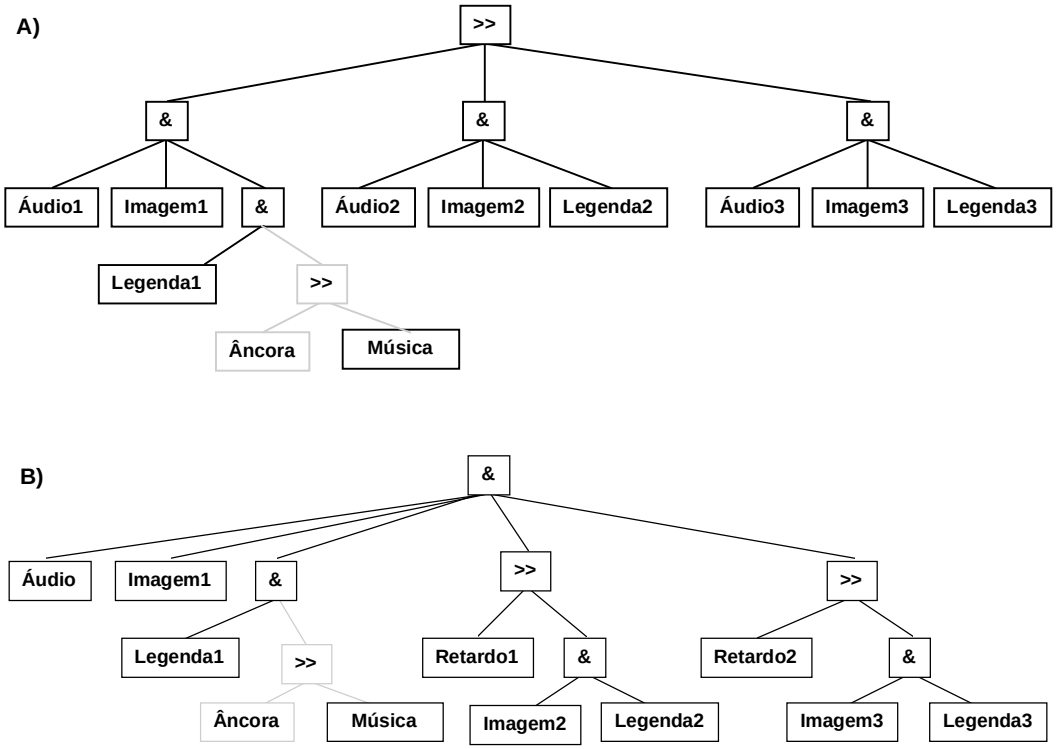


Figura 2 - Descrição do sincronismo da apresentação exemplo usando o método de sincronização hierárquico.

A impossibilidade de se definir explicitamente pontos de sincronização relacionando eventos que ocorrem ao longo da exibição de um componente é a principal limitação do método hierárquico, pois obriga o autor a quebrar os componentes em subcomponentes (Figura 2-A) ou a usar ações retardo para definir pontos de sincronização implícitos (Figura 2-B).

A quebra de componentes em subcomponentes provoca uma fragmentação que pode dificultar, e até mesmo impossibilitar, a manutenção da continuidade na apresentação de mídias contínuas provocando problemas, como por exemplo, a inclusão na reprodução de um trecho de áudio de intervalos de silêncio que não fazem parte do componente original.

O problema com os retardos é que são semanticamente associados à duração de trechos de componentes, porém as máquinas que executam as apresentações não são determinísticas,

ou seja, variações em seu estado[5] podem fazer com que a duração da exibição dos componentes também varie. Este problema se agrava quando a apresentação é executada em plataformas (software e hardware) diferentes, onde a variação pode ser suficiente para provocar perdas de sincronismo graves. Por exemplo, na execução da apresentação descrita na Figura 2-B em uma máquina lenta no processamento de imagens, poderíamos ter o componente *imagem2* ainda sendo exibido ao mesmo tempo que no áudio já estivesse sendo descrita a terceira imagem.

Outro problema provocado pelo uso de ações de retardo para definição de pontos de sincronização implícitos ocorre quando o trecho do componente semanticamente ligado ao componente retardo é modificado. Nesse caso, se o autor não ajustar manualmente o valor do retardo para a nova situação, o ponto de sincronização implícito perde o sentido.

2.3.2. Sincronização Baseada em Intervalos

Na especificação de sincronização baseada em intervalos, a duração da apresentação de um componente é manipulada como um intervalo de tempo. Os intervalos podem ser sincronizados de acordo com 13 relações básicas [Hamb72, Alle83, KiSo95], ou usando uma das 29 relações que [WaRo94] considerou relevantes para especificação do sincronismo em documentos multimídia. Algumas das relações básicas são invertíveis. A Figura 3 mostra as treze relações básicas.

No modelo baseado em intervalos, objetos com duração de apresentação imprevisível, por exemplo interações com usuários, podem ser modelados como intervalos com duração indefinida, por exemplo, na referência [WaRo94] a duração dos intervalos pode ser maior que zero.

Relações Básicas	Significado
x before y y after x	├——x——┤ ├—τ_1—┤ ├——y——┤
x meets y	├——x——┤

[5] Por exemplo, um aumento do número de processos concorrentes disputando recursos em uma máquina com sistema operacional time-sharing.

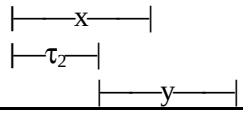
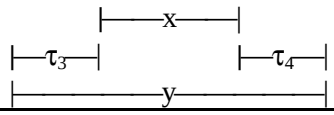
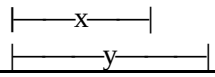
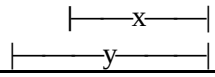
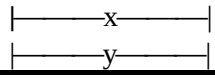
y met-by x	
x overlaps y y overlapped-by x	
x during y y contains x	
x starts y y started-by x	
x finishes y y finished-by x	
x equal y	

Figura 3 - Relações entre básicas entre intervalos temporais.

Na Figura 4, a apresentação exemplo é especificada com o método baseado em intervalos.

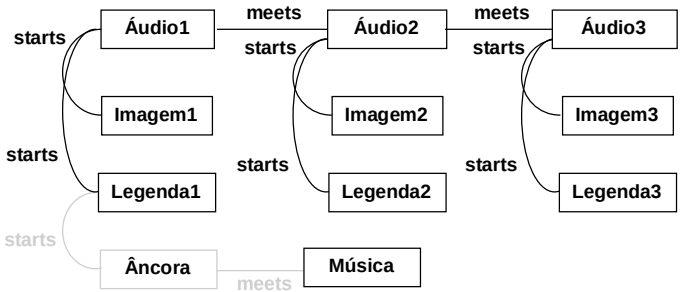


Figura 4 - Apresentação exemplo descrita segundo o método baseado em intervalos.

O principal problema com o método de especificação baseado em intervalos, como no método hierárquico, é não permitir a especificação de relações temporais entre eventos que ocorrem ao longo da exibição dos componentes. Tais relações só podem ser definidas indiretamente com o uso de retardos ou através da divisão dos componentes em subcomponentes, o que leva aos mesmos problemas descritos para o método hierárquico.

2.3.3. Sincronização Baseada em um Eixo de Tempo

Na sincronização baseada em um eixo de tempo (SET), os componentes do documento são posicionados em um eixo de tempo que representa uma abstração do tempo real [HoSA89, Macr89, OgHK90, Pool91]. Neste método de sincronização, os pontos de sincronização são definidos com base em uma referência de tempo (relógio de tempo real do sistema)

independente da duração da exibição dos componentes da apresentação. A ocorrência dos pontos de sincronização é determinada no tempo, pela contagem de intervalos periódicos (milisegundos, por exemplo) gerados por um relógio de tempo real do sistema, e não por eventos gerados (definidos no tempo) pela exibição dos componentes da apresentação.

No modelo SET, o usuário não pode definir, explicitamente, que a exibição de um componente α deve começar quando terminar a exibição de um componente β. Neste modelo, este ponto de sincronização só pode ser definido, implicitamente, da seguinte forma: a exibição do componente β começa no instante 0 mS e termina no instante 600 mS; e a exibição do componente α começa no instante 600 mS e termina no instante 1500 mS. Se, por um motivo qualquer, a máquina que executa a apresentação levar 800 mS para apresentar β, o ponto de sincronização acima descrito não é respeitado. A Figura 5 mostra a especificação da apresentação exemplo segundo o método SET.

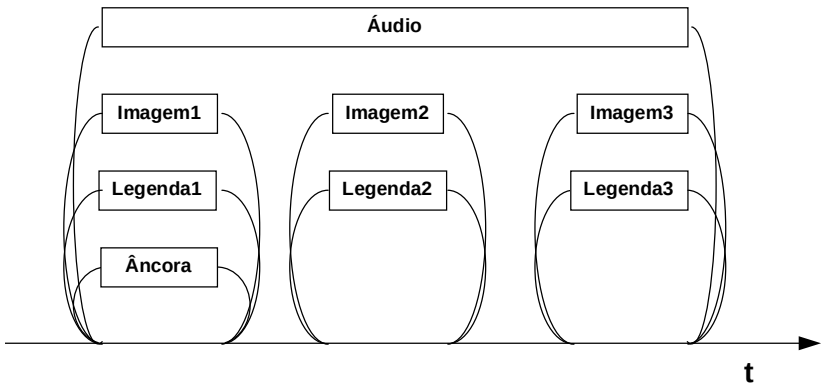


Figura 5 - Apresentação exemplo descrita segundo o método baseado em um eixo de tempo.

Embora seja um método de especificação bastante intuitivo e consequentemente de fácil aprendizado para os usuários, este método de sincronização apresenta três problemas graves [BuZe92]. Primeiro, é difícil representar comportamentos assíncronos como eventos com tempo de ocorrência imprevisíveis, como por exemplo interações com os usuários, e componentes com duração também imprevisível, como execuções de programas e exibições de componentes virtuais.[6] Segundo, a edição dos documentos não é uma tarefa simples

[6] Um componente virtual é um componente de um documento cujo conteúdo é definido por uma expressão que retorna um objeto compatível com o do conteúdo esperado para o componente. A expressão é avaliada sempre que o componente precisar ser exibido. O conceito de componente virtual é melhor definido no Capítulo 3.

porque os autores podem precisar editar manualmente os componentes, cortando partes do conteúdo ou mudando a velocidade de exibição das unidades de informação, para adaptar sua duração de modo a obter os relacionamentos temporais desejados entre componentes. Terceiro, não é uma tarefa simples manter os documentos ao longo de sua existência, pois uma pequena modificação em um dos componentes pode requerer um grande esforço para reposicionar todos os componentes a ele relacionados através de pontos de sincronização implícitos. Como todos os relacionamentos temporais entre componentes do documento são definidos indiretamente neste método de especificação, cabe ao autor "identificar" os relacionamentos temporais e editar manualmente a especificação de modo a mantê-los válidos após modificações no documento.

No modelo SET continuam existindo os problemas de perda do sincronismo definido implicitamente devido a variações na duração da apresentação dos componentes causadas pelo não determinismo da máquina que executa a apresentação, ou pela sua execução em plataformas diferentes.

O padrão HyTime [ISO92] utiliza uma generalização da abordagem baseada em um eixo de tempo: sincronização baseada em *Eixos Virtuais*. Neste método de especificação, é possível especificar sistemas de coordenadas baseados em unidades de medição definidas pelos usuários. Por exemplo, o autor de um documento pode definir um eixo cuja unidade de medição é determinada pelos batimentos do coração de um paciente em um documento médico. A especificação de sincronização é feita com base nos eixos virtuais. É possível, adicionalmente, usar vários eixos definindo um espaço de coordenadas virtuais. Em tempo de execução os eixos virtuais são mapeados para eixos reais. O conceito de espaço de coordenadas virtuais é usado na definição do *generic space* no padrão MHEG [MHEG95].

2.3.4. Sincronização Baseada em Scripts

No método de especificação de sincronização baseado em scripts, o autor usa uma linguagem textual para programar a estrutura da apresentação dos documentos [FiTD87, IBM90, TsGD91]. Freqüentemente, as linguagens de script tornam-se linguagens de programação completas estendidas com o acréscimo de operações temporais [StNa95]. A Figura 6 apresenta a especificação da apresentação exemplo com uma linguagem de script.

```
activity DigAudio Audio(“ explicação.au” );
activity Picture Imagem1(“ imagem1.jpeg” );
activity Picture Imagem2(“ imagem2.jpeg” );
activity Picture Imagem3(“ imagem3.jpeg” );
activity Text Legenda1(“ legenda1.txt” );
activity Text Legenda2(“ legenda2.txt” );
activity Text Legenda3(“ legenda3.txt” );
activity StartInteraction Ancora;
activity DigAudio Musica(“ musica.au” );

script Bloco1 = Imagem1.Duration(10) &
            (Legenda1.Duration(10) & Ancora);

script Bloco2 = Imagem2.Duration(15) &
Legenda2.Duration(15);

script Bloco3 = Imagem3.Duration(8) & Legenda3.Duration(8);

script Apresentação_exemplo
    {
    Audio &
    ( Bloco1 >> Bloco2 >> Bloco3 )
    }
```

Figura 6 - Apresentação exemplo descrita por um script.

No método baseado em scripts, como no caso do método baseado em um eixo de tempo, se após completar uma especificação complexa o autor precisar modificar as características temporais de parte da especificação, ele vai precisar verificar se as relações temporais previamente definidas estão consistentes e, eventualmente, precisará modificar algumas delas. Esse ajuste pode envolver a identificação e reescrita de todas as partes do código do script afetadas pela modificação. O problema torna-se mais complexo se o componente modificado, ou a parte da especificação modificada for reutilizada em diferentes pontos da estrutura da apresentação do documento [HaRB93]. Outro problema com a especificação baseada em scripts [BHLM92] é a dificuldade dos autores de extrair os relacionamentos temporais e espaciais de uma especificação baseada em uma linguagem textual. Por outro

lado, este método de especificação é muito poderoso, uma vez que se baseia em um ambiente de programação completo.

2.3.5. Sincronização Baseada em Redes de Petri

Um outro esquema utilizado para especificação temporal da apresentação de documentos é o baseado em redes de Petri. Redes de Petri fornecem uma representação gráfica que permite modelar esquemas de sincronização de documentos multimídia e analisar propriedades do documento [Will96].

Para efeito de ilustração, será descrita uma variação proposta por Little e Ghafoor [LiGh90, LiGh91a, LiGh91b], que se baseia em modelos OCPN (Object Composition Petri Net). O OCPN amplia o modelo convencional de redes de Petri, associando aos nós da rede valores de tempo (durações) e recursos utilizados. Em uma OCPN, a cada nó α da rede está associado um objeto O_α, uma duração D_α e um recurso R_α. Uma transição T_i dispara quando cada um dos seus nós de entrada contém uma ficha desbloqueada. Após disparar, a transição T_i remove uma ficha de cada um dos seus nós de entrada e adiciona uma ficha a cada um dos seus nós de saída. Após recebida, uma ficha permanece bloqueada pela duração definida por D_α. Enquanto um nó possui uma ficha, ele está ativo e o objeto O_α é exibido através do recurso R_α. Em se tratando de um objeto contínuo β, um nó estará associado a um trecho T_β do objeto, sendo as transições usadas para sincronizar a exibição de T_β com trechos de outros objetos contínuos ou com a exibição de objetos estáticos. O modelo OCPN permite, adicionalmente, associar uma subrede a um lugar tornando possível a definição de hierarquias. A Figura 7 mostra a apresentação exemplo especificada por uma OCPN.

Outros tipos de redes de Petri [StFu90, WSSD96] simulam o comportamento de elos baseados na seleção de âncoras permitindo o disparo assíncrono de transições habilitadas.

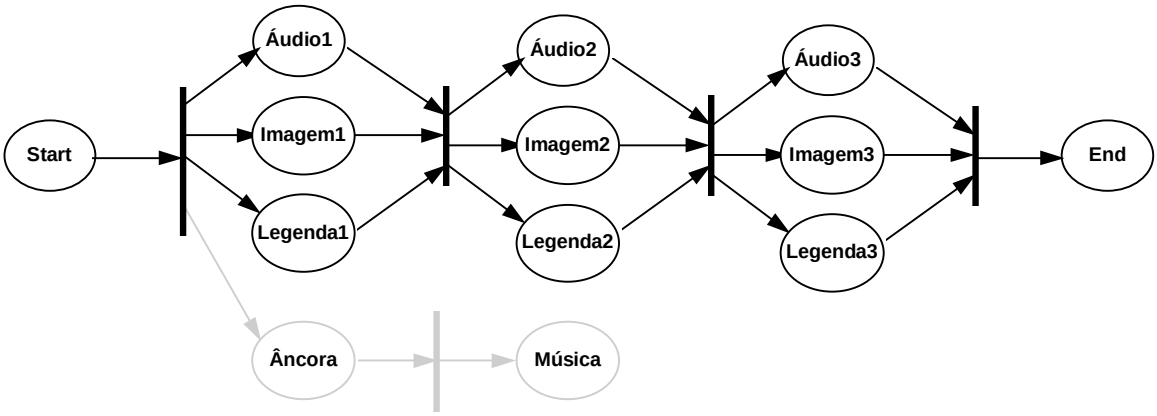


Figura 7 - Apresentação exemplo descrita por uma OCPN.

Como nos casos da sincronização baseada em intervalos e hierarquia, a especificação baseada em redes de Petri só permite a sincronização com os eventos início e fim de exibição de um componente. Para definir pontos de sincronização com eventos que ocorrem ao longo da exibição dos componentes é necessário dividir o componente em subcomponentes cuja exibição é então associada a lugares na rede. A divisão faz com que a especificação da apresentação dos documentos dê origem a redes de Petri complexas e consequentemente de difícil leitura e interpretação para os usuários (autores e leitores manipulando a estrutura do documento). Porém, o embasamento matemático que dá suporte às redes de Petri como ferramenta para modelagem de sistemas dinâmicos torna fácil sua análise. Assim, elas são uma ferramenta interessante quando se deseja simular a execução de apresentações para provar propriedades e antecipar o comportamento futuro da apresentação.

2.3.6. Sincronização Baseada em Pontos de Referência

Nos modelos de documentos onde a sincronização é descrita com base em pontos de referência [Stei90, BHLM92, StNa95], a exibição de um componente é formada pela exibição de suas unidades de informação apresentadas em intervalos de tempo definidos pelo método utilizado para codificar a informação armazenada no componente. Os instantes inicial e final da apresentação dos componentes ou de suas unidades de informação são chamados pontos de referência. A sincronização inter-objetos é definida conectando-se pontos de referência em diferentes componentes. Um conjunto de pontos de referência conectados é chamado de ponto de sincronização. A apresentação das subunidades

associadas a um ponto de sincronização é iniciada ou encerrada quando o ponto de sincronização é atingido.

A sincronização baseada em pontos de referência é, sem dúvida, superior aos modelos anteriormente descritos pois, ao permitir a definição de pontos de sincronização associando pontos de referência no interior dos componentes, elimina a necessidade de quebra dos componentes ou de definição de pontos de sincronização implícitos. Outro ponto positivo é que o tempo da apresentação dos componentes é quem define os instantes de ocorrência dos pontos de referência, e não um relógio externo, assim modificações no tempo de apresentação dos componentes devido a variações no ambiente onde ela está ocorrendo tem um impacto menor na qualidade da apresentação. Como todo o sincronismo é especificado de forma direta conectando-se pontos de referência nos diferentes objetos, este método de especificação comporta-se melhor que os demais em relação à modificação nas características temporais de componentes. Isto é, o ajuste do sincronismo pode ser feito de forma automática. O método baseado em pontos de referência, quando acompanhado do suporte de uma interface gráfica para especificação do sincronismo, é certamente o método mais simples de utilizar, pois permite aos usuários identificarem com clareza os relacionamentos temporais entre os componentes dos documentos.

Um problema com a especificação baseada em pontos de referência é que ela pode levar a especificações inconsistentes, isto é, especificações que não podem ser executadas em determinados ambientes de execução. Nos Capítulos 4 e 5 este problema receberá um tratamento mais aprofundado.

A Figura 8 mostra a apresentação exemplo especificada segundo o método baseado em pontos de referência.

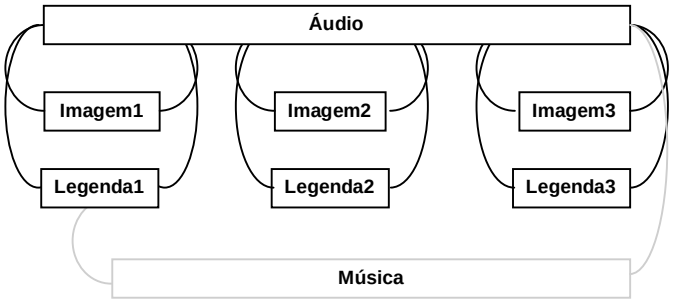


Figura 8 - Apresentação exemplo descrita com base em pontos de referência.

2.3.7. Sincronização Baseada em Eventos

A interpretação da apresentação ou seleção de uma unidade de informação como definindo um evento, e a associação da ocorrência de eventos à execução de ações (play, stop, etc.) em unidades de informação de destino, define uma variação da sincronização baseada em pontos de referência denominada sincronização baseada em eventos. Este é o esquema de sincronização adotado pelo padrão MHEG que define um formato de intercâmbio para documentos multimídia/hipermídia [MHEG95]. As considerações tecidas para o método baseado em pontos de referência aplicam-se igualmente ao esquema baseado em eventos.

2.3.8. Sincronização Baseada em Modelos Híbridos

Os diversos métodos de especificação de sincronização possuem diferentes facilidades que podem ser combinadas com o objetivo de facilitar o processo de autoria ou o controle da execução das apresentações dos documentos. Por exemplo, no editor CMIFed [RJMB93, HaRB93] os autores utilizam o método hierárquico para especificar a apresentação dos documentos na *hierarchy view* combinado a uma mistura dos métodos baseados em eixo de tempo e pontos de referência na *channel view.*

O modelo de documentos NCM, descrito no Capítulo 3, adota o método baseado em eventos como esquema básico para descrição da estrutura de apresentação dos documentos (as âncoras nos nós definem os pontos de referência e os elos definem os relacionamentos entre eventos). Entretanto, esse esquema básico não impediu que a ferramenta desenvolvida para dar suporte à edição de documentos baseados no modelo forneça aos autores um ambiente onde são combinados os métodos baseado em intervalos, em eixo de tempo e em eventos. O editor de documentos NCM é descrito no Capítulo 4 deste texto.

Adicionalmente, em tempo de execução, a especificação do sincronismo baseada em eventos dos documentos NCM pode ser traduzida para uma Rede de Petri Temporizada ou para um DTA (Dynamic Timed Automaton) que seria então a estrutura de dados utilizada para monitorar o andamento da execução da apresentação do documento, como será explicado no Capítulo 5.

No Capítulo 6, após a apresentação detalhada do modelo e dos ambientes para edição e execução de documentos NCM, são descritos outros modelos de documentos que, em seguida, são comparados com o proposto nesta tese.

Capítulo 3

Modelo de Contextos Aninhados

A maioria dos sistemas hipermídia foram desenvolvidos como aplicações auto-contidas, não proporcionando facilidades para intercâmbio de informações, interoperabilidade e reuso de código entre aplicações. São poucas as exceções a este contexto, como os sistemas Neptune [DeSc85], HyperBase [ScSt90], MultiCard [RiSa92], Hyperform [WiLe92] e HyperProp [SoCC93]. O sistema HyperProp provê não apenas um modelo conceitual de dados hipermídia, o Modelo de Contextos Aninhados - NCM (*Nested Context Model*), como uma arquitetura aberta, cujo modelo de interface separa os componentes de dados e de exibição dos objetos, permitindo a manipulação de documentos independente da plataforma final de exibição.

As primeiras descrições do NCM se ativeram mais aos aspectos da estrutura de dados do modelo [Casa91] e das regras de estruturação. As estruturas e operações para controle de versão foram assunto de especificação posterior [SoCR95]. O presente trabalho revê e atualiza as definições anteriores, acrescentando ao modelo básico operações para criação, edição e exibição da estrutura do documento, novas estruturas de dados para a definição de relações de sincronismo espacial e temporal de documentos, bem como operações para definição e exibição dessas relações de sincronismo.

Para a introdução dos novos conceitos, a hierarquia de classes original do NCM foi inteiramente revista. Novas classes foram acrescentadas, novos métodos foram acrescentados às classes anteriormente especificadas [SoCR95], além do que foram necessárias modificações na especificação de algumas das classes originais, especialmente da classe *elo*, através da introdução de novos atributos ou mesmo da redefinição dos antigos atributos. Na verdade, todas as classes do modelo que podem ter objetos

instanciados foram parcialmente ou significativamente modificadas. Um resumo das alterações e acréscimos sofridos pelo NCM básico [SoCR95] é dada na Seção 3.1, antes de se começar a descrever, propriamente, a nova especificação completa do NCM.

A descrição do NCM apresentada neste capítulo é complementada com sua especificação utilizando a notação em sintaxe abstrata ASN.1 (*Abstract Sintax Notation 1)* padrão ISO 8824 listada no Anexo A.

3.1. Reavaliação do NCM

A Figura 9 apresenta a hierarquia de classes da versão anterior do NCM ora implementada. A título de comparação, a Figura 10 (Seção 3.2) apresenta a hierarquia de classes da versão atual. Observando as hierarquias percebe-se o quanto foi acrescentado, principalmente devido ao submodelo de apresentação do documento. As figuras escondem, no entanto, que praticamente todas as classes originais sofreram revisões, algumas profundas, como, por exemplo, todo o tratamento de versões, que teve de ser modificado para se adaptar ao submodelo de apresentação. Essa seção discute resumidamente os acréscimos e as modificações introduzidas no modelo. Leitores não familiarizados com a versão anterior do NCM são aconselhados a pular essa seção, voltando a ela no final da Seção 3.2, quando todos os termos aqui empregados forem completamente definidos.

Várias definições do NCM original foram tornadas mais precisas na versão atual. Entre elas a definição de conteúdo, região, unidades de informação, unidades de informação marcadas e também a identificação de âncoras. Tornar essas definições mais precisas foi imperativo para a definição de eventos.

Grande parte dos novos conceitos da versão atual do NCM derivou da introdução do conceito de evento, parte central de todo o submodelo de apresentação. O conceito de evento como exibição, seleção ou modificação de um objeto já havia sido definido em outros modelos, como o MHEG. O NCM estende esse conceito. Os eventos NCM são definidos não apenas como a exibição, seleção ou modificação (atribuição) de todo o nó, mas de qualquer região do nó. Esta definição vai permitir a especificação de pontos de

sincronização com granularidade mais fina entre objetos, sem a necessidade de subdividi-los. No entanto, acrescenta uma complexidade muito maior ao modelo.

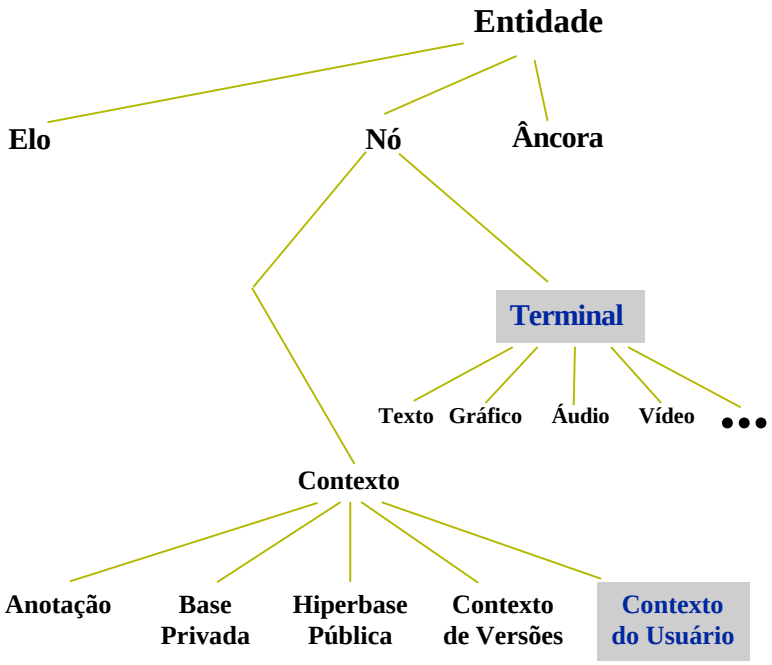


Figura 9 - Hierarquia de Classes do NCM - Versão Anterior

Eventos no MHEG não têm estado. Nós têm estado. Toda definição de condições e ações no MHEG é baseada no estado dos nós. No MHEG, implicitamente, estado de um evento é igual ao estado de um nó. No NCM, eventos têm estado, diferente dos estados dos nós. Em particular, o evento de exibição de toda a região de um nó define, implicitamente, o estado de um nó, como definido no MHEG. Toda definição de condições e ações no NCM é baseada no estado dos eventos. Como os eventos podem definir um sincronismo mais fino que os nós, o modelo de sincronização por elos NCM é bem mais geral que o modelo MHEG.

A definição de eventos e a dinâmica exigida em uma apresentação fez com que o conceito de elo fosse totalmente redefinido para permitir relações de sincronização espacial e temporal entre nós. Diferentes dos hiper-elos do modelo original, os elos passam agora a definir relações entre eventos e não entre âncoras de nós. Âncoras passam a ser simplesmente parte da definição de um evento.

Elos passam a ter, como atributo, condições e ações associadas aos seus eventos, definidos nos pontos terminais de origem e destino, respectivamente. A definição das condições e ações passam a fazer parte do NCM através da criação do objeto ponto de sincronização. Todas as condições e ações, simples e compostas, são definidas na versão atual do NCM.

A necessidade de especificação de mudanças comportamentais durante a exibição de um documento, bem como a necessidade de se permitir várias apresentações diferentes para um mesmo nó, levaram à definição de um novo conceito, o descritor. Embora mencionado em [SoCR95], já como resultado do trabalho aqui apresentado, a definição do descritor é finalmente introduzida na versão atual do NCM. As operações de mudanças de comportamento são associadas aos eventos, cabendo ao descritor definir essas associações.

Ações de elos vão poder inibir ou desinibir operações de mudanças de comportamento. Elos também podem indicar o descritor que um determinado evento de exibição de seu ponto terminal de destino usará. Todo o relacionamento entre elos, eventos, mudanças de comportamento e descritores é especificado na versão atual do NCM.

Também mencionadas na versão original, as entidades virtuais são definidas com maior precisão na versão atual, embora ainda longe da precisão ideal. Entre as entidades virtuais definidas encontra-se a classe de elos genéricos.

Na versão anterior do NCM foi introduzido o conceito de objetos de armazenamento, objetos de dados e objetos de representação. Apesar de introduzidos, todo o modelo, incluindo aí o controle de versões, só leva em consideração os objetos de armazenamento e os de dados. Na realidade, sua hierarquia de classes foi feita para objetos de representação, supondo que não poderia haver duas representações diferentes para o mesmo objeto de dados, apesar do controle de versões contestar isso. Pensava-se no NCM como constituído do plano de exibição (objetos de dados ou de representação, não importa qual, uma vez que havia uma relação biunívoca entre eles) e do plano de armazenamento.

Ao se permitir, de fato, várias representações de um mesmo objeto de dados, foi exigido um refinamento total das definições de objetos de dados, representação e armazenamento, que antes eram mencionados, mas não hierarquizados, o que causava várias confusões. Entre

elas, a relação de inclusão: podia um contexto objeto de representação incluir nós objetos de dados e de armazenamento?

Os objetos de armazenamento, de dados e de representação estão intimamente relacionados por operações de versionamento. Vários desses relacionamentos, no entanto, não eram sequer citados, com por exemplo o relacionamento entre os estados de um objeto de dados e de representação.

A replicação de elos na criação de versões não era mencionada na versão anterior do NCM. Quando se permite a um objeto de dados gerar vários objetos de representação, a definição rigorosa dessa replicação torna-se de grande importância.

Várias foram as operações para controle de versões de uma base privada definidas na versão anterior do NCM. A introdução do conceito de que nós de representação são versões de nós objetos de dados e que nós objetos de dados podem ser associados a diferentes descritores para formar nós de representação diferentes, traz uma série de modificações às operações anteriormente definidas. Todas as operações tiveram que ser cuidadosamente revistas. Operações em bases privadas objetos de dados passaram a ser completamente diferentes das mesmas operações em bases de dados objetos de representação. Enfim, todo o controle de versão foi reformulado de forma a se adequar aos requisitos especificados na versão anterior do NCM, mas que não foram nela resolvidos.

A introdução do conceito de composição como uma subclasse da classe nó, cujo conteúdo é uma lista *L* de nós e elos, onde um nó pode estar contido mais de uma vez, permitiu a definição de trilhas, entidade de grande importância na orientação de um usuário.

O conceito de perspectiva também foi estendido, exigindo que o nó mais externo de uma perspectiva não esteja contido em qualquer outro nó. Esta extensão foi necessária para os cálculos da distância realizados pelo algoritmo visão olho de peixe, responsável pela filtragem de nós e elos nos métodos de exibição das bases privadas e hiperbase pública.

Não só a definição de métodos de exibição, mas a definição de métodos para edição das composições foi incorporada à versão atual do NCM. Um modelo conceitual deve representar não apenas os conceitos estruturais dos dados, mas também definir operações

sobre os dados para manipulação e atualização das estruturas. Assim, para todo nó *C* de composição, foi definida a pertinência dos métodos para criação, exibição e seleção de seus componentes.

Como mencionado, o conceito de trilhas foi introduzido com o intuito de melhor orientar o usuário, reduzindo seu espaço navegacional. Para tanto, não só os atributos da classe trilha foram definidos, mas também métodos para navegação sobre trilhas.

As diversas formas de manipulação de trilhas também são discutidas na nova versão do NCM e sugerida a criação de uma trilha especial, denominada *trilha especial do sistema*, que mantém a história de navegação do usuário. O uso apropriado dessa trilha, em conjunto com outras trilhas do sistema, levou também à introdução de novos métodos para ativação e desativação de trilhas.

Finalmente, ao se definir o submodelo de apresentação, toda a estrutura e operação da máquina NCM de apresentação, denominado no texto Formatador Temporal, teve de ser definida. O NCM, na sua versão atual, traz agora de fato embutido todo um conceito dinâmico de apresentação temporal e espacial facilmente implementável, como demonstra [Rodr97].

3.2. O Modelo de Contextos Aninhados

O objetivo do sistema HyperProp é fornecer um ambiente para a construção de aplicações hipermídia através de uma biblioteca de classes que reflitam seu modelo conceitual. A descrição que se segue é, assim, uma descrição destas classes e de sua funcionalidade.

A possibilidade de representar referências arbitrárias entre partes quaisquer de um documento representa, para muitos, a característica marcante de um sistema hipertexto. No entanto, é necessário combinar esta flexibilidade com mecanismos de organização do documento, a fim de evitar o problema conhecido como “desorientação do usuário”. Para isso são necessários mecanismos para definição de diferentes visões de um mesmo documento, mecanismos para organização hierárquica (linear ou não) de documentos e mecanismos para permitir a definição das relações de referência do documento

independente do conteúdo de seus dados, de forma a permitir o reuso dos dados sem a herança obrigatória das relações, como acontece na maioria dos sistemas hipermídia atuais, exemplificados pelo WWW[7]. Os modelos com nós de composição fornecem suporte a tais mecanismos, em especial modelos que permitem o aninhamento destes nós.

Original na definição de composições aninhadas [Casa91], a definição de documentos hipermídia no NCM é baseada nos conceitos usuais de nós e elos. *Nós* são fragmentos de informação e *elos* são usados para a definição de relacionamentos entre os nós que interligam. No entanto, os elos não são a única forma de definição de relacionamentos, como se evidenciará a seguir.

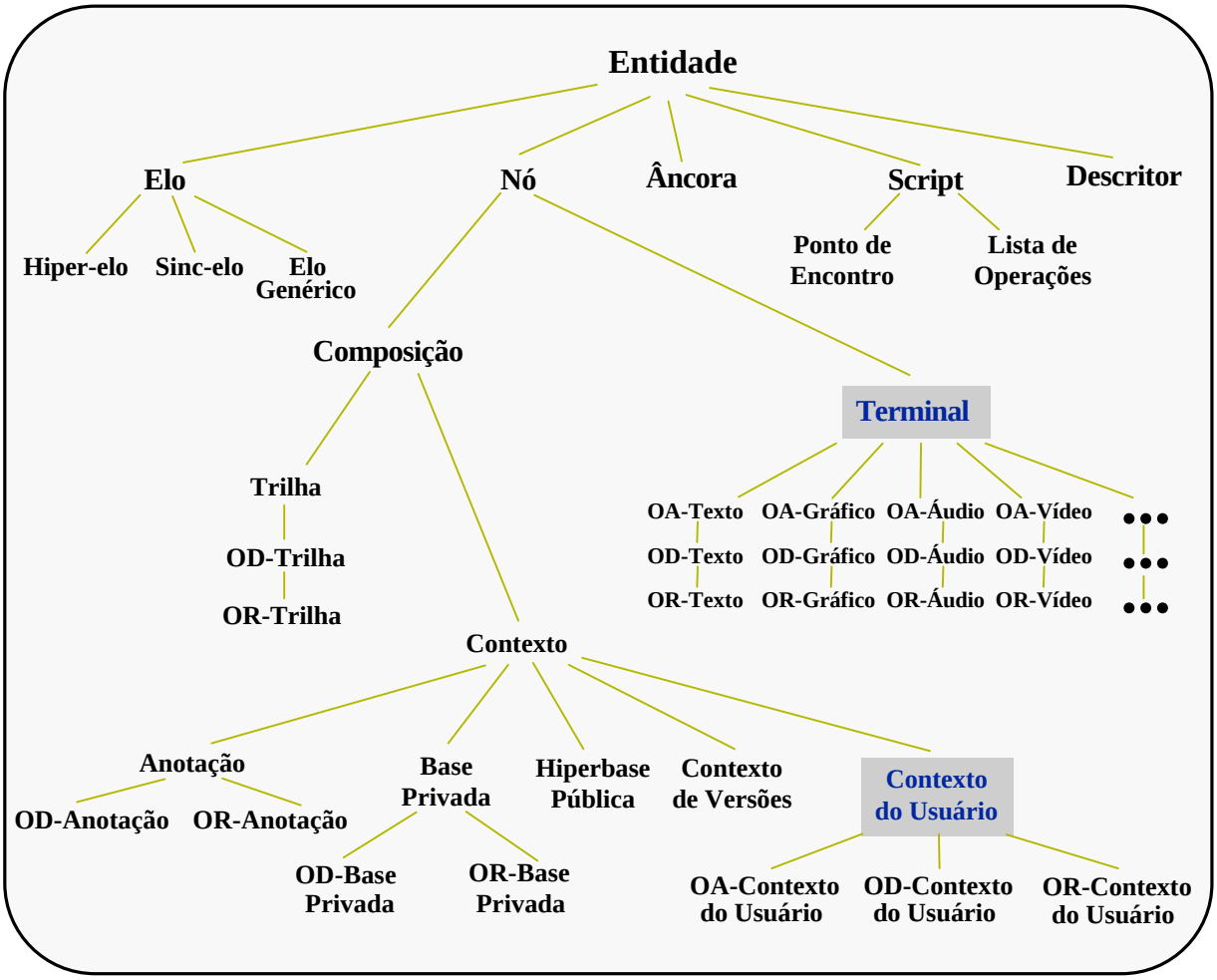


Figura 10 - Hierarquia de Classes do NCM

[7]Uma discussão desses problemas podem ser encontrados em [Hala88] e [STCN92]. Um exemplo das dificuldades encontradas na utilização de sistemas onde os elos (referências) são definidos ligados ao conteúdo das informações, pode ser encontrado em [BaRS95].

O modelo distingue duas classes básicas de nós, chamados de nós terminais (ou de conteúdo) e nós de composição, sendo estes últimos o ponto central do modelo. A Figura 10 ilustra a hierarquia de classes proposta (compare com a hierarquia de classes da versão anterior do NCM implementada, ilustrada na Figura 9 da Seção 3.1).

Uma *entidade* permite que pares atributo/valor sejam associados a qualquer objeto. Cada entidade possui um identificador único (UID), hora de criação, autor e uma lista de controle de acesso (ACL). Cada entrada nesta lista associa um usuário, ou grupo de usuários, aos seus direitos de acesso a cada atributo do objeto.

3.2.1. As Classes Nó e Âncora

Um *nó* é uma entidade que tem como atributos adicionais um conteúdo, uma lista ordenada de âncoras e um descritor.

O *conteúdo* de um nó é um conjunto de unidades de informação. A noção exata do que constitui uma unidade de informação (UI) é parte da definição do nó e depende de sua classe.

Cada elemento da *lista ordenada de âncoras* é uma entidade âncora, chamada âncora do nó. Todo objeto *âncora* possui um atributo adicional chamado região. Uma *região* 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. Dessa forma, as âncoras podem impedir que modificações no conteúdo de um nó se reflitam em outras entidades que a referenciam. Tome como exemplo um nó de texto com uma âncora cuja região especifique seu segundo parágrafo. Qualquer mudança no texto, por exemplo, a eliminação de um parágrafo, deve se refletir na região da âncora, mas não deverá afetar as demais entidades que referenciam a região. Uma exceção a essa “blindagem” oferecida pela âncora será discutida na definição do nó de composição. É

importante notar que uma âncora é identificada pelo seu UID, mas a referência a uma âncora será sempre feita pela identificação do nó (UID do nó) e a posição da âncora na lista ordenada de âncoras.

O *descritor* de um nó contém informação determinando como ele deve ser apresentado, no tempo e no espaço, similar à *presentation specification* do modelo Dexter [HaSc90]. O atributo descritor de um nó contém um objeto descritor, a ser definido na Seção 3.2.5, ou o valor nulo.

Um *nó terminal (ou de conteúdo)* é um nó cujo conteúdo e conjunto de âncoras é dependente da aplicação. O modelo permite que a classe de nós terminais seja especializada em outras classes (OA-texto, OA-áudio, OA-imagem, etc.). 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 C* é um nó cujo conteúdo é uma lista *L* de nós e elos, que se constituem em suas unidades de informação, tais que todo nó base de cada elo em *L* ou é a própria composição *C*, ou um nó de *L* (a definição de elo e nó base de um elo é dada logo a seguir; as definições de elo e composição são de fato mutuamente recursivas). Note que a região de uma âncora de *C* é uma sublista de *L*, ou toda a lista *L*.

Diz-se que uma entidade *E* em *L* é um *componente de C* e que *E está contido em C.* Diz-se também que um nó *A* está *recursivamente contido em B* se e somente se *A* está contido em *B* ou *A* está contido em um nó recursivamente contido em *B*. Deve-se também notar que os componentes de *C* podem ser ordenados (lista ordenada), o que será útil na definição de operações de navegação. Note também que uma entidade pode estar contida mais de uma vez em *L*. Uma restrição importante é feita: um nó não pode estar recursivamente contido em si mesmo.

O conjunto de âncoras de um nó de composição tem o mesmo papel que aquele de um nó terminal, exceto que as âncoras não necessariamente impedem que mudanças no conteúdo do nó de composição se reflitam em outras entidades. Em particular, os elos são

particularmente sensíveis às mudanças no conteúdo de um nó de composição. Esta observação será esclarecida na Seção 3.2.2.

Nós de composição diferentes podem conter um mesmo nó e nós de composição podem ser aninhados em qualquer profundidade, desde que a restrição de um nó não conter recursivamente a si mesmo seja obedecida. Para identificar através de que seqüência de nós de composição aninhados uma dada instância de um nó *N* está sendo observada, é introduzida a noção de perspectiva de um nó. 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 pode 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 a posteriori). Dada a perspectiva P = (N_m,....,N_1), o nó N_1 é chamado *nó base da perspectiva*.

Nós de composição são objetos cuja semântica é bem conhecida pelo modelo. Um modelo conceitual deve representar não apenas os conceitos estruturais dos dados, mas também definir operações sobre os dados para manipulação e atualização das estruturas. Assim, todo nó *C* de composição deve possuir os seguintes métodos:

1. *Cria nó*: insere um nó na lista de nós da composição.
2. *Retira nó*: retira um nó da lista de nós da composição.
3. *Cria elo*: insere um elo na lista de elos.
4. *Destrói elo*: retira o elo da lista de elos e o destrói.
5. *Exibe estrutura* (definido no descritor da composição): exibe toda a estrutura de nós e elos (grafo) da lista de nós e elos da composição. Normalmente o método exibidor também é o editor através do qual todas as outras operações do nó de composição podem ser ativadas.
6. *Seleciona elo*: instancia (se já não instanciado) e chama o método exibidor (que normalmente é também editor) do elo, pertencente a lista de elos.

7. *Seleciona nó*: instancia (se já não instanciado) e chama o método exibidor (que normalmente é também editor) do nó, pertencente a lista de elos.
8. *Seleciona nó pai*: instancia (se já não instanciado) e chama o método exibidor da composição que contém o nó *C*, dentro da perspectiva corrente.
9. Métodos para mudança de estado do nó *C* serão definidos quando da introdução da noção de estado de um nó.

Parâmetros como grande número de nós, grande número de elos, muitas mudanças no documento, tempo de resposta ruim para as ações do usuário, diferenças visuais insuficientes entre elos e nós, e usuários não orientados visualmente, se combinam para dificultar os mecanismos de navegação em um documento. Usuários desorientados precisam de informações de escopo para restabelecerem a noção de localização. Em particular dois tipos de escopo são necessários: escopo espacial, que responde à questão "para onde posso ir agora?", e escopo temporal, que responde à questão "como eu cheguei aqui?". A primeira questão é tratada no Capítulo 4 e em [Much96], a segunda encontra sua resposta com a introdução do conceito de trilha.

Dado um nó de composição *C*, uma *trilha T para C* é um nó de composição contendo apenas nós, tais que todos os nós estão recursivamente contidos em *C* e todos os seus nós trilhas são trilhas para *C*. Os nós componentes de uma trilha devem formar uma lista ordenada. Mais ainda, *T* tem um atributo denominado *nó-corrente*, cujo valor é um inteiro indicando uma posição na lista ordenada de *T*, chamada de *entidade corrente de T*, e um atributo denominado *visão*, cujo valor associa a cada ocorrência de um nó *N*, na lista de *T*, um aninhamento $(N_m,....,N_1)$, com $m \geq 1$, tal que: $N_1 = N$, $N_m = C$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para $i \in [1,m)$. Diz-se que a trilha *T* é *associada* a *C*.

Toda trilha deve possuir, adicionalmente aos métodos de toda composição, os seguintes métodos:

- *próximo*: incrementa o atributo nó-corrente
- *anterior*: decrementa o atributo nó-corrente
- *primeiro*: retorna à primeira entidade da trilha (atributo nó-corrente igual a um)

- *último*: faz o atributo nó-corrente apontar para o último elemento da trilha
- *insere*: insere um nó após o último elemento da trilha

Além dos métodos citados, toda trilha deve possuir os métodos ativa e desativa, que inibe ou desinibe as ações dos outros métodos. Assim:

- *ativa*: desinibe os métodos próximo, anterior, primeiro e último; e inibe os métodos insere cria nó e retira nó.
- *desativa*: inibe os métodos próximo, anterior, primeiro e último; e desinibe os métodos insere, cria nó e retira nó.

A razão dos métodos ativa e desativa é não permitir que uma trilha seja usada ao mesmo tempo para navegação (pela trilha) e para manter o histórico da navegação. Ou ela é usada para a realização de uma tarefa ou da outra.

A definição de trilhas [Zell89] é uma ferramenta importante para sistemas hipermídia, pois representam travessias ordenadas de alguns elos do documento. Através delas, os autores podem fornecer ordens de leitura, que auxiliam leitores não familiarizados com o material informativo a navegar pelo hiperdocumento, ou determinar uma ordem apropriada de apresentação para uma certa audiência. Os usuários sentem-se menos desorientados quando seguem uma trilha já definida, pois têm limitado o número de opções que possuem para percorrer o documento.

Caso a trilha sugira uma ordem linear de leitura do documento, ela deve ser previamente construída pelo autor, que deverá inserir nós na lista, de acordo com a ordem em que estes devem ser exibidos.

Especificamente falando do sistema HyperProp, uma trilha especial, denominada *trilha especial do sistema*, mantém a história de navegação do usuário. Toda vez que um usuário navega para um nó, esse nó, e sua perspectiva corrente, são inseridos na trilha pelo sistema. Note que esta é uma das maneiras de se criar uma trilha. Se um usuário acionar o método ativa da trilha especial do sistema, é criada uma trilha cópia sobre a qual a navegação por trilhas terá efeito. Mesmo durante a navegação nessa cópia, a trilha especial do sistema é

atualizada, de forma a sempre manter o histórico da navegação. Se o usuário fizer qualquer navegação fora da dada pela trilha cópia, essa trilha é destruída.

Um *nó de contexto T* é um nó de composição tal que seu conteúdo contém apenas elos, nós terminais e nós de contextos, sem repetição de componentes. Os nós de contexto possuem um atributo adicional chamado conjunto de descritores. Um *conjunto de descritores* contém, para cada nó contido em *T*, um *descritor* ou o valor nulo. A noção de *descritor*, como já mencionado, será definida na Seção 3.2.5. Infelizmente as definições das classes do NCM são mutuamente recursivas, o que torna a tarefa de descrevê-las difícil de ser realizada em um texto linear. Este trabalho é um exemplo típico onde o paradigma de hipertexto seria de grande auxílio.

Nós de contexto podem ser especializados nos nós anotação, hiperbase pública, base privada, contexto de versão e contexto do usuário, como se verá após a discussão e a definição da classe elo.

3.2.2. Elos, Script, Ponto de Encontro e Eventos

Um *elo* é uma entidade que possui quatro atributos adicionais: o conjunto de pontos terminais de origem, o conjunto de pontos terminais de destino, o ponto de encontro e o conjunto de descritores. O conjunto de pontos terminais fonte e destino definem eventos, o ponto de encontro vai definir relações entre eventos e os descritores vão definir como os nós correspondentes aos eventos de destino devem ser apresentados.

A Seção 5.3.2 apresenta um esquema que emprega a Técnica de Descrição Formal RT-LOTOS [CoOl94] para especificar as entidades NCM que capturam a especificação do sincronismo temporal dos documentos. A especificação RT-LOTOS destas entidades NCM define formalmente a semântica do modelo no aspecto sincronização temporal.

Um *evento* é 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 nó ou da condição de habilitação das mudanças de comportamento definidas no objeto descritor, chamado *evento de atribuição* (como se verá na Seção 3.2.5). O início ou fim de um evento é denominado *ponto de sincronização*.

Um evento pode estar em um dos seguintes estados: dormindo, preparando, preparado, ocorrendo e suspenso. Além disso, todo evento possui uma variável ocorrido, inteira, que indica quantas vezes o evento, em uma dada apresentação, passou do estado ocorrendo para o estado preparado, sendo responsabilidade do Formatador Temporal a manutenção dessa variável. 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, assunto da Seção 3.2.5.

Intuitivamente, tomando o evento de exibição como exemplo, o evento parte, inicialmente, do estado dormindo. Ele entra no estado preparando quando algum processamento de busca das unidades de informação é realizado. Ao término desse processamento, o evento entra no estado preparado. Ao iniciar o evento propriamente dito, por exemplo, ao iniciar a exibição, o evento entra no estado ocorrendo. A ocorrência do evento pode ser suspensa temporariamente, quando ele passa para o estado suspenso. Ao término do evento, ao terminar a exibição no caso do exemplo, o evento volta ao estado preparado.

A manutenção do estado dos eventos é responsabilidade do Formatador Temporal do documento, assunto do Capítulo 5 e da referência [Rodr97]. A Figura 11 apresenta a máquina de estados para os três tipos de evento.

Os valores dos atributos *conjunto de pontos terminais de origem* e *conjunto de pontos terminais de destino* de um elo são conjuntos cujos elementos, chamados *pontos terminais* de um elo, são quádruplas da forma: $<(N_k,..., N_2, N_1), A, \alpha, tipo>$ 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 N_1 for um contexto de versão (que será definido na Seção 3.2.6.2), caso contrário A é nulo (ϕ).
3. α é uma âncora ou a identificação de um atributo (e seu valor) de um nó N. Se N_1 for um contexto de versão, N é qualquer nó identificado por A; caso contrário, $N = N_1$.

4. *tipo* especifica o evento (de exibição, seleção ou de atribuição) associado a α. No caso de evento de atribuição, o valor de α deve identificar o atributo e seu valor, ou uma âncora (no caso de alteração da condição de habilitação das mudanças de comportamento definidas no objeto descritor, como se verá na Seção 3.2.5). No caso de evento de exibição ou seleção, α é sempre uma âncora.

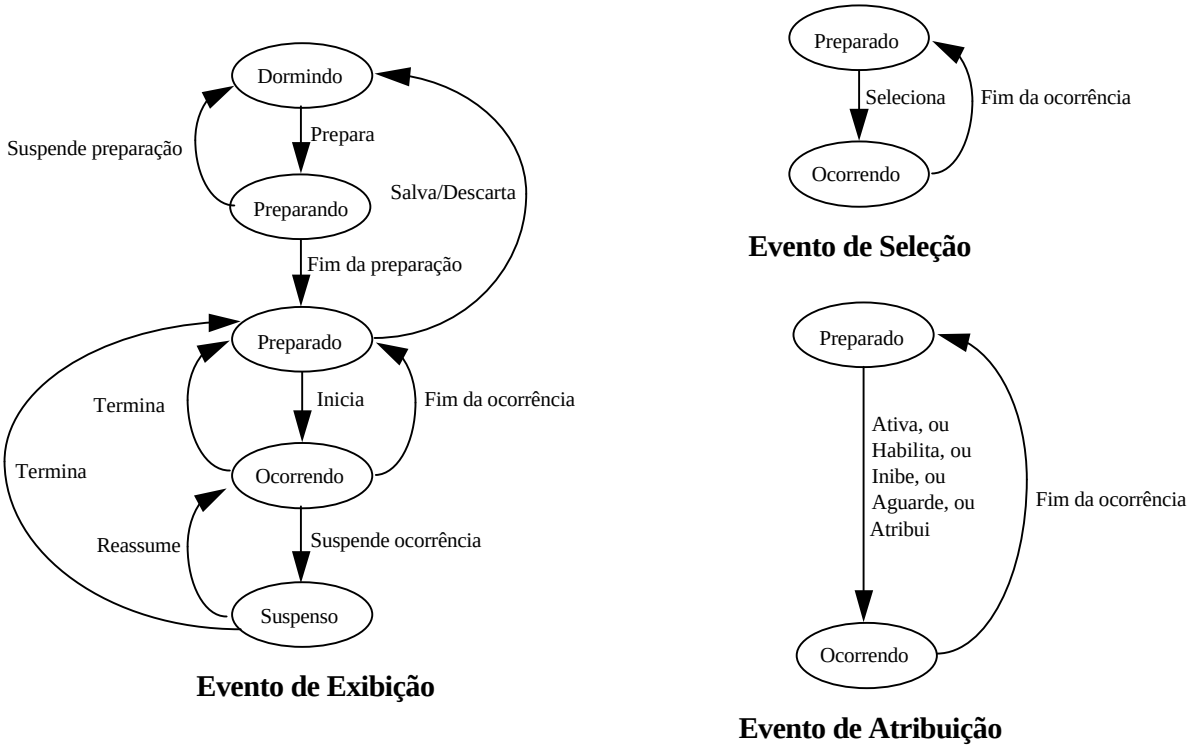


Figura 11 - Máquina de estados dos diversos tipos de eventos

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

Um elo deve ser interpretado como uma asserção lógica sobre uma coleção de eventos. Os relacionamentos entre eventos associados a um elo são expressos através do valor do seu atributo *ponto de encontro*, que é uma especialização da classe *script*. Um objeto *script* possui um atributo denominado *conteúdo*, que contém uma seqüência ordenada de operações. Um objeto *ponto de encontro* é um script, que contém uma única operação composta por uma condição e uma ação. Cada condição satisfeita implica no disparo da ação a ela associada. Ações de pontos de encontro são operações que devem ser executadas

nos pontos terminais de destino dos elos ou ações de espera de um tempo (retardo). Condições dizem respeito aos pontos terminais de origem.

As condições de um ponto de encontro avaliam valores booleanos e podem ser binária simples ou compostas. Toda condição binária simples (ou simplesmente condição simples) é expressa por duas condições unárias: uma condição prévia, a ser satisfeita imediatamente antes do instante de tempo onde a condição é avaliada, e uma condição corrente, a ser satisfeita no instante de tempo onde a condição 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 usados na avaliação das condições simples são:

- = comparação de igualdade
- $\neq$ comparação de desigualdade
- < comparação de inferioridade
- $\leq$ comparação de igualdade ou inferioridade
- > comparação de superioridade
- $\geq$ comparação de igualdade ou superioridade

As comparações são realizadas com respeito a:

1. Estados de um evento *E*
2. A variável *ocorrido* associada a um evento *E*
3. Atributos de um nó, no caso de um evento de atribuição, e o valor especificado no ponto terminal de origem por α.

Qualquer expressão lógica de condições baseada nos operadores $\wedge$ (e), $\vee$ (ou), $\neg$ (negação) e $\oplus$ (retardo) define uma condição composta. Os operadores lógicos $\wedge$, $\vee$ e $\neg$ possuem os significados usuais da lógica proposicional. Uma expressão E $\oplus$ (retardo) é satisfeita em um tempo *t* se a condição *E* estava satisfeita em *t-retardo*.

As ações de um ponto de encontro podem ser simples ou compostas. As ações simples podem ser:

1. *Inicia* evento *E* (no caso de evento de exibição, pode opcionalmente especificar um descritor do conjunto de descritores do elo associados ao nó âncora), que passa para o estado ocorrendo.
2. *Suspende* evento *E*, que passa para o estado suspenso.
3. *Reassume* evento *E*, que passa para o estado ocorrendo.
4. *Termina* evento *E*, que passa para o estado preparado.
5. *Prepara* evento *E* (no caso de evento de exibição, pode opcionalmente especificar um descritor do conjunto de descritores do elo associados ao nó âncora), que passa para o estado preparando.
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 α identificar uma âncora. As operações são ativadas se estiverem habilitadas.
7. *Habilita* usada para habilitar as operações de mudança de comportamento associadas ao evento *E* (operações de mudança de comportamento serão definidas nos objetos da classe descritor). O evento *E* deve ser do tipo *atribuição* e α identificar uma âncora.
8. *Inibe* usada para inibir as operações de mudança de comportamento associadas ao evento *E* (operações de mudança de comportamento serão definidas nos objetos da classe descritor). O evento *E* deve ser do tipo *atribuição* e α identificar uma âncora.
9. *Aguarde* (retardo) usada para se aguardar um tempo especificado.
10. *Atribui* (valor) usada para atribuir o valor especificado pela âncora α de um ponto terminal de destino com evento do tipo atribuição ao atributo também especificado por α.

Uma ação composta é definida por uma expressão que combina ações *Aguarde* com as outras ações usando os operadores | (paralelo) e → (seqüencial) para definir a ordem de execução de cada elemento ação da expressão.

A classe elo é especializada em duas subclasses: hiper-elo e sinc-elo. Um hiper-elo é um elo que possui pelo menos um evento de seleção associado a um dos elementos de seu conjunto de pontos terminais de origem. Um sinc-elo é um elo cujos pontos terminais de origem não estão associados a eventos de seleção.

Elos são sempre direcionais, embora possam ser percorridos em qualquer direção. Múltiplos pontos terminais de origem e de destino permitem definir relações m:n. Como um elo *l* pode ter um ponto terminal da forma <(N_k,..., N_2, N_1), *A*, α, *tipo*>, onde *l* e o nó base N_k pertencem ao mesmo nó de composição, mas o nó âncora N_1 está recursivamente contido (mas não diretamente contido) em N_k, se o conteúdo de N_{i+1} for alterado, por exemplo pela eliminação do nó N_i, o elo *l* terá de sofrer mudanças. Isto é, a entidade — o elo *l* — não é preservada e mudanças no conteúdo de N_{i+1} provocam mudanças na entidade.

O NCM permite que o sincronismo de uma apresentação seja opcionalmente especificado através de um objeto script. A função de um objeto script é encapsular um programa escrito em uma linguagem cujas instruções, quando executadas, geram mensagens invocando a ativação de ações NCM válidas. O modelo requer que o objeto script possua todos os atributos e métodos necessários para interpretar e controlar sua própria execução.

3.2.2.1. Exemplo de Diversas Possibilidades de Definição de Elos

Tome o exemplo originalmente apresentado em [SoCR95] e aqui apropriadamente modificado. Considere um documento de trabalho do Grupo de Pesquisa de Poesias Inglesas do Século XVI. Suponha que o documento é modelado como um contexto *E* contendo um contexto *S,* agrupando peças de Shakespeare, e um outro contexto *M,* agrupando sonetos de Christopher Marlowe. Suponha também que *S* contém os nós de texto *H* e *L* representando as peças “Hamlet” e “King Lear” e que *M* contém um nó de texto *F* representando “Dr. Faustus”. Pode ser que o grupo deseje registrar uma conexão entre “Hamlet” e “Dr. Faustus” (tal elo poderia ser usado, por exemplo, para registrar uma

conexão entre peças onde o tema principal fosse o conflito). Este elo poderia ser definido em *E* como (<(*S,H*), ϕ, *r, seleção*>,<(*M,F*), ϕ, *s, exibição*)>), onde as âncoras *r* e *s* permitem uma conexão mais precisa, por exemplo, apontando para as sentenças onde o conceito comum apareceu primeiramente. Uma seleção da âncora *r* pode acarretar (especificada pela ação do ponto de encontro) a exibição do nó *F*, a partir de *s*.

Em outro exemplo, uma vez que *S* contém *H* e *L*, um elo conectando esses nós poderia, em princípio, ser definido em *S* como (<(*H),* ϕ, *i, seleção*>,<(*L),* ϕ, *j, exibição*>), onde *i* e *j* são âncoras válidas para *H* e *L*. Caso fosse criado em *E*, o elo conectando *H* e *L* seria definido como (<(*S,H*), ϕ, *i, seleção*>,<(*S,L*), ϕ, *j, exibição*>). Note a diferença em se definir o elo em *S* e em *E*. O elo definido em *S* será herdado por todos os documentos que contiverem *S* (o contexto que agrupa as peças de Shakespeare será provavelmente compartilhado por vários documentos), enquanto que o elo definido em *E* será herdado por *S* apenas para os leitores do documento *E*.

Poderíamos ainda definir no mesmo exemplo os seguintes elos: em *S* os elos L_1 = (<(*H),*ϕ, *i, seleção*>,<(*S),* ϕ, λ, *exibição*>) e L_2 = (<(*S*), ϕ, λ, *exibição*>,<(*L*), ϕ, *j, exibição*)>); e em *E* o elo L_3 = (<(*S*), ϕ, λ, *exibição*>,<(*S*), ϕ, λ, *exibição*)>), com os seguintes pontos de encontro: PL_1 = {condição: <evento de seleção ocorrendo, evento de seleção preparado>; ação: inicia exibição}*;* PL_3 = {condição: <verdade, ocorrendo evento de exibição>; ação: aguarde (10 segundos)→ termina a exibição}; e PL_2 = {condição: <evento de exibição ocorrendo, evento de exibição preparado>; ação: inicia exibição}. Embora o efeito final fosse o mesmo do exemplo anterior, isto é, a seleção da âncora *i* ocasiona a exibição do nó *L*, a partir da âncora *j*, entre a seleção da âncora e a exibição do nó *L*, o nó *S* seria temporariamente exibido, dando uma melhor noção de contexto ao leitor.

3.2.2.2. Elos Visíveis, Entidades e Elos Virtuais

Informalmente, a noção de elo visível já foi apresentada. 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. A esse conjunto de elos chamamos 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.

Por exemplo, suponha que os nós *A* e *Z* contenham o nó *B*, que por sua vez contém os nós *C, D, E* e *F*. Assuma também que os nós de composição contêm os seguintes elos:

A: <(*B*,*C*), ϕ, i, tipo> , <(*B*,*E*), ϕ, j, tipo>

Z: <(*B*,*C*), ϕ, r, tipo> , <(*B*,*D*), ϕ, k, tipo>

B: <(*E*), ϕ, s, tipo> , <(*D*), ϕ, t, tipo> e <(*C*), ϕ, m, tipo> , <(*F*), ϕ, n, tipo>

Então, a exibição da instância do nó *C*, através da perspectiva (*A,B,C*), vai mostrar um elo da âncora *i* de *C* para a âncora *j* de *E* e um elo da âncora *m* de *C* para âncora *n* de *F*, definidos em *A* e *B*, respectivamente. A exibição da instância do nó *B*, através da perspectiva (*A,B,C*), vai mostrar um elo de *B* para *B*, que é definido no nó *A*. Por outro lado, a exibição da instância do nó *C*, através da perspectiva (*Z,B,C*), vai mostrar um elo a partir da âncora *r* de *C* para a âncora *k* de *D* e um elo a partir da âncora *m* de *C* para a âncora *n* de *F*, definidos em *Z* e *B*, respectivamente. A exibição da instância do nó *B*, através da mesma perspectiva (*Z,B,C*), vai mostrar um elo de *B* para *B*, definido no nó *Z*.

Introduz-se aqui o conceito de entidade virtual, cuja definição formal está fora do escopo desse trabalho. Uma *entidade virtual X* é uma entidade que tem como valor de pelo menos um de seus atributos *A* uma expressão *E*, escrita em uma linguagem de consulta formalmente definida, cuja avaliação resulta em um objeto ou valor de tipo apropriado. O atributo *A* é chamado *virtual*. Quando alguma operação requer o valor de *A*, o valor resultante da avaliação de *E* é retornado. Por exemplo, um nó virtual pode ter seu conteúdo e regiões de suas âncoras definidos por expressões.

No caso particular das âncoras, uma *âncora virtual*, em um conjunto de âncoras de um nó *N*, é uma âncora cuja região é definida por uma expressão *r*, computada quando um elo que a possui em seu ponto terminal é percorrido. A expressão *r* deve, nesse caso, resultar em um conjunto de unidades de informação dentro do nó *N*, incluindo o símbolo especial λ,

representando todo o conteúdo do nó. Na discussão sobre contextos de versões, na Seção 3.2.6.2, ter-se-á um bom exemplo de uso de âncora virtual.

Embora as únicas expressões definidas hoje na linguagem de consulta do sistema HyperProp digam respeito às âncoras virtuais em contextos de versões (Seção 3.2.6.2), é interessante aqui salientar as potencialidades dos elos virtuais, ficando como sugestão para trabalhos futuros a extensão da linguagem de consulta hoje existente e ainda extremamente limitada.

Um *elo virtual* é um elo *l* que tem pelo menos um dos seus pontos terminais definidos por uma expressão *e* que será computada quando o elo for selecionado. Se *l* está contido em um nó de composição *C*, então *e* deve retornar um nó base que seja *C* ou um nó contido em *C*, como definido pela Seção 0. Elos virtuais constituem-se em uma ferramenta de autoria poderosa, como a encontrada no sistema Microcosm [FHHD90].

Mais especificamente ainda, pode-se definir um *elo genérico* para nós das classes $\boldsymbol{T}_1,...,\boldsymbol{T}_m$ como um elo cujo conjunto de pontos terminais de origem contém expressões $\boldsymbol{e}_1,...,\boldsymbol{e}_m$ tal que $\boldsymbol{e}_i$ retorna uma região definida em âncoras de $\boldsymbol{T}_i$, para $i \in [1,m]$. Mais precisamente, o ponto terminal de origem que contém a expressão $\boldsymbol{e}_i$ tem a sua quádrupla $<(N_k,..., N_2, N_1)$, $A, \alpha, tipo>$ avaliada da seguinte forma:

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. N_k é o nó base do elo.
2. *A* é uma âncora de N_1, se N_1 for um contexto de versão (que será definido na Seção 3.2.6.2), caso contrário *A* é nulo (ϕ).
3. α é uma âncora de um nó *N*, cuja região r_i casa com o resultado da avaliação da expressão $\boldsymbol{e}_i$. Se N_1 for um contexto de versão, *N* é qualquer nó identificado por *A*; caso contrário, $N = N_1$.
4. *tipo* especifica o evento (de exibição ou seleção) associado a α.

Considera-se, então, que *l* conecta qualquer tupla de nós $M_1,...,M_m$ nas classes $\boldsymbol{T}_1,...,\boldsymbol{T}_m$ que têm âncoras cujas regiões $r_1,...,r_m$ casam com resultados da avaliação das expressões $\boldsymbol{e}_1,...,\boldsymbol{e}_m$, respectivamente, aos pontos de destino especificados no elo.

Intuitivamente, um elo é genérico quando não se aplica a um único documento, mas pode ser usado por uma classe de documentos. Por exemplo, pode-se definir um elo que tem como ponto terminal de destino um nó de vídeo cujo conteúdo é uma visão aérea da Floresta Amazônica, e como ponto terminal de origem a expressão ***Floresta Amazônica***. Assim, qualquer nó de texto com uma âncora cuja região contém a sentença *Floresta Amazônica* terá um elo automático com esse nó de vídeo.

A implementação do conceito de elos genéricos não é trivial. Seja *N* um nó representando um dado documento. Não parece razoável tentar casar todo elo genérico com *N*, criando as âncoras apropriadas, se o número de elos genéricos for muito grande. Seria também impraticável, se o conteúdo de *N* for grande, tentar casar cada possível região de *N* contra a coleção de elos genéricos. Uma alternativa e sugestão para a implementação de elos genéricos no sistema HyperProp é permitir ao usuário marcar uma área no documento e selecionar de um menu uma operação que tenta casar todas as possíveis regiões na área selecionada (e não o nó inteiro) contra a coleção de elos genéricos, sendo os possíveis casamentos apresentados ao autor. O autor, então, para cada região encontrada, pode criar uma âncora α para a região e um novo elo tendo a mesma definição do elo genérico encontrado pelo casamento, exceto que o elo terá α como a âncora do ponto terminal de origem. O elo pertenceria a qualquer nó de composição que recursivamente contivesse os nós âncora, como requerido pelo modelo.

3.2.3. Nós de Contexto

Um *nó de contexto T*, como já mencionado, pode ser especializado em outros tipos de nós, um dos quais o nó de contexto de usuário. Um *nó de contexto de usuário* é um nó de contexto cujo atributo conteúdo contém apenas nós terminais, nós contextos de usuário e elos. Intuitivamente, esses foram os nós de contexto que veio se considerando até agora no transcorrer deste texto. Nós de contexto do usuário vão servir, entre outras coisas, para definir uma estrutura, hierárquica ou não, para documentos, para permitir a definição de

diferentes visões de um mesmo documento e para melhorar a orientação do usuário na navegação em um documento.

A classe nó de contexto pode ser ainda especializada nos nós contexto de versão, hiperbase pública, base privada e anotação. Tais classes são acrescentadas ao modelo com a finalidade de possibilitar versionamento de documentos e trabalho cooperativo. Antes, no entanto, de se passar a discutir versionamento e trabalho cooperativo, as próximas seções introduzem uma pequena pausa na discussão de nós contexto, que será retomada na Seção 3.2.6.

3.2.4. Arquitetura de referência do Sistema HyperProp

A Figura 12 apresenta a arquitetura aberta para o tratamento de documentos do sistema HyperProp, em conformidade como Modelo Dexter [HaSc90] e o padrão MHEG [MHEG95].

Figura 12 - Arquitetura do Sistema HyperProp.

A *camada de armazenamento* implementa o armazenamento persistente de objetos multimídia/hipermídia do modelo NCM. Intuitivamente, nesta camada estarão armazenados todos os objetos do NCM de acesso público. Esses objetos estarão sempre em sua forma final (vide Seção 3.2.6), isto é, não poderão ser alterados. Os objetos da classe nó pertencentes a essa camada são chamados *objetos de armazenamento*. A camada oferece uma interface para o intercâmbio de dados, chamada *multimedia hypermedia interchangeable objects* (interface MHIO).

A interface MHIO é a chave para fornecer a compatibilidade entre aplicações e equipamentos, pois estabelece dois pontos em que as camadas de armazenamento e

apresentação devem concordar: (1) a representação codificada para objetos multimídia a serem intercambiados, correspondente ao padrão ISO MHEG; e (2) as mensagens, pedidos, confirmações etc., usados por essas camadas para solicitar um objeto, conteúdo ou ação requisitados por camadas superiores.

A *camada de aplicação* introduz os conceitos de *objetos de dados* e de *objetos de representação*. Conceitos similares podem ser encontrados em [PuGu90]. Um *objeto de dados* é criado ou como um objeto totalmente novo, ou como uma cópia local de um objeto de armazenamento, acrescida de novos atributos (não-persistentes) que são dependentes da aplicação. Ele contém métodos para manipular os novos atributos, assim como métodos para manipular a informação originalmente pertencente ao objeto de armazenamento, exceto o atributo conteúdo que não é interpretável nesse nível de abstração. O formato de armazenamento de um objeto de dados corresponde a uma representação interna concreta de um objeto MHEG.

Um *objeto de representação* é criado como uma cópia local de um objeto de dados, adicionando-se novos métodos para exibir e editar o atributo conteúdo, no formato mais apropriado para um uso particular da informação. Um objeto de representação age como uma nova versão de um objeto de armazenamento, no caso de nós contexto de usuário e nós terminais, derivada de um objeto de dados. Objetos de representação oferecem diferentes visões de objetos de dados, como será discutido na Seção 3.2.5.1.

A principal função da *camada de apresentação* é converter de/para o formato dos objetos de dados usados por aplicações e plataformas particulares e a representação codificada para objetos multimídia definida pela interface MHIO. Note que a camada de apresentação não implementa qualquer método associado com os objetos de dados.

Para ter acesso a um dado multimídia, uma aplicação procede então, resumidamente, como se segue. Usando uma linguagem de consulta ou facilidades navegacionais do sistema hipermídia, ela identifica, indiretamente, o objeto de armazenamento contendo o dado desejado e requer a criação de um objeto de representação correspondente.

A informação necessária para apresentar os componentes de um documento multimídia é fornecida pelo objeto de representação, formado pela associação de um objeto de dados e um objeto da classe descritor, como será detalhado na próxima seção. Um *descritor* especifica como um objeto de dados será exibido para o usuário, detalhando como será iniciado, qual dispositivo de E/S será utilizado e quais mudanças ocorrerão no seu comportamento durante a exibição, caso elas existam. A associação entre objetos de dados e descritores é representada na Figura 13 por linhas conectando os objetos de dados no plano inferior com os objetos de representação, desenhados no plano superior. Na figura, nós são representados por círculos, elos por arcos e composições pela inclusão de círculos e arcos em círculos maiores.

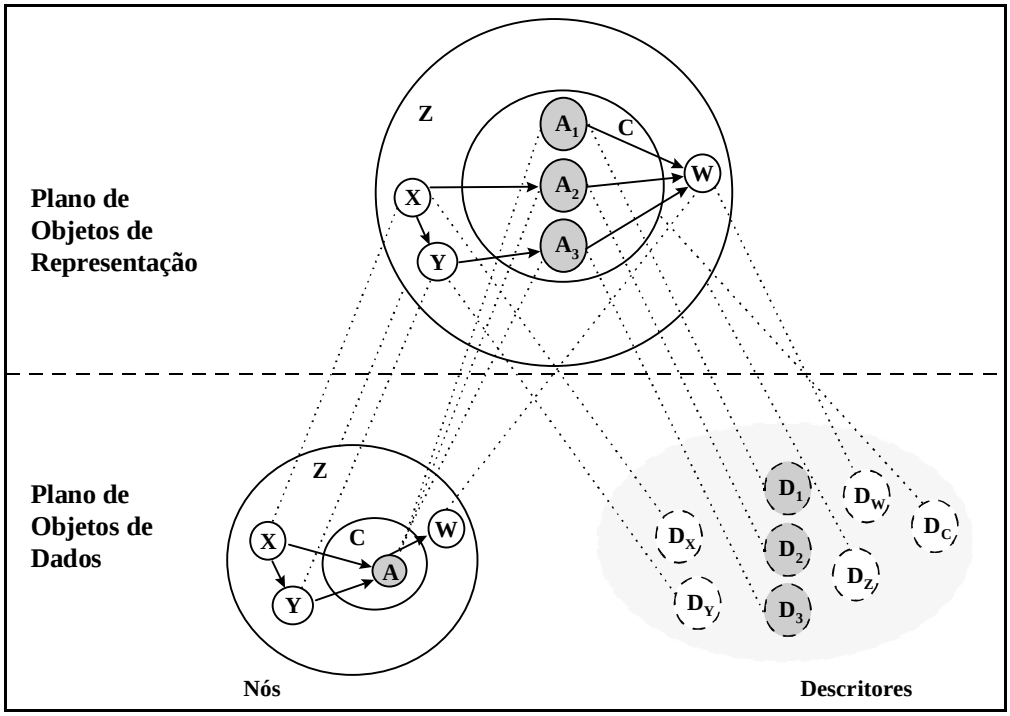


Figura 13 - Planos de Objetos de Dados e de Representação.

Note que o modelo de apresentação permite a combinação de uma entidade objeto de dados com diferentes descritores, originando diferentes representações (objetos de representação) da mesma entidade. A figura 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, por exemplo, três diferentes formas de navegação até ele, através dos dois elos ou através da hierarquia de composições. Note

assim que, devido ao fato de um mesmo objeto de dados poder gerar vários objetos de representação, um objeto de representação composição pode conter um número de elementos diferentes da composição objeto de dados; por exemplo, o contexto de usuário *C'* da figura possui três nós, ao passo que o nó objeto de dados correspondente só tem um.

3.2.4.1. Objetos e Armazenamento, de Dados e de Representação

As especializações da classe nó terminal, mencionadas anteriormente, são classes que instanciam objetos de armazenamento. Assim, por exemplo, um objeto da classe *OA-texto*, é um objeto de armazenamento (motivo do prefixo OA na definição da classe). Qualquer dessas classes podem ser especializadas em classes OD, isto é, classes cujos objetos instanciados são objetos de dados. Assim, por exemplo, a classe *OD-texto* é uma especialização da classe OA-texto, obtida pelo acréscimo de novos atributos (não-persistentes), dependentes da aplicação, e de métodos para manipular os novos atributos, assim como métodos para manipular a informação originalmente pertencente ao objeto de armazenamento, exceto o atributo conteúdo. Por sua vez, a classe OD-texto pode ser especializada na classe *OR-texto*, pelo acréscimo de novos atributos e métodos para a manipulação do atributo conteúdo do objeto de dados correspondente. Estes novos métodos e atributos são derivados a partir de um objeto descritor, como se verá na próxima seção. Por simplicidade, chama-se nó OA-terminal um nó terminal pertencente a qualquer subclasse da classe nó terminal cujos objetos instanciados são objetos de armazenamento. Analogamente, tem-se um nó OD-terminal e um nó OR-terminal.

A classe nó contexto de usuário é especializada em três subclasses: OA-contexto do usuário, OD-contexto do usuário e OR-contexto do usuário.

Um *nó OA-contexto de usuário* é um nó contexto de usuário cujo atributo conteúdo contém apenas nós OA-terminais, nós OA-contextos do usuário e elos.

Um *nó OD-contexto de usuário* é um nó contexto de usuário cujo atributo conteúdo contém apenas nós OA-terminais, nós OA-contextos do usuário, nós OD-terminais, nós OD-contextos do usuário e elos.

Um *nó OR-contexto de usuário* é um nó contexto de usuário cujo atributo conteúdo contém apenas nós OA-terminais, nós OA-contextos do usuário, nós OD-terminais, nós OD-contextos do usuário, nós OR-terminais, nós OR-contextos do usuário e elos. A classe OR-contexto do usuário possui, também, novos atributos e métodos para a manipulação do atributo conteúdo do objeto. Estes novos métodos e atributos são derivados a partir de um objeto descritor, como se verá na próxima seção.

Por simplicidade chama-se nó OA-contexto um nó contexto pertencente a qualquer subclasse da classe nó contexto cujos objetos instanciados são objetos de armazenamento. Analogamente, tem-se um nó OD-contexto e um nó OR-contexto.

A classe trilha é especializada nas classes OD-trilha e OR-trilha. Dado um nó de composição *C*, uma *OD-trilha T para C* é um nó de composição contendo apenas nós OR-contexto, nós OR-terminal, e trilhas. A classe OR-trilha, é derivada da classe OD-trilha adicionando novos atributos e métodos para a manipulação do atributo conteúdo do objeto. Estes novos métodos e atributos são derivados a partir de um objeto descritor, como se verá na próxima seção.

Por simplicidade, chama-se nó OA-composição um nó de composição pertencente a qualquer subclasse da classe nó de composição cujos objetos instanciados são objetos OA-contexto. Chama-se nó OD-composição um nó de composição pertencente a qualquer subclasse da classe nó de composição cujos objetos instanciados são objetos OD-contexto ou OD-trilha. Analogamente, tem-se um nó OR-composição.

3.2.5. Descritor

Um *descritor* é uma entidade que tem como atributos adicionais uma especificação de iniciação, uma especificação de término e uma coleção de descrições de eventos.

Uma *especificação de iniciação* contém as informações necessárias para iniciar a apresentação de uma entidade. Em particular, ela define os métodos para exibição ou edição das entidades. Esses métodos podem ser qualquer programa e, em particular, qualquer editor. Uma especificação de iniciação possui também uma lista ordenada de operações que

devem ser executadas para preparar a exibição do nó. Ela deve definir todos os parâmetros necessários para a criação de um objeto representação a partir de um objeto de dados.

Uma *especificação de término* contém as informações necessárias para finalizar a apresentação de uma entidade. Em particular, ela define os métodos que devem ser executados ao final de uma exibição. Uma especificação de término possui também 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 nó (objeto de dados) ao qual o descritor será associado. Por exemplo, para um nó texto, uma especificação de iniciação pode determinar o editor de texto *word*, podendo passar, para a criação do objeto de representação, o tamanho da janela de apresentação do texto e sua posição na tela, etc. No caso especial de uma composição, o método default de exibição é o definido na composição (método *exibe estrutura* definido na Seção 0), ou redefinido nas subclasses, por exemplo o método para exibição da hiperbase pública e da base privada, conforme comentado na Seção 3.2.6.3 e apresentado em maiores detalhes no Capítulo 4 e em [Much96]. Entretanto, é possível determinar outro método de exibição da estrutura através de sua redefinição no descritor, ou mesmo reconfigurar o método default (por exemplo, exibir nós como quadrados ou círculos, mudar a cor de exibição de nós e elos, etc.).

A lista de operações é uma especialização da classe script e, como tal, contém uma seqüência ordenada de operações. Tal qual o objeto ponto de encontro, cada operação da seqüência é composta por um conjunto de condições e um conjunto de ações. A satisfação das condições implica no disparo das ações a elas associadas. As condições de um objeto lista de operações são semelhantes às 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, as condições avaliam valores booleanos e podem ser (binárias) simples ou compostas. Toda condição binária simples é expressa por duas condições: uma condição prévia, a ser satisfeita imediatamente antes do instante de tempo onde a condição é avaliada, e uma condição corrente, a ser satisfeita no instante de tempo onde a condição é avaliada. Uma condição 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 associada. Os operadores de comparação usados na avaliação das condições simples são:

- = comparação de igualdade
- ≠ comparação de desigualdade
- < comparação de inferioridade
- ≤ comparação de igualdade ou inferioridade
- > comparação de superioridade
- ≥ comparação de igualdade ou superioridade

As comparações são realizadas com respeito a:

1. Estados de um evento *E* do objeto de representação criado a partir do descritor.
2. A variável *ocorrido* associada a um evento *E*.
3. Atributos do nó de representação criado a partir do descritor.

Qualquer expressão lógica de condições baseada nos operadores ∧ (e), ∨ (ou), ¬ (negação) e ⊕ (retardo) define uma condição composta. Os operadores lógicos ∧, ∨ e ¬ possuem os significados usuais da lógica proposicional. Uma expressão E ⊕ (retardo) é satisfeita em um tempo *t* se a condição *E* estava satisfeita em *t-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 áudio, uma ação de mudança de comportamento poderia ser “coloque o volume a X dB”.

Uma *descrição de um evento*, em um descritor, por sua vez, consiste da tupla <*ida, tipo, duração, oper, estado*>. *Ida* 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 o tipo de evento associado à âncora (exibição, seleção) ou tem o valor especial ξ. *Duração* especifica a duração de exibição do evento, no caso de um evento de exibição. No caso de um evento de seleção a duração recebe o valor default

"infinitésimo", indicando que a duração é instantânea. *Oper* especifica um objeto lista de operações, que deve ser executado caso o evento associado à âncora ocorra e se o parâmetro estado permitir. *Estado* indica se a lista de operações pode ser executada ou não. Caso o parâmetro tipo receba o valor ξ, a lista de operações deve ser executada, se permitida pelo parâmetro estado e se o descritor receber uma mensagem de outro objeto especificando a âncora (no caso, mensagem de um elo, resultado da ação ativa de um ponto de encontro).

Cabe então aqui ressaltar que:

1. As mudanças de comportamento são programadas em listas de operações, como em [BuZe92], isto é, a descrição de evento indica mudanças comportamentais na exibição do objeto representação criado a partir do descritor.
2. As descrições de eventos podem ser inibidas ou desinibidas pela atribuição de valores ao parâmetro estado.
3. As ações “Habilita” e “Inibe” dos pontos de encontro de um elo (vide Seção 3.2.2) são usadas (através de troca de mensagens entre objetos, conforme abordado na Seção 3.2.7) para atribuir valores ao parâmetro estado de uma descrição de evento.
4. A ação “Ativa” dos pontos de encontro de um elo (vide Seção 3.2.2) é usada (através de troca de mensagens entre objetos, conforme abordado na Seção 3.2.7) para ativar a lista de operações de uma descrição de evento que tem ξ como valor para o parâmetro tipo.
5. Uma lista de operações pode ter restrições de sincronização temporal, como por exemplo, “a ação X deve ser executada daqui a 5 segundos”.

Voltando a duração de um evento de exibição, ela é especificada por uma tupla <*mínima*, *ótima*, *esperada*, *máxima*, *custo*>. Esta tupla indica ao Formatador Temporal o intervalo de tempo que uma exibição pode assumir, intervalo [mínimo, máximo], bem como a duração que levaria a uma melhor qualidade de exibição, dada pelo valor ótimo, e o custo correspondente a não se assumir a qualidade ótima. O valor da duração esperada inicialmente assume o mesmo valor da ótima. O Formatador Temporal pode, baseado em procedimentos de sincronização, comentados no Capítulo 5, alterar esse valor para aquele

que atenda a melhor qualidade de exibição do documento como um todo, otimizando o custo global calculado a partir do custo especificado para cada evento.

Um exemplo de especificação de duração pode ser dado para um evento de exibição de um nó de áudio. Sabe-se que, para um sinal de áudio — voz humana, 60 a 65% do sinal se constitui de silêncio. Sabe-se também que a eliminação de silêncio dentro de certos limites (até 50% entre surtos de voz) é tolerado pelo ser humano, bem como a inserção de silêncio. Assim, para um nó de áudio — correspondente à voz humana, com 10 segundos de duração — poderíamos estipular uma duração igual a <7,5 seg., 10 seg., 10seg., 11 seg., custo>, onde o custo daria, por exemplo, um fator de qualidade perdida a cada mseg acrescentado ou retirado do sinal.

3.2.5.1. Revisitando os Objetos de Representação

Conforme mencionado na Seção 3.2.4, um objeto de representação é criado a partir de um descritor e um nó associado. Mais precisamente, um objeto de representação é criado pelo executor de apresentações, ou formatador temporal NCM (o equivalente ao instanciador do Modelo Dexter), a partir de um *identificador de descritor*, de *procedimentos* e de um *identificador de nó*. Os objetos representação são responsáveis pela detecção e sinalização da ocorrência dos eventos.

O identificador do descritor pode ser definido em um atributo do nó a partir do qual o objeto de representação (uma instância do nó) é criado (Vide Seção 0). Os elos também podem possuir um conjunto de identificadores de descritores, como no Modelo Dexter, que contém informações para o modelo de apresentação indicando como os objetos representação dos nós referenciados devem ser criados (Vide Seção 3.2.2). Os nós de contexto também possuem, para cada um dos seus nós componentes, um atributo onde pode ser definido o identificador do descritor que deve ser usado na criação do objeto representação do componente (Vide Seção 0).

Quando um nó vai ser apresentado, o identificador de descritor definido explicitamente pelo usuário sobrepõe-se ao identificador de descritor definido pelo elo usado 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 usado se nenhuma das formas de identificação do descritor já apresentadas for definida.

Os procedimentos especificados na criação de um objeto de representação operam nos objetos terminais ou de composição e em seus eventos. Os *procedimentos de especificação de comportamento* fornecem as funções para implementar uma interface com o usuário usada para criar e editar eventos nos objetos representação. Exemplos desses procedimentos são aqueles encontrados no Editor e Exibidor de Sincronismo, assunto do Capítulo 4 e de [Cost96]. Os *procedimentos de análise de tempo* facilitam a criação automática de um escalonamento da apresentação do documento, provendo o sistema com informação de tempo sobre o elemento que está sendo exibido, como a duração do intervalo decorrido após a ocorrência do último evento. São procedimentos que atuarão como parte do Executor, assunto do Capítulo 5 e de [Rodr97]. Os *procedimentos de controle* permitem que o sistema controle o comportamento dos objetos — como, por exemplo, aumentando ou diminuindo a velocidade de exibição de um vídeo — durante a apresentação do documento. Normalmente, esses procedimentos fazem parte dos editores e exibidores especificados pelo descritor no atributo especificação de iniciação.

Cada tipo de mídia deve fornecer os procedimentos de controle que podem ser usados para afetar o comportamento da exibição dos objetos representação do tipo de mídia em questão. Todos os tipos de mídia devem fornecer procedimentos para iniciar, finalizar, suspender e retomar a exibição do objeto representação. Contudo, um tipo de mídia pode adicionalmente fornecer procedimentos de controle específicos, por exemplo, acelerar uma apresentação, etc.

3.2.6. Controle de Versões

Conforme já mencionado na Seção 3.1, o mecanismo de controle de versões, apresentado em [SoCR95], embora saliente a importância de considerar os vários objetos de representação, derivados a partir de um mesmo objeto de dados, como diferentes versões deste objeto de dados, não considera esse caso. Na verdade, o mecanismo descrito considera que pode haver apenas um objeto de representação instanciado correspondente ao

mesmo objeto de dados. O presente trabalho estende a descrição anterior de forma a atender os requisitos 13 e 14 lá especificados, quais sejam:

"R13. Representações Distintas da Mesma Informação — é importante que o sistema tenha como capturar a noção de que diferentes representações, em particular, em diferentes mídias, podem ser usadas para descrever uma mesma informação (como, por exemplo, a versão escrita e falada de uma palestra). Representações diferentes de uma mesma informação devem ser tratadas como diferentes versões desta informação.

R14. Uso Concorrente da Mesma Informação — cópias (temporárias) da mesma informação em uso em determinado instante devem ser consideradas versões desta informação. O acoplamento desta noção de versão com o mecanismo de notificação vai proporcionar um bom suporte a trabalhos cooperativos. Na verdade este caso é uma generalização do caso anterior."

Embora não fosse assunto específico desse trabalho, o controle de versões em documentos teve de ser esmiuçado, com a introdução de novos conceitos ao modelo, referentes aos nós de contexto. Estes novos conceitos são apresentados em detalhes no presente trabalho. Os conceitos que não sofreram modificações são apenas resumidos, com a finalidade de prover ao leitor uma visão completa do NCM, podendo uma discussão mais detalhada sobre o controle de versões no sistema HyperProp ser encontrada em [SoCR95].

No NCM, apenas os nós terminais e nós de contexto de usuário estão sujeitos a versionamento de dados (a ser definido nos próximos parágrafos), como salientado pelo fundo cinza na Figura 10. Além disso, cada atributo (incluindo o conteúdo) de um nó terminal ou contexto de usuário pode ser especificado como versionável ou não. Um atributo não versionável pode ter seu valor modificado sem a necessidade da criação de uma nova versão. Ao contrário, modificações no valor de um atributo versionável devem ser realizadas em uma nova versão do objeto, se este já estiver no estado permanente, como será detalhado.

A possibilidade de se adicionar novos atributos a um nó sem a criação de novas versões é bastante atraente. Suponha, por exemplo, que após um nó ser criado, uma nova ferramenta é introduzida no sistema. A ferramenta pode querer guardar algumas informações específicas nos nós. Em sintonia com esta possibilidade, no NCM o usuário pode especificar se a adição de novos atributos é permitida sem a criação de novas versões do objeto.

Nós objetos de dados podem ser criados pelo versionamento de objetos de armazenamento ou de outros objetos de dados e são chamados *versões de dados*. Nós objetos de representação são criados pela associação de um descritor a um objeto de dados e são chamadas *versões de representação*, para diferenciar do primeiro caso. Versões de representação darão suporte ao atendimento dos requisitos R13 e R14.

Versionamento de elos não foi incluído no modelo, uma vez que acredita-se que esta facilidade adiciona mais complexidade do que funcionalidade ao sistema e, além disso, se necessário, pode ser modelada pelo versionamento dos nós de contexto de usuário. Resta porém avaliar se esta extensão torna-se relevante quando elos carregam ações e condições complexas definidas em seus pontos de encontro, que não era o caso da especificação encontrada em [SoCR95].

Finalmente, cabe ressaltar que os mecanismos de suporte a versão introduzidos no NCM podem ser configurados pelas aplicações. Versões podem ser criadas automaticamente ou criadas por comandos explícitos, o que pode ser usado pelas aplicações para implementação de várias políticas de versionamento.

3.2.6.1. Estados de um Nó

NCM inclui a noção de estado de um nó terminal ou um nó de contexto de usuário para facilitar o controle de consistência entre nós, o suporte ao trabalho cooperativo e a criação automática de versões. Esta noção é válida apenas para os nós que contêm atributos versionáveis. O estado é somente um novo atributo dos nós, cujo valor afeta e é afetado pela execução de certas operações. Todo nó deve possuir um método, *altera estado*, que permite alterar seu estado, de forma consistente.

Um nó terminal ou um nó de contexto de usuário *N* pode estar em um dos seguintes estados: *permanente*, *temporário* e *obsoleto*. Um nó é criado no estado temporário e permanece neste estado enquanto está sendo modificado. Quando se torna estável, o nó pode ser promovido ao estado permanente explicitamente através de uma operação do usuário ou implicitamente como efeito colateral de certas operações do modelo. Como um exemplo de mudança de estado implícita, tem-se que um nó temporário pode se tornar permanente quando uma primitiva para criação de versão é a ele aplicada. Um nó permanente não pode ser diretamente removido ou alterado, mas o usuário pode torná-lo obsoleto, permitindo que outros nós que o referenciem, ou dele derivados, sejam notificados. Nós obsoletos são mantidos apenas enquanto são referenciados por um elo ou são contidos em outro nó.

O conceito de estado de um nó é relevante apenas para nós terminais ou de contexto de usuário que têm atributos versionáveis. Assim, quando se diz, por exemplo, que nós no estado permanente não podem ser modificados, quer-se dizer que os atributos versionáveis não podem ser modificados. Cabe salientar também que, como mencionado anteriormente, a adição de novos atributos em um nó no estado permanente pode ser realizada sem, necessariamente, a criação de uma nova versão do nó.

Mais precisamente, um nó terminal ou de contexto de usuário no estado permanente, chamado de um *nó permanente*, possui as seguintes características:

- Não pode ter seus atributos versionáveis modificados (o que significa que os atributos explicitamente definidos não podem ser modificados, bem como as consultas que o definem, se o nó é virtual).
- Só pode conter nós permanentes ou obsoletos, se for um nó de contexto de usuário.
- Pode ser usado para derivar outros nós (isto é, versões, incluindo versões de dados e de representação), se for nó objeto de dados ou de armazenamento.
- Não pode ser diretamente removido.
- Pode se tornar obsoleto, mas não pode se tornar temporário novamente.

Um nó terminal ou de contexto de usuário no estado temporário, chamado de um *nó temporário*, possui as seguintes características:

- Todos os seus atributos podem ser modificados.
- Pode conter nós em qualquer estado, se for um nó de contexto de usuário.
- Não pode ser usado para derivar outros nós objetos de dados (isto é, versões de dados).
- Pode ser usado para derivar outros nós de representação (isto é, versões de representação), se for nó objetos de dados.
- Pode ser diretamente removido.
- Pode se tornar permanente, mas não pode se tornar obsoleto.

Um nó terminal ou de contexto de usuário no estado obsoleto, chamado de um *nó obsoleto*, possui as seguintes características:

- Não pode ter nenhum de seus atributos modificados (o que significa que os atributos explicitamente definidos não podem ser modificados, bem como as consultas que o definem, se o nó é virtual).
- Só pode conter nós permanentes ou obsoletos, se for um nó de contexto de usuário.
- Não pode ser usado para derivar outros nós (isto é, versões).
- É automaticamente removido pelo sistema, através de um processo de coleta de lixo, quando não é mais necessário (por exemplo, quando não é mais referenciado nem está incluído em nenhum outro nó).
- Não pode mais mudar de estado.

Um nó objeto de armazenamento só pode estar nos estados permanente ou obsoleto, consistente com o definido na Seção 3.2.4, onde objetos da camada de armazenamento estão sempre em sua forma final, isto é, não podem ter seus atributos versionáveis alterados.

Um nó objeto de dados, a princípio, poderia ser definido como podendo estar apenas nos estados permanente ou obsoleto, tal qual os objetos de armazenamento. Se assim fosse

feito, versões de uma forma geral, e não apenas versões de dados, só poderiam ser derivadas de nós permanentes. Embora não explicitado na versão anterior do NCM, esta suposição foi assumida. No entanto, ao se permitir várias versões de representação diferentes de um mesmo objeto de dados, manter essa suposição resultaria em uma explosão muito grande do número de versões propagadas. Assim, um nó objeto de dados pode também estar no estado temporário e dele ser derivadas versões de representação, como se verá nas operações de versões definidas na Seção 3.2.6.3.

3.2.6.2. Contexto de Versão

O nó contexto de versão foi definido com o intuito de equacionar o problema de manutenção da história de um documento. Um *nó contexto de versão V* é um nó de contexto que contém apenas ou nós terminais ou nós contexto do usuário. Todos os nós de um contexto de versão são objetos de armazenamento ou objetos de dados. Intuitivamente, *V* agrupa nós que representam versões de dados da mesma entidade, em algum nível de abstração, sem necessariamente implicar que uma versão foi derivada de outra. Os nós de *V* são chamados de versões correlatas e não precisam pertencer à mesma classe. Um elo da forma (<v_1, ϕ, $i_{1,}$ ϕ>, <v_2, ϕ, $i_{2,}$ ϕ>) em *V* indica que v_2 é *derivada* de v_1. As âncoras de um elo em *V* servem apenas para indicar com mais precisão quais partes de v_1 geraram quais partes de v_2. Os elos de *V* não estão sujeitos a restrições, exceto que o grafo induzido por eles deve ser acíclico. Note que versões de representação não são incluídas nos contextos de versão.

Um usuário poderá manualmente adicionar novos nós a um contexto de versões *V*, para indicar explicitamente que tais nós são versões do mesmo objeto, e novos elos, para indicar como as versões foram derivadas. Ele também poderá alterar *V* através de operações específicas de versionamento. A criação de elos de derivação causam, automaticamente, a mudança do nó origem do elo para o estado permanente, para a preservação da consistência.

Suponha uma perspectiva *P*, só contendo objetos de dados e de armazenamento, para um objeto de dados *D*. Para atender os requisitos R13 e R14 mencionados anteriormente, quando mais de um objeto de representação R_1,.., R_i,.. R_n for derivado de *D*, versão de um

objeto *A*, apenas *D* é adicionado ao contexto de versões, como derivado de *A*. Se um objeto de representação R_i qualquer passar do estado temporário para permanente, diversas alternativas podem ocorrer, dependendo se R_i é o último nó de representação que passa para o estado permanente ou não, se foi alterado ou não, e se *D* está no estado permanente ou temporário, a saber:

1. R_i sofreu alterações e R_i não é o último nó de representação que passa para o estado permanente. Neste caso é criado um novo objeto de dados D_i correspondente a R_i. D_i estará no estado permanente e será incluído no contexto de versões correspondente como derivado de *A*. D_i será incluído no mesmo OD-contexto *C* que contém *D* em *P*. Os elos visíveis por *D* em *P* que tinham *D* como nó âncora de destino e o descritor usado para criar R_i, devem agora ser apropriadamente direcionados para D_i. Os demais elos visíveis por *D* em *P* devem ser duplicados e ter, na cópia, *D* substituído por D_i em seus pontos terminais. Uma notificação deve ser enviada a todos os outros nós de representação.
2. R_i sofreu alterações, *D* está no estado temporário e R_i é o último nó de representação que passa para o estado permanente. Neste caso *D* recebe as mesmas modificações do conteúdo de R_i e é tornado permanente.
3. R_i sofreu alterações, *D* está no estado permanente e R_i é o último nó de representação que passa para o estado permanente. Também neste caso é criado um novo objeto de dados D_i correspondente a R_i. D_i estará no estado permanente e será incluído no contexto de versões correspondente como derivado de *A*. D_i será incluído no mesmo OD-contexto *C* que contém *D* em *P*. Os elos visíveis por *D* em *P* que tinham *D* como nó âncora de destino e o descritor usado para criar R_i, devem agora ser apropriadamente direcionados para D_i. Os demais elos visíveis por *D* em *P* devem ser duplicados e ter, na cópia, *D* substituído por D_i em seus pontos terminais. Uma notificação deve ser enviada a todos os outros nós de representação.
4. R_i não sofreu alterações e *D* está no estado temporário. Neste caso *D* simplesmente é tornado permanente.

5. R_i não sofreu alterações e *D* está no estado permanente. Neste caso nada mais é realizado.

Uma aplicação poderá definir, de várias formas, um nó em um contexto de versões *V* como a sua *versão corrente*, escolhida de acordo com um critério específico. No caso mais simples, a aplicação poderá reservar uma âncora de *V* para manter uma referência à sua versão corrente *V*. Outras âncoras podem especificar outras versões de acordo com outros critérios de seleção. No modelo com entidades virtuais, a aplicação poderá se valer de consultas para localizar dinamicamante a sua versão corrente. A consulta poderá ser armazenada em uma âncora ou definida em um elo fora do contexto de versões, ou mesmo especificada através de mecanismos mais gerais, conforme discutido na Seção 3.2.6.3. Note que tal consulta poderá em certos casos retornar mais de uma versão (como por exemplo, "retorne todas as versões criadas por João"), que deverão ser interpretadas como alternativas e apresentadas dentro de um nó de contexto de usuário.

No NCM, um ponto terminal de um elo foi definido como $<(N_k,..., N_2, N_1), A, \alpha, tipo>$. No caso de N_1 ser um nó de contexto de versões, diz-se que N_2 contém, não N_1, mas sim, o nó especificado por *A*.

3.2.6.3. Hiperbase Pública e Base Privada

Define-se *hiperbase* como um conjunto de nós tal que se um nó de composição *C* pertence à hiperbase, então todos os nós contidos em *C* devem também pertencer à hiperbase.

Entende-se por autoria cooperativa o processo de criar ou modificar uma hiperbase, ou um subconjunto da hiperbase, por um grupo de usuários. De forma geral, um ambiente de autoria cooperativa deve permitir que usuários compartilhem informação, oferecer alguma forma de criar informações privadas, e permitir a fragmentação do espaço global de trabalho em frações menores para reduzir o espaço de navegação.

A noção de nó de contexto oferece um ponto de partida para organizar a autoria cooperativa. Considere o conjunto de todos os nós de contexto de usuário e nós terminais. Considere agora uma partição desse conjunto. Um e apenas um dos subconjuntos da

partição forma a *hiperbase pública*, denotada por H_B, que corresponderá à informação pública e estável. Os outros subconjuntos formam as bases privadas, usadas na modelagem da interação do usuário com o documento, de acordo com o paradigma de sessão de trabalho proposto pelo Modelo Dexter. Uma base privada pode conter outras bases privadas, o que permite organizar a sessão de trabalho em várias subsessões aninhadas. Note que uma versão específica de um nó terminal ou de contexto de usuário pode pertencer a apenas uma destas bases, incluindo-se aqui a hiperbase pública.

Mais precisamente, uma *hiperbase pública* é um nó de contexto que agrupa nós terminais e nós de contexto de usuário. Todos os nós em H_B devem ser nós objetos de armazenamento, isto é, estão no estado permanente ou obsoleto e, como em toda hiperbase, se um nó de contexto de usuário *C* está em H_B, então todos os nós contidos em *C* devem também estar contidos em H_B. O atributo conjunto de âncoras de uma hiperbase pública é o conjunto vazio.

Uma *base privada* é também uma especialização do nó de contexto, que agrupa qualquer entidade, exceto nós de contexto de versões e a hiperbase pública, tal que:

1. Uma base privada pode pertencer a no máximo uma base privada.
2. Se um elo está contido em uma base privada, o nó base de seu ponto terminal de origem deve ser um nó de anotação.

A classe base privada é especializada em OD-base privada e OR-base privada. Uma OD-base privada é um objeto de dados tal que:

1. Só contém objetos de dados.
2. Se um nó de composição *N* está contido na OD-base privada *PB*, seus componentes ou estão contidos em *PB*, ou na hiperbase pública, ou em qualquer base privada de um aninhamento de bases privadas contido em *PB*.

Uma OR-base privada é um objeto de representação tal que:

1. Só contém objetos de representação.

2. Se um nó de composição *N* está contido na OR-base privada *PB*, seus componentes ou estão recursivamente contidos em *PB*, ou recursivamente contidos na OD-base privada de onde *PB* se originou, ou na hiperbase pública.

Intuitivamente, uma OR-base privada agrupa todas as entidades usadas pelo usuário durante uma sessão de trabalho.

Várias foram as operações para controle de versões de uma base privada definidas em [SoCR95]. A introdução do conceito que nós de representação são versões de nós objetos de dados e que nós objetos de dados podem ser associados a diferentes descritores para formar nós de representação diferentes, traz uma série de modificações às operações anteriormente definidas, explicitadas a seguir.

Um usuário pode mover um nó terminal ou de contexto de usuário de uma OR-base privada para uma OD-base privada ou de uma OD-base privada para a hiperbase pública através da primitiva de *check-out*, desde que o nó seja permanente. Se um nó permanente de contexto de usuário *C* for movido, então todos os nós contidos em *C* devem também ser movidos para a mesma base.

No caso do check-out de um nó objeto de representação *R*, ele é simplesmente tornado permanente e depois (tornado obsoleto e) destruído (a mudança de base de fato não ocorre). O nó objeto de dados correspondente *D* é inserido nas composições onde estava o nó de representação, se lá não estiver ainda inserido. Os elos da OR-base privada devem ser apropriadamente modificados para conterem *D* em substituição a *R*.

Quando um objeto de dados sofre check-out, todos os objetos de representação dele derivados devem sofrer check-out. No caso de check-out de um nó objeto de dados, realmente ocorre uma mudança de bases, isto é, o nó objeto de dados migra para a hiperbase pública tornando-se um nó objeto de armazenamento. Note que a movimentação de uma versão para a hiperbase pública só precisa ser realizada quando ocorrer alguma modificação no conteúdo original. Se após a criação de uma versão de dados *D* de um nó *A* da hiperbase pública em uma OD-base privada, *D* não sofrer qualquer modificação, *D* é simplesmente destruída (ou seja, tornada obsoleta e depois destruída) ao ser movida para a

hiperbase pública, pois não há a necessidade de replicação de nós. Os nós de composição que contêm *D*, tanto na OD-base privada quanto na OR-base privada, bem como os elos, devem ser atualizados para conterem agora *A*, ao invés de *D*. Da mesma forma, todas as versões criadas a partir de *D* devem ser transformadas em versões de *A* no contexto de versões apropriado.

Todos os nós contextos de usuário e nós terminais de uma OR-base privada *PB* podem ser movidos em bloco para uma OD-base privada através da primitiva *mova*. Neste caso diz-se que a OR-base privada foi movida para a OD-base privada. Quando a primitiva é aplicada, todos os nós terminais e de contexto do usuário da base privada vão para o estado permanente e são transferidos para a OD-base privada (através da primitiva check-out); e todas as bases privadas contidas em *PB* são recursivamente movidas para a OD-base privada. No final do processo, a OR-base privada *PB* conterá apenas trilhas, anotações (e elos associados) e bases privadas, que conterão apenas trilhas, anotações e bases privadas, recursivamente.

Analogamente, todos os nós contextos de usuário e nós terminais de uma OD-base privada *PB* podem ser movidos em bloco para a hiperbase pública através da primitiva *mova*. Neste caso, diz-se que a OD-base privada foi movida para a hiperbase pública. Quando a primitiva é aplicada, todos os nós terminais e de contexto do usuário da base privada vão para o estado permanente e são transferidos para a hiperbase pública (através da primitiva check-out); e todas as bases privadas contidas em *PB* são recursivamente movidas para a hiperbase pública. No final do processo, a OD-base privada *PB* conterá apenas trilhas, anotações (e elos associados) e bases privadas, que conterão apenas trilhas, anotações e bases privadas, recursivamente.

Um nó na hiperbase pública não pode migrar desta hiperbase para uma OD-base privada, embora possam ser criadas versões de dados destes nós nas diversas bases privadas. No HyperProp, manipular um documento implica na criação de novas versões, na OD-base privada corrente, de todos os nós terminais e de contexto de usuário visitados. Estas novas versões podem ser derivadas de um nó permanente, ou corresponder à criação de uma informação completamente nova. Analogamente, um nó em uma OD-base privada não pode migrar desta base para uma OR-base privada, embora possam ser criadas versões de

representação destes nós na OR-base privada. Estas versões correspondem às instanciações do Modelo Dexter.

A primitiva *Check-in (Perspectiva, Descritor)* permite a criação de uma nova versão de representação de um nó objeto de dados terminal ou de contexto de usuário *N*, onde *N* é o nó base da perspectiva. A versão, incluída na OR-base privada *PB*, é criada pela associação do descritor especificado ao nó *N*. Ao criar uma versão de representação *R* do nó objeto de dados *N* em *PB* — check-in (P,D) — se a cardinalidade da perspectiva for maior que 2, isto é, se *N* na perspectiva *P* estiver contido em um contexto de representação *C* diferente de *PB*, então:

1. Se não existirem outras referências ao nó *N*, com outros descritores (referências por elo ou composição), especificadas nos vários nós que compõem a perspectiva *P*. *R* é incluído em *C* e *N* é removido de *C*. Todos os elos visíveis em *P* por *N*, se existirem, são alterados de forma a refletir apropriadamente, nos seus pontos terminais, o novo nó *R*, em substituição a *N*.
2. Se existirem outras referências ao nó *N* , com outros descritores (referências por elo ou composição), especificadas nos vários nós que compõem a perspectiva *P*. *N* não é removido de *C*. *R* é incluído em *C* e todos os elos visíveis em *P* por *N* que têm *N* como âncora de destino e *D* como descritor devem agora ser apropriadamente direcionados para *R*. Todos os demais elos visíveis por *N* em *P* devem ser duplicados e ter, na cópia, *N* substituído por *R* em seus pontos terminais.

Existem duas primitivas, *open* e *check-in*, disponíveis para a criação de uma nova versão de dados temporária de um nó terminal ou de contexto de usuário *N* em uma OD-base privada *PB*. Estas primitivas diferem no tratamento de nós de contexto de usuário, da forma descrita a seguir.

Suponha que *N* seja um nó de contexto de usuário. A primitiva *Check-in (Perspectiva)* permite a criação de uma nova versão de dados de *N*, onde *N* é o nó base da perspectiva. A versão temporária *V* criada é exatamente idêntica a *N*, ou seja, com os mesmos nós e elos, e deve ser incluída na OD-base privada *PB*. Ao criar uma versão de dados *V* do nó *N* em *PB* — Check-in (P) — se cardinalidade da perspectiva for maior que 2, isto é, se *N* na

perspectiva *P* estiver contido em um contexto de representação *C* diferente de *PB*, então *V* é incluído em *C* e *N* é removido de *C*. Todos os elos visíveis em *P* por *N*, se existirem, são alterados de forma a refletir apropriadamente, nos seus pontos terminais, o novo nó *V*, em substituição a *N*.

Já a primitiva *Open(Perspectiva)* cria na OD-base privada *PB* uma versão de dados temporária *V* de *N*, onde *N* é o nó base da perspectiva, e, recursivamente, de cada componente de *N*. *V* conterá as novas versões de dados dos componentes de *N*, e seus elos serão criados de forma a refletir apropriadamente os elos de *N*. Se um componente permanente recursivamente contido em *N* pertencer a mais de um contexto, apenas uma versão de dados temporária será criada para este nó. Ao criar uma versão de dados *V* do nó *N* em *PB* — Open (P) — se cardinalidade da perspectiva for maior que 2, isto é, se *N* na perspectiva *P* estiver contido em um contexto de representação *C* diferente de *PB*, então *V* é incluído em *C* e *N* é removido de *C*. Todos os elos visíveis em *P* por *N*, se existirem, são alterados de forma a refletir apropriadamente, nos seus pontos terminais, o novo nó *V*, e todas as versões de dados recursivamente contidas em *V* criadas, em substituição aos nós dos quais as versões de dados foram derivadas.

Algumas consequências interessantes decorrem do comportamento diferente das primitivas *open* e *check-in* para a criação de versões de dados. De fato, seja *N* um nó da hiperbase pública e *M* um nó contido recursivamente em *N* através de duas perspectivas. Suponha inicialmente que a versão de dados *N'* tenha sido criada de *N* pela primitiva *check-in*. Observe que, pela definição de *check-in*, temos que *N* e, consequentemente, *M* são necessariamente permanentes e que *N'* conterá recursivamente *M* exatamente como *N* contém. Ao atualizar *M* através de uma das perspectivas, o usuário criará então uma versão de dados de *M*, que não será visível pela outra perspectiva. Portanto, ao tentar atualizar *M* pela outra perspectiva, o usuário criará uma segunda versão de dados. Suponha agora que *N'* tenha sido criada através da primitiva *open*. Neste caso, *N'* já conterá uma (única) versão de dados *M'* de *M*, temporária, que poderá ser atualizada por quaisquer das duas perspectivas.

Uma implementação de *open* poderá postergar a criação das versões dos componentes de *N* para o momento em que de fato eles forem atualizados através da versão de dados *N'*.

Versões permanentes em uma OD-base privada *PB* podem ser usadas para derivar novas versões de dados, ou serem incluídas em um nó de contexto de usuário, na própria base *PB*, na base privada que contém *PB*, e em todas as bases privadas que contêm estas bases, e assim recursivamente. Isto parece bastante razoável, uma vez que estes nós representam um estado de trabalho consistente sobre o ponto de vista da tarefa que está sendo realizada em alguma OD-base privada. Versões temporárias, por outro lado, são apenas acessíveis para a manipulação na OD-base privada onde está contida, ou para criação de versões de representação.

Pode-se agora definir operações de versionamento compostas, de acordo com a exigência de uma particular aplicação. Como exemplo, pode-se citar as operações de Check-in (P,D) e Open (P,D) para criação de uma versão de representação de um nó de armazenamento *N* em uma OR-base privada, pela associação com um descritor *D*. Estas operações podem ser definidas como se segue.

Seja *N* um nó objeto de armazenamento terminal ou de contexto de usuário, onde *N* é o nó base da perspectiva *P* em uma OR-base privada *PB*. Seja *P'*, a perspectiva correspondente a *P* na OD-base privada correspondente a *PB*, contendo *N* como nó base e substituindo todos os nós de representação *P* pelos nós de dados de onde foram derivados. A primitiva *Check-in (Perspectiva, Descritor)* — Check-in (P,D) — permite a criação de uma nova versão de representação de *N*, primeiramente realizando a operação Ckeck-in (P'), na OD-base privada de onde a OR-base privada é derivada. Analogamente, a primitiva *Open (Perspectiva, Descritor)* — Open (P,D) — permite a criação de uma nova versão de *N*, primeiramente realizando a operação Open (P'), na OD-base privada de onde a OR-base privada é derivada. Em qualquer dos dois casos, uma nova versão de dados *N*' de *N* é criada na OD-base privada. *N* é então substituído por *N'* na perspectiva *P*, gerando uma nova perspectiva *P''*, na OR-base privada. Todos os elos visíveis por *N* em *P* são atualizados substituindo-se *N* por *N*'. Após é então realizada a operação Check-in (P", D).

As figuras que seguem ilustram o uso das primitivas check-out, check-in e open em uma sessão onde um nó de composição *Z* (objeto de armazenamento) e seus componentes são manipulados.

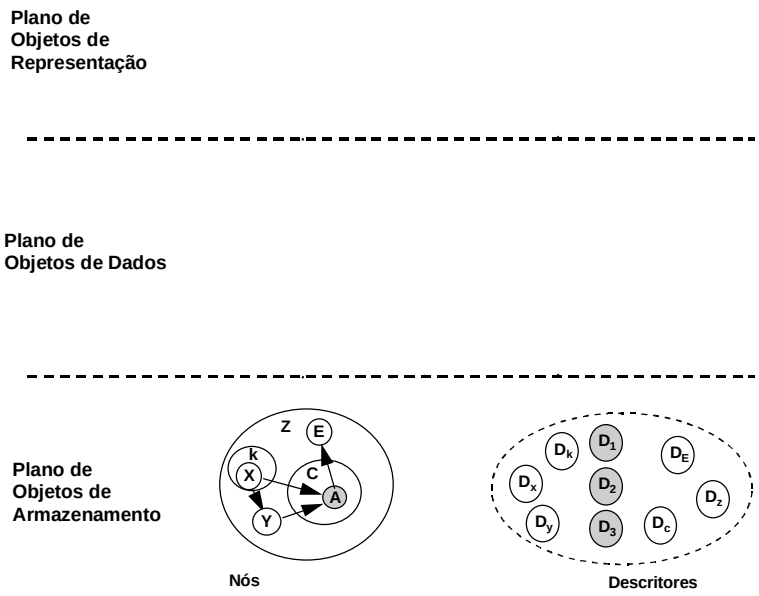


Figura 14 - Objeto de armazenamento nó de composição Z

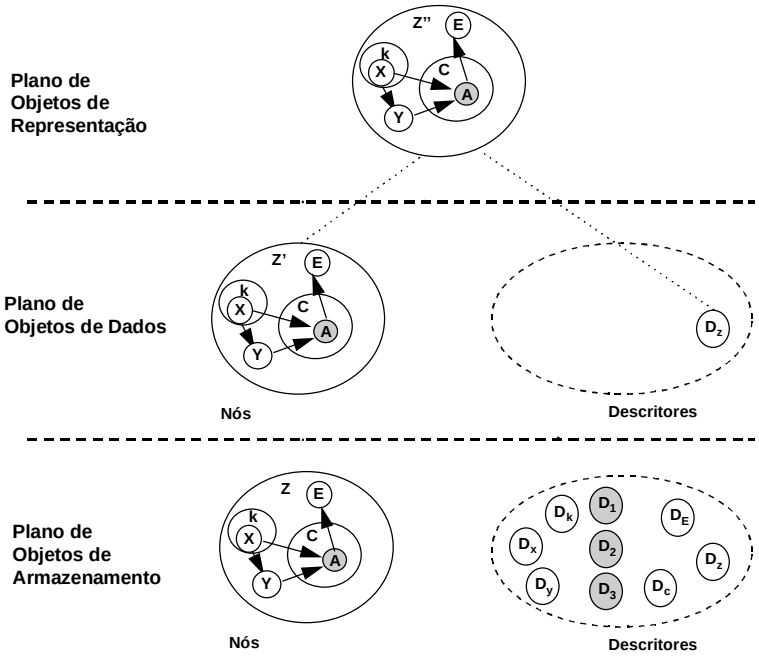


Figura 15 - Resultado da aplicação da primitiva Check-in(R-PB, Z , D_Z)

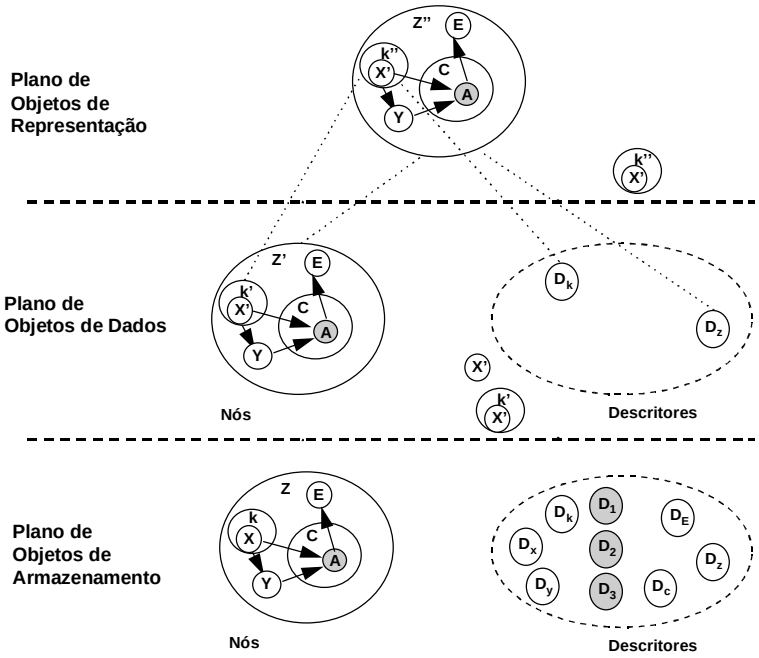


Figura 16 - Resultado da aplicação da primitiva Open(R-PB, Z'',K) , D_K)

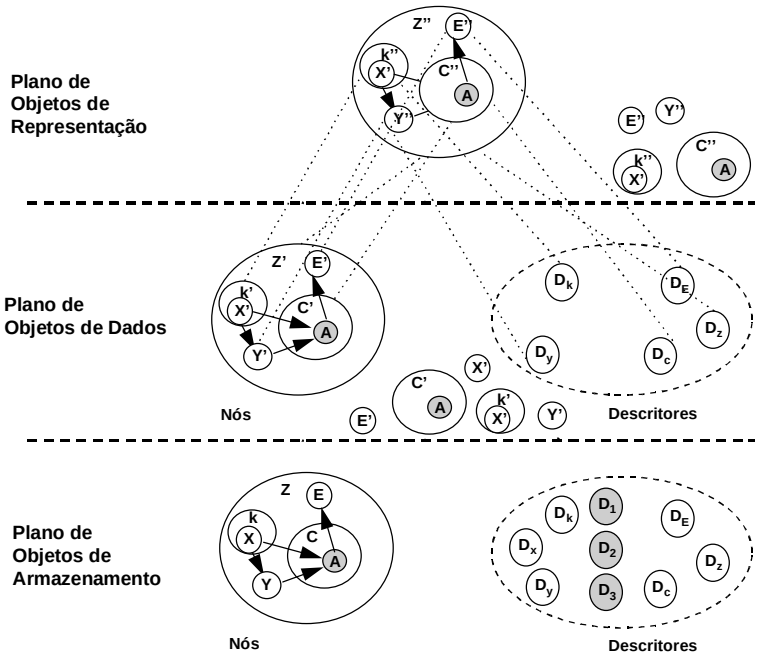


Figura 17 - Resultado da aplicação das primitivas Open(R-PB, Z'',E) , D_E), Open(R-PB, Z'',Y) , D_Y) e Check-in(R-PB, Z'',C) , D_C)

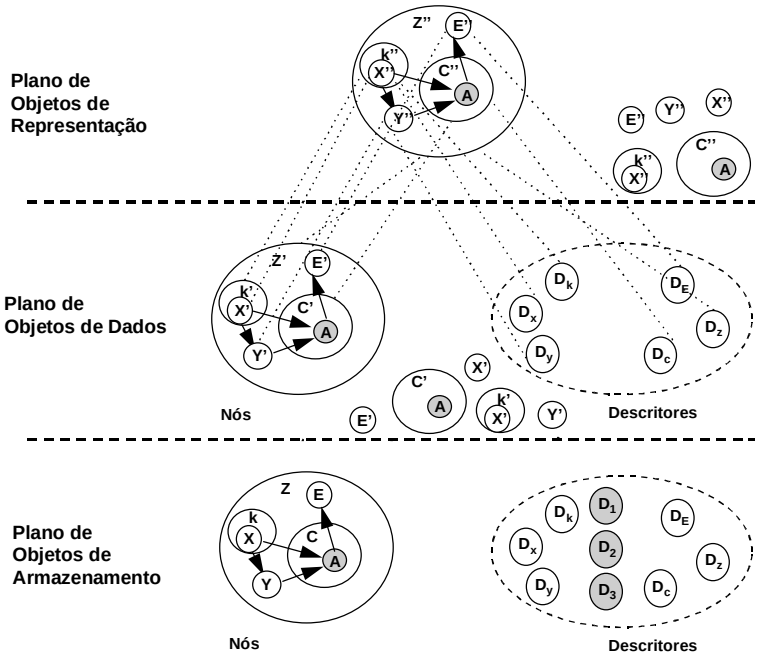


Figura 18 - Resultado da aplicação da primitiva Check-in(R-PB, Z'',K'',X') , D_X)

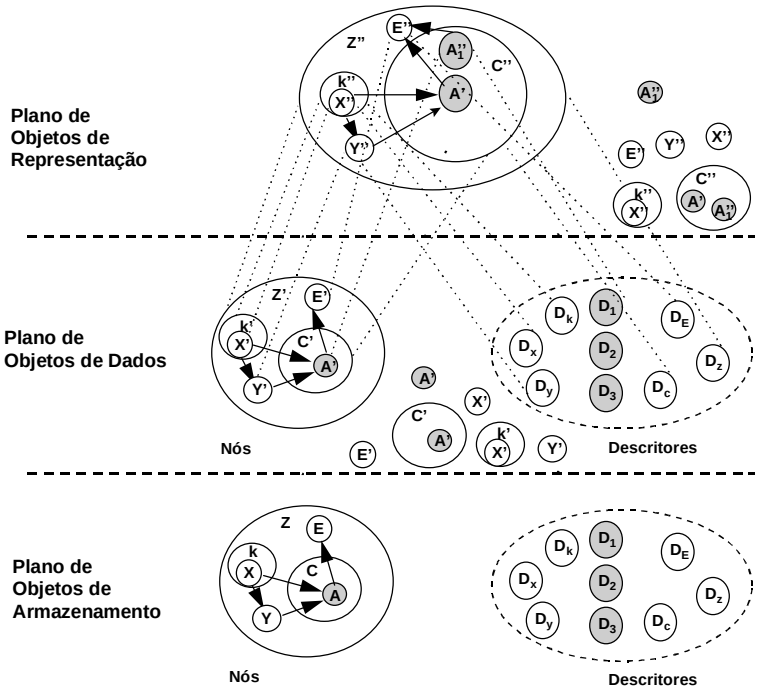


Figura 19 - Resultado da aplicação da primitiva Open(R-PB, Z'',C'',A) , D_1)

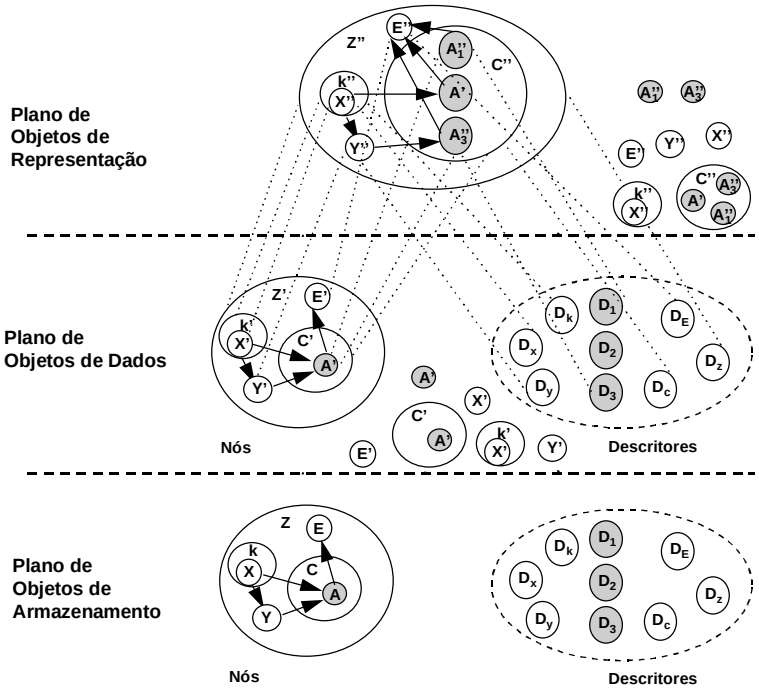


Figura 20 - Resultado da aplicação da primitiva Check-in(R-PB, Z'',C'',A') , D_3)

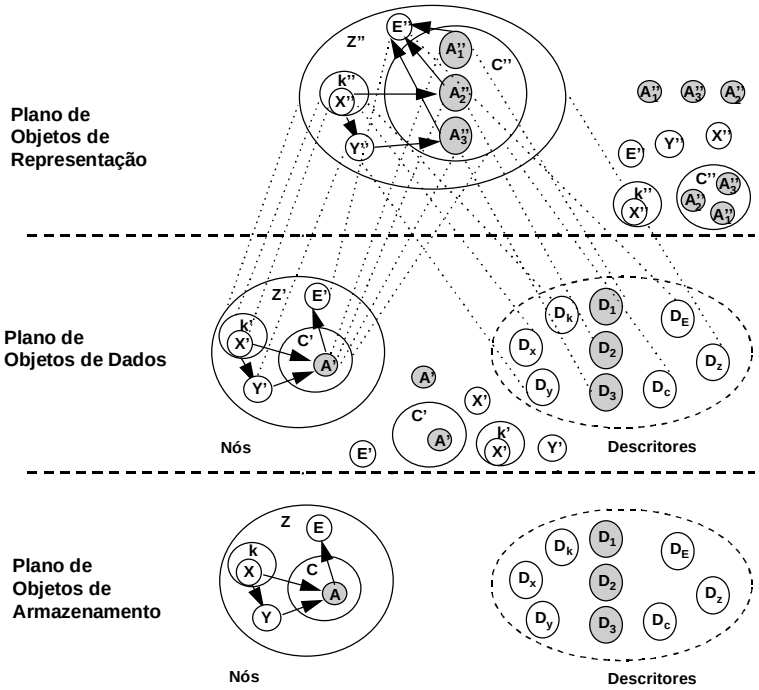


Figura 21 - Resultado da aplicação da primitiva Check-in(R-PB, Z'',C'',A') , D_2)

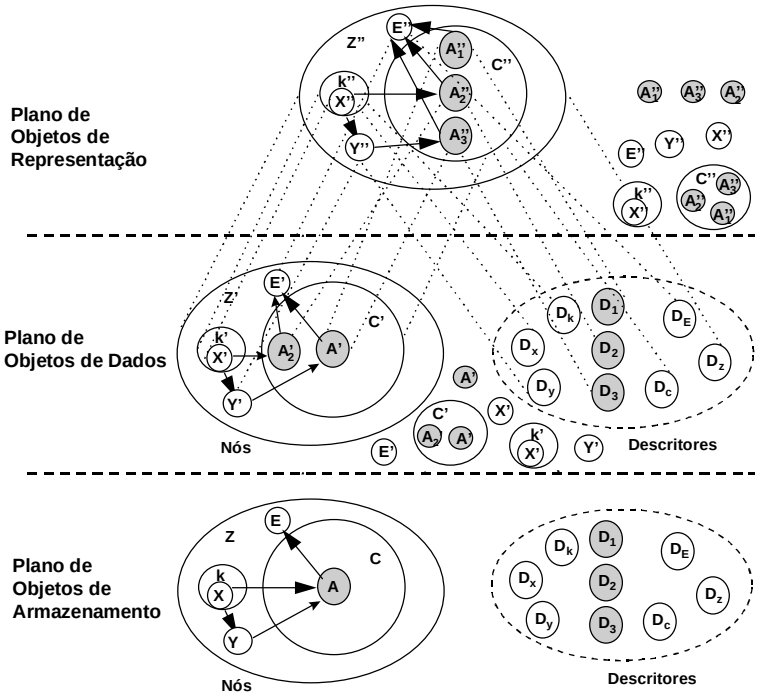


Figura 22 - A_2'' é tornado permanente depois de ter sido modificado.

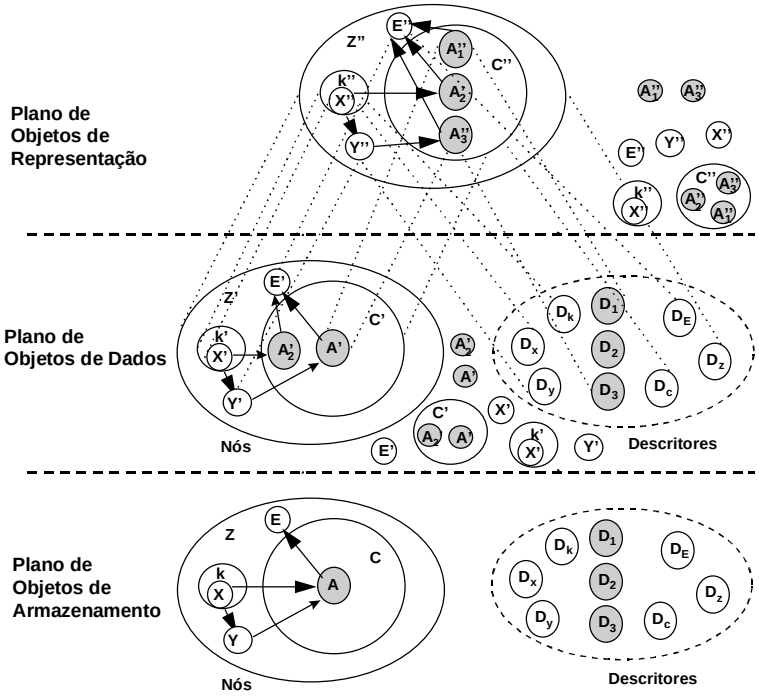


Figura 23 - Resultado da aplicação da primitiva Check-out(A_2'')

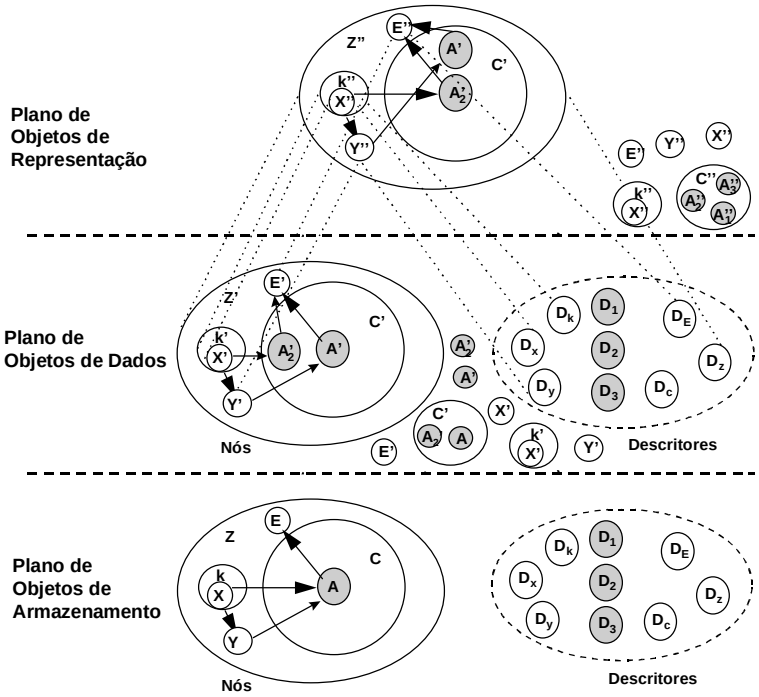


Figura 24 - C'' é tornado permanente e são aplicadas as primitivas Check-out(A_1'') e Check-out(A_3''). Os nós C'', A_1'' e A_3'' não foram modificados.

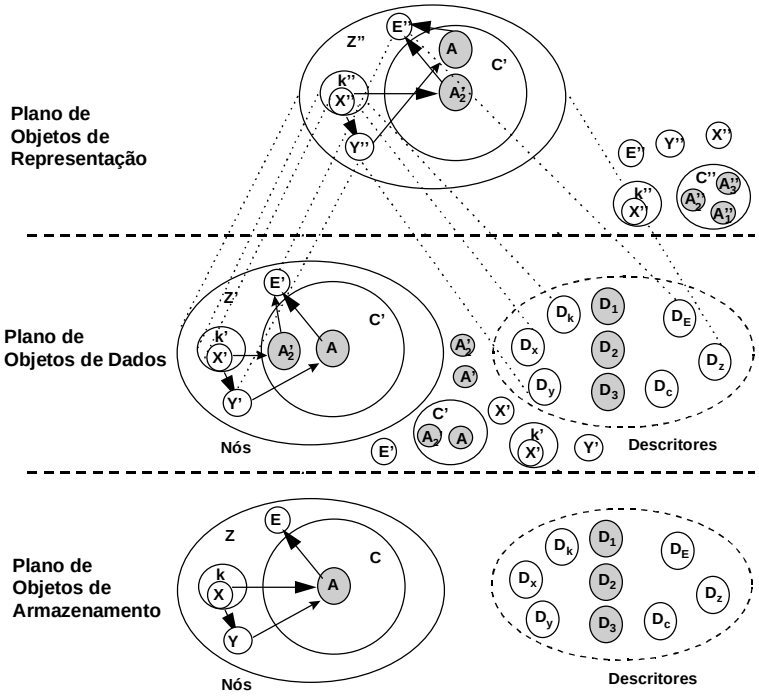


Figura 25 - Resultado da aplicação da primitiva Check-out(A')

Um usuário pode remover um nó *N* de uma base privada *PB* através da primitiva *delete*, conforme especificada para todo nó de composição. Se *N* for uma trilha, ou um nó de anotação (conceito a ser discutido na Seção 3.2.6.4), ele é simplesmente removido da base privada e destruído. Se *N* é um nó terminal ou de contexto de usuário, o resultado depende do estado do nó. Se *N* é temporário, ele é destruído e removido do contexto de versões apropriado; se *N* é permanente, ele é tornado obsoleto. Quando um nó se torna obsoleto em uma OD-base privada, ele é movido para a hiperbase pública. Se ele é um nó de contexto de usuário obsoleto, todos os seus nós componentes são transferidos também. Quando um nó é removido de uma OD-base privada, todos os objetos de representação dele derivados devem ser removidos das OR-bases privadas correspondentes.

Uma base privada *PB* também pode ser removida. Neste caso, todos os seus nós, incluindo bases privadas, são também removidos, recursivamente. A base privada *PB* é então destruída.

Na Seção 3.2.6.2, foi discutido como especificar versões correntes em nós de contexto de versões. A seleção de versão corrente baseada em uma linguagem de consulta é uma técnica poderosa, mas pode aumentar o *overhead* cognitivo do usuário, que tem de definir o critério de seleção para cada elo criado. Para minimizar este problema, no NCM cada nó contexto de versões tem duas âncoras especiais, definidas por consultas *default*, para a recuperação da versão corrente. Uma destas consultas *default* é definida no próprio contexto de versões. A outra é especificada, de uma forma mais geral, em um atributo da base privada. Quando um elo é criado, o nó de destino é examinado. Se o nó é um componente de um nó contexto de versões não especificado explicitamente pelo usuário (diretamente ou através de uma consulta), o elo é criado usando o critério de seleção especificado na base privada que contém o nó de contexto onde o elo está contido. Se a consulta não estiver especificada nesta base privada, o elo usará a consulta *default* definida no contexto de versões.

Qualquer dos mecanismos de navegação oferecidos pelo HyperProp pode ser usado para visitar um nó. Freqüentemente a navegação é baseada em uma consulta. Se cada vez que uma consulta for resolvida, por exemplo em uma navegação em profundidade, uma nova versão for criada na base privada, a sessão de trabalho rapidamente se encherá de versões

(considere o caso de um usuário navegando para baixo e para cima em uma perspectiva). Para evitar o problema, no NCM, uma vez que a consulta é resolvida, ela se torna permanentemente resolvida para aquela base (intuitivamente isto significa “para o resto da sessão de trabalho”).

Quando nós são movidos de uma OD-base privada para a hiperbase pública, o usuário pode escolher armazenar a configuração estática (com as consultas resolvidas) presente na base privada, ao invés da configuração dinâmica original, armazenada por *default*.

Na presença de nós contendo outros nós, quando o usuário cria uma nova versão de um dos componentes de um nó *N*, o sistema pode criar automaticamente uma nova versão de *N* de acordo com um critério preestabelecido. Denomina-se esse mecanismo de propagação automática de versões. Como um nó pode estar contido em muitas composições diferentes, a propagação automática de versões pode criar um número bastante grande de nós indesejáveis. Para evitar esse problema, a propagação automática deve ser limitada por alguma regra. A referência [SoCR95] apresenta o tratamento adotado no NCM.

Ainda com respeito aos métodos da base privada e da hiperbase pública, cabe aqui ressaltar vários pontos que cabem a esse trabalho definir e resolver.

Em primeiro lugar, os métodos de exibição da estrutura (ver definição em 3.0) de nós e elos da base privada e da hiperbase pública devem apresentar não apenas os nós e elos do primeiro aninhamento, mas de todos os níveis de aninhamento, para que o usuário tenha uma exata noção da estrutura do documento. Ora, a apresentação de toda essa estrutura pode trazer um *overhead cognitivo* ao usuário que pode se sentir completamente desorientado depois da apresentação de tal grafo composto. Assim, técnicas de filtragem devem ser empregadas de forma a apresentar ao usuário informações locais e globais do documento, conforme seu foco de interesse. Exibidores e editores para a estrutura da hiperbase pública e base privada, bem como das outras composições são objeto do Capítulo 4 e da referência [Much96].

Outro ponto importante diz respeito à definição de sincronização inter-nós de um documento. Editores e exibidores de estrutura são insuficientes para a definição e exibição

do sincronismo temporal e espacial existente entre objetos de um documento. Editores e exibidores de sincronismo assim se fazem necessários. Toda OR-base privada deve ter portanto um método editor e exibidor de sincronismo que facilite ao usuário a definição do sincronismo espacial e temporal através de elos e descritores. O ideal é que esses novos editores/exibidores trabalhem em conjunto com o editor e exibidor de estrutura para, novamente, evitar a desorientação do usuário. Tais editores e exibidores são também objeto de estudo do Capítulo 4 e das referências [Cost96] e [CMSS96].

3.2.6.4. Nó de Anotação

Anotações consistem de um comentário (em qualquer formato ou mídia) e mantêm referências às versões objeto do comentário, e às versões consideradas respostas ao comentário. Anotações são uma facilidade importante no desenvolvimento exploratório e cooperativo de um documento.

Mais precisamente, uma *anotação A* é um nó de contexto que agrupa um conjunto de elos, nós terminais, nós de contextos de usuários e trilhas. Intuitivamente, elos de uma base privada podem ligar os componentes de um nó de anotação a nós da mesma base privada, indicando nós que originaram o comentário e nós de resposta aos comentários. Anotações permitem que observações sejam feitas a um nó permanente, sem a criação de novas versões.

A classe anotação é especializada nas classes OD-anotação e OR-anotação. Um nó OD-anotação contém recursivamente apenas objetos de dados. Nós OD-terminais e OD-contexto de usuário de um nó OD-anotação não sofrem versionamento, isto é, têm todos os seus atributos definidos como não versionáveis. Além disso, esses nós não derivam por versionamento de nenhum nó de armazenamento. De forma análoga, um nó OR-anotação contém recursivamente apenas objetos de representação. Uma base privada pode conter nós de anotação, mas não pode conter nós terminais e de contexto do usuário recursivamente contidos na anotação. Estes nós não pertencem à definição de nenhuma base.

3.2.6.5. OR-Trilha, OR-Anotação e OR-Base Privada

A Seção 3.2.6 iniciou mencionando que, no NCM, apenas os nós terminais e nós de contexto de usuário estão sujeitos a versionamento de dados. No entanto, nós trilha, anotação e base privada podem sofrer operações análogas ao versionamento de representações. Nós objetos de representação — OR-trilha, OR-anotação e OR-base privada — são criados pela associação de um descritor ao objeto de dados OD-trilha, OD-anotação, OD-base privada, respectivamente. No entanto os novos nós formados não são tratados como versões. Assim, pode-se dizer que apenas os nós terminais e nós de contexto de usuário estão sujeitos a versionamento, de uma forma geral.

Um nó OR-base privada *ORPB* só pode ser derivado de uma OD-base privada *ODPB*, que contenha apenas OD-trilhas, OD-anotações e OD-bases privadas. O novo nó terá acrescido, novos atributos e métodos para a manipulação do atributo conteúdo da base privada. A OR-base privada conterá apenas nós de representação. Todos as OD-trilhas, OD-anotações e OD-base privadas da *ODPB* serão associados aos seus descritores formando as OR-trilhas, OR-anotações e OR-base privadas que estarão contidas na *ORPB*. A partir da criação, a relação dos nós contidos na *ORPB* e na *ODPB*, vai depender das operações de versionamento executadas, conforme mencionadas ao longo de toda a Seção 3.2.6.

Uma OR-trilha ou OR-anotação pode ser incluído sempre em uma OR-base privada. A inclusão implicará sempre na inclusão de um nó objeto de dados correspondente na OD-base privada.

Um nó OR-trilha (OR-anotação) é derivado de um nó OD-trilha (OD-anotação) adicionando novos atributos e métodos para a manipulação do atributo conteúdo do objeto. Estes novos métodos e atributos são derivados a partir de um objeto descritor.

Um nó OD-trilha (OD-anotação) pode ser derivado de um nó OR-trilha (OR-anotação) pela exclusão dos métodos para a manipulação do atributo conteúdo do objeto.

3.2.7. Elos, Eventos e Mensagens — Considerações de Implementação do Modelo

O funcionamento dos elos e eventos definidos no NCM pode ser implementado com base na troca de *mensagens*.

À medida que as âncoras de um objeto são exibidas ou selecionadas pelo usuário ou o valor dos atributos de nós sofrem modificações, os eventos associados são processados. O funcionamento dos elos no NCM pode ser implementado pelo envio de mensagens *"ocorreu"* de um nó âncora de um elo, identificado por um ponto terminal de origem, para o elo, quando da mudança de estado de eventos definidos no nó âncora. Avaliadas as regras do ponto de encontro de um elo, ele pode enviar mensagens para os nós âncoras identificados por seus pontos terminais de destino. A mensagem "*ocorreu*" é enviada à instância do nó de contexto *C*, onde o elo foi definido, que a passará ao elo *L* em questão. Ao receber mensagens *"ocorreu"* de suas extremidades de origem, *L* avalia as regras definidas em seu ponto de encontro. Quando o resultado da avaliação de uma regra é positivo são encaminhadas mensagens às instâncias dos nós âncoras de destino do elo *L*, definidas na regra satisfeita.

Uma mensagem do elo *L*, do tipo "inicia evento *E*", causa a exibição da instância do nó âncora *N*, especificado pelo elo *L*, a partir da região associada ao ponto terminal de destino. Se *N* ainda não tiver sido instanciado, *C* (contexto que contém o elo) deve solicitar sua instanciação. Cabe lembrar que um ponto terminal de destino pode definir um caminho de nós (N_k,..., N_2, N_1). Neste caso, todos os contextos que contêm recursivamente o nó âncora do evento e que ainda não tiverem sido instanciados, deverão ser instanciados neste momento.

A Figura 26 exemplifica o modo como funcionam as sincronizações através de elos. Na figura são mostrados três objetos de representação (RO_1, RO_2 *e* RO_C), o evento origem do elo (E_2 em RO_1), o evento destino (E_5 em RO_2) e o ponto de encontro (*PE*) que define, neste caso, que o evento destino seja iniciado (*Inicie Evento*) no exato momento que o evento origem passar do estado preparado para ocorrendo.

Quando a mudança de estado no evento origem E_2 é processado pela objeto representação RO_1, é enviada uma mensagem "*ocorreu*" ao objeto RO_C que contém o elo. No atributo ponto de encontro deste elo, uma regra define que deve ser encaminhada imediatamente uma mensagem ao destino do elo, fazendo com que RO_C envie a mensagem "Inicie Evento

Exibição" a RO_2. Ao receber a mensagem, RO_2 começa a exibir o conteúdo do objeto de dados a ele associado, a partir da região 2, especificada pela âncora de E_5.

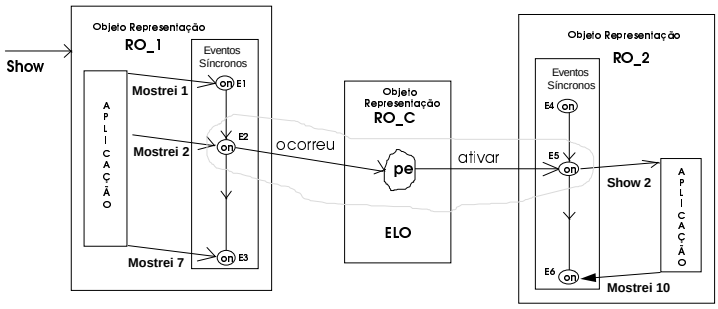


Figura 26 - Sincronização de Objetos

Como outro exemplo de elos de referência hipertextuais, tome a ilustração da Figura 27. Nessa figura, o objeto RO_1 entra em exibição em E_1. Ao ser processado E_2, pode-se, por exemplo, acionar um bip (mudança comportamental) avisando da existência de um elo que pode ser ativado pela seleção da região 2 associada aos eventos E_2 e E_4. A seleção desta região (evento E_4) provoca o envio de uma mensagem "*ocorreu*" destinada ao objeto RO_C. A partir deste ponto o procedimento segue idêntico ao já descrito no exemplo da Figura 26.

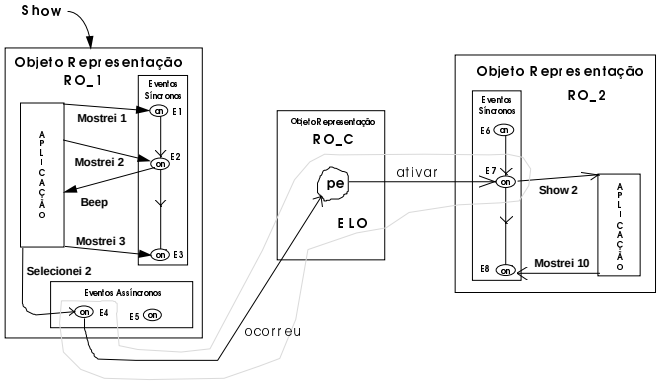


Figura 27 - Navegação por Seleção de Âncoras

A Figura 28 mostra como os elos podem ser usados no alinhamento temporal de eventos, com a finalidade de facilitar a tarefa de manter o sincronismo temporal da apresentação dos documentos dentro de padrões de qualidade aceitáveis. No alinhamento mostrado na figura, cada ponto de sincronização origem é seguido de um ponto de sincronização destino. Quando os eventos origem do elo são processados (mudança de estado de ocorrendo para

preparado em E_2 e E_6), além de enviarem uma mensagem "*ocorreu*" para o RO_C, eles comandam a suspensão da exibição dos dados controlada pelo objeto representação que os processou. A regra do ponto de encontro determina como condição a ocorrência do *E lógico* das duas condições determinadas por E_2 e E_6. Quando esta regra for satisfeita, RO_C envia mensagens "*Inicie Evento Exibição*" para os objetos de representação de destino, comandando o reinicio da exibição dos dados.

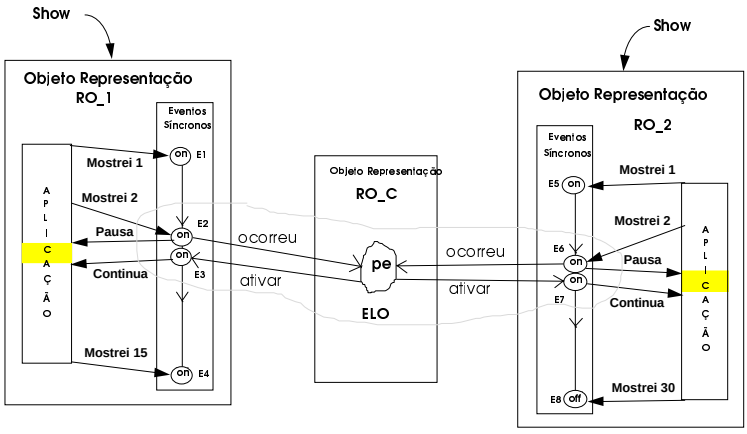


Figura 28 - Alinhamento Temporal de Eventos

3.3. Conclusão

A partir das operações básicas oferecidas pelo NCM é possível definir primitivas de navegação de mais alto nível específicas a uma aplicação. O sistema HyperProp oferece os seguintes mecanismos genéricos de navegação. *Navegação por elos* é a idéia central de sistemas hipermídia e é facilmente implementada conforme exemplificado na Seção 3.2.7. *Navegação por consulta* permite ao usuário selecionar um elo específico descrevendo as propriedades que esse nó deve satisfazer. *Navegação em profundidade* é a ação de percorrer , para cima e para baixo, a hierarquia de aninhamento dos nós de composição. *Navegação através de exibidores de estrutura* usa a visão pictorial oferecida para permitir a seleção de nós. Finalmente, a navegação por trilhas, facilmente implementada pelos métodos apresentados nos nós trilhas na Seção 0, permite ao usuário seguir trilhas estabelecidas em sessões prévias ou definidas como orientação de leitura no próprio documento.

Este capítulo apresentou extensões ao modelo NCM descrito em [SoCR95]. As principais contribuições acrescentadas foram:

- Definição mais precisa dos conceitos de conteúdo, região, unidades de informação, unidades de informação marcadas e identificação de âncoras.
- Especificação dos métodos e atributos para os objetos de composição, em geral.
- Especificação dos métodos e atributos para os objetos trilha.
- Especificação dos métodos para os diversos nós de contexto.
- Introdução dos conceitos eventos de exibição, de seleção e de atribuição.
- Especificação dos métodos de exibição e edição da estrutura e sincronismo dos nós e elos.
- Relacionamento dos exibidores e editores de estrutura e sincronismo com a arquitetura em camadas.
- Atualização da definição dos conceitos de perspectiva, elo, elo visível e entidade virtual.
- Inclusão de novas classes para definição do modelo de apresentação, entre elas: script, ponto de encontro, lista de operações e descritor.
- Especificação da classe descritor para definição do sincronismo espacial e temporal entre eventos.
- Redefinição da classe elo para especificação de relações n:m entre eventos.
- Definição das ações e condições de um elo e descritor.
- Inclusão de novas classes de elos: Sinc-elo, hiper-elo e elo genérico.
- Relacionamento das diversas novas classes incluídas com a arquitetura HyperProp.
- Definição da hierarquia de classes do NCM baseada nos objetos de armazenamento, de dados e de representação.
- Definição do controle de versões dos objetos de representação.
- Redefinição das operações de controle de versões, levando em conta o requisito de vários objetos de representação poderem ser derivados de um único objeto de dados.

- Introdução, através de exemplos, de um executor para a apresentação de documentos multimídia/hipermídia.

Capítulo 4

Edição e Exibição da Estrutura e Sincronismo de Documentos Multimídia/Hipermídia

Sistemas multimídia devem facilitar o trabalho de autoria oferecendo um ambiente completo para edição e manipulação de documentos. Editores gráficos para a definição da estrutura de documentos, definição das relações de sincronização entre seus componentes e especificação de mudanças no seu comportamento durante uma apresentação multimídia são cruciais para esses sistemas.

No NCM, o método *exibe estrutura* da classe composição é o responsável por ativar a interface gráfica que constrói o mapa (uma representação gráfica da rede de interconexões dos nós e elos pertencentes à composição) e disponibiliza facilidades (muitas delas por manipulação direta dos objetos gráficos da interface) para o acesso aos métodos de navegação nas composições (*seleciona elo, seleciona nó* e *seleciona pai*) e edição (*cria nó, retira nó, cria elo* e *destrói elo*) definidos no Capítulo 3.

O sistema HyperProp oferece três tipos de editores gráficos ou browsers, como denomina a referência [Much96]: o browser de contexto, o browser de hiperbase e o browser de base. Esta divisão é necessária porque as subclasses da classe composição possuem características específicas, o que implica na necessidade de disponibilizar facilidades distintas para as interfaces de seus respectivos métodos “exibe estrutura”.

O NCM, através do aninhamento de composições, permite a criação de documentos bem estruturados. Portanto, nós de contexto de usuário ou de contexto de versão, normalmente, apresentam estruturas simples, o que viabiliza a visualização de todos os seus componentes

simultaneamente. Esta abordagem diminui o “overhead” para a apresentação do diagrama e permite a edição dos seus elementos, uma vez que todo o conteúdo do contexto está sendo visto. Por outro lado, as redes de interconexões da hiperbase pública e das bases privadas, usualmente, são bastante complexas, exigindo o emprego de mecanismos que melhorem a legibilidade da representação gráfica de seu conteúdo. As operações de edição só são disponibilizadas aos usuários, via interface gráfica, pelos métodos exibe estrutura dos nós de contexto do usuário, contexto de versões e da base privada, não sendo disponibilizado na hiperbase pública, por razões explicadas no Capítulo 3. Estes motivos levaram à definição de três tipos de browsers, um permitindo operações de edição com um mecanismo de exibição mais simples, outro onde a edição não é permitida, mas que fornece maneiras de controlar a complexidade do diagrama e outro que combina todas estas facilidades, como explicitado nos próximos parágrafos.

O browser de contexto apresenta a estrutura de um nó de contexto desenhando representações gráficas para todos os nós e elos pertencentes ao contexto e para todos os elos visíveis a partir da perspectiva corrente. Este browser disponibiliza operações de navegação e edição gráfica do contexto através da inclusão, eliminação e alteração de nós e elos. A edição de elos só será realizada para os elos que pertencerem ao contexto sendo exibido, desconsiderando os que foram herdados pela perspectiva corrente. O browser de contexto implementa a interface para os métodos de exibição e edição das classes contexto do usuário, anotação e contexto de versões.

O browser de hiperbase apresenta uma visão um pouco diferente do mapa, pois a hiperbase, normalmente, possui uma estrutura bem mais complexa que a de um nó de contexto. Por este motivo, são apresentados não apenas os nós e elos do primeiro aninhamento, mas de todos os níveis de aninhamento, para que o usuário tenha uma exata noção da estrutura do documento. Ora, a apresentação de toda essa estrutura pode trazer um *overhead cognitivo* ao usuário, que pode se sentir completamente desorientado depois da apresentação de tal grafo composto. Assim, técnicas de filtragem devem ser empregadas de forma a apresentar ao usuário informações locais e globais do documento, conforme seu foco de interesse. O browser de hiperbase disponibiliza apenas o acesso aos métodos exibição de estrutura e de

navegação, não permitindo o acesso aos métodos de edição. O browser de hiperbase é usado pelo método exibe estrutura da classe hiperbase pública.

O browser de base inclui as facilidades dos browsers de contexto e de hiperbase. Ele disponibiliza, portanto, o acesso aos métodos de exibição de estrutura, navegação e edição do conteúdo do nó. O mapa de componentes é exibido com o emprego de filtros que visam melhorar sua legibilidade, como no browser de hiperbase. O browser de base é usado pelos objetos da classe base privada.

Outro ponto importante a ser observado na fase de autoria de um documento diz respeito à definição de sincronização inter-nós de um documento. Editores e exibidores de estrutura são insuficientes para a definição e exibição do sincronismo temporal e espacial existente entre objetos de um documento. Editores e exibidores de sincronismo assim se fazem necessários. Toda OR-base privada deve ter portanto um método editor e exibidor de sincronismo que facilite ao usuário a definição do sincronismo espacial e temporal. As definições das posições temporais e espaciais de um objeto, em relação a um ponto específico no tempo ou a outros objetos, e também a definição de mudanças do comportamento na apresentação de um nó são capturadas pelas entidades elo e descritor, respectivamente. Assim, da mesma forma que nos browsers de estrutura, a interface do editor de sincronismo disponibiliza o acesso aos métodos das entidades do modelo que capturam a especificação do sincronismo: composições (*cria elo* e *destrói elo*), elos e descritores.

O ideal é que os editores/exibidores de sincronismo trabalhem em conjunto com o editor e exibidor de estrutura para, novamente, evitar a desorientação do usuário, sendo este o caso do browser de base associado ao método *exibe estrutura* da classe base privada.

Neste capítulo nos concentraremos na apresentação do browser de base, uma vez que os outros tipos de browsers disponibilizam, pelas razões supracitadas, apenas subconjuntos de suas funções.

O editor gráfico de estrutura e sincronismo [CMSS96, MSCS97] disponibiliza o acesso aos métodos exibição de estrutura e sincronismo, de navegação e de edição em objetos da

classe base privada definindo três visões do documento. As visões são mostradas na Figura 29.

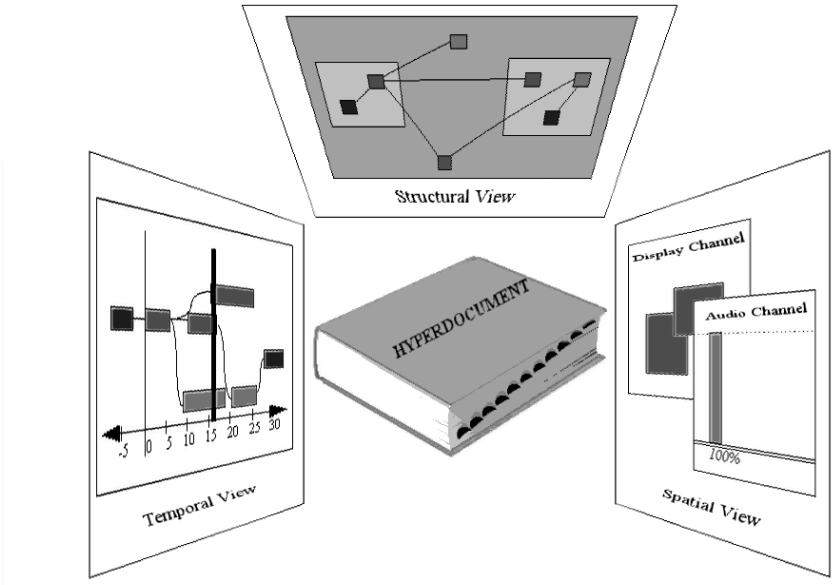


Figura 29 - Diferentes visões de um documento multimídia.

A primeira visão permite editar e visualizar a estrutura do documento, fornecendo recursos para editar fragmentos de informação (nós),[8] seus relacionamentos (elos), e o agrupamento desses nós em composições. Essa visão é apresentada no plano superior da Figura 29, onde fragmentos de informação são representados por retângulos, e o relacionamento de inclusão em composições é representado pela inclusão de um retângulo (nó) em outro retângulo (nó de composição). A visão estrutural é também um componente importante na navegação, pois ajuda a evitar o problema denominado "lost in the hyperspace" [Hala88]. Ao mostrar a estrutura dos documentos, ou parte dela, como um grafo, a visão estrutural apresenta informações importantes sobre contexto e localização no documento, ajudando o usuário a identificar que nós ele está visitando e como estes relacionam-se com seus vizinhos através de elos ou composições.

[8] A visão estrutural permite editar alguns dos atributos dos nós terminais, por exemplo a lista de âncoras. Porém, o atributo conteúdo é editado, bem como exibido, por um exibidor qualquer com facilidades para manipular o tipo do conteúdo do nó em questão. A idéia ao definir esta abordagem para a edição dos conteúdos foi reforçar a característica de sistema aberto do HyperProp, pois ela permite que qualquer exibidor/editor seja usado para exibir componentes dos documentos. Como explica o Capítulo 5, a arquitetura do HyperProp define interfaces entre o módulo do sistema responsável pelo controle da apresentação do documento e os diversos exibidores/editores responsáveis pela apresentação dos componentes do documento; dessa forma, a integração de exibidores/editores é simplificada exigindo apenas a construção de um adaptador (interface) para o HyperProp.

A segunda visão é responsável pela especificação dos relacionamentos temporais entre os componentes de um documento multimídia, definindo suas posições relativas no tempo, como exibido no plano da esquerda da Figura 29. Nesse plano, nós são representados por retângulos, cujos comprimentos indicam a duração ótima de exibição de cada nó e cujas posições relativas definem suas sincronizações no tempo.

A terceira visão permite a definição de relacionamentos espaciais entre componentes de um documento multimídia, estabelecendo suas características de apresentação em um determinado dispositivo, em um dado instante do tempo, como apresentado no plano da direita da Figura 29. Nesse plano, os retângulos representam as características espaciais dos objetos em um dispositivo de exibição, especificando sua posição em um monitor de vídeo, seu volume em um dispositivo de áudio, etc.

4.1. Editor Gráfico para a Manipulação de Documentos Multimídia

A interface do editor gráfico é ilustrada na Figura 30. Cada janela do editor corresponde a uma visão para a manipulação de documentos, conforme anteriormente definido. As visões trabalham de forma cooperativa, permitindo ao usuário editar a estrutura de um documento multimídia e especificar suas características de apresentação temporais e espaciais de forma simultânea e integrada.

Quando o usuário seleciona um nó na visão estrutural, esse nó torna-se o objeto base para definição da sincronização na visão temporal. Essa visão é então atualizada passando a mostrar todos os sinc-elos da cadeia temporal que inclui o nó base.

Na Figura 30, parte de um documento sobre os “Beatles” é exibido como exemplo. No documento, o nó em foco selecionado na visão estrutural é um áudio que contém a canção “Help”. A time view mostra a *time chain* (cadeia síncrona de nós interconectados por sinc-elos) baseada nesse nó. A barra de tempo está posicionada em 5 segundos após o início da time chain. Nesse instante relativo do tempo, o autor especificou que devem ser apresentados a letra da música (nó de texto HelpLyrics), uma foto de John Lenon (nó de

imagem Lenon) e um clip sobre os Beatles (nó de vídeo BeatlesClip). A visão espacial mostra como os nós usam os dispositivos de saída nesse instante.

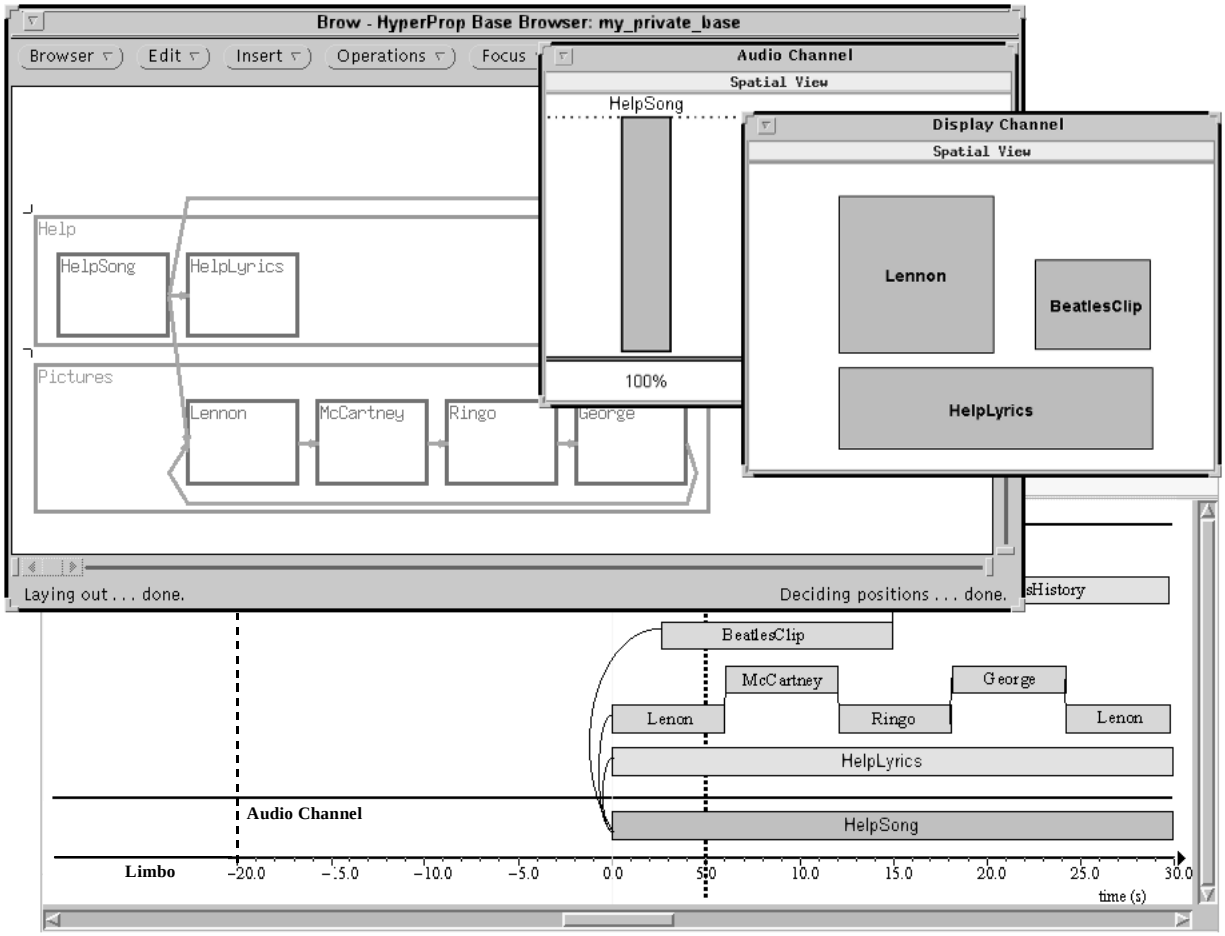


Figura 30 - Interface do editor gráfico para a manipulação de documentos multimídia

Quando o usuário deseja definir relacionamentos de sincronização entre objetos já sendo exibidos na visão temporal e objetos mostrados na visão estrutural, ele deve arrastar os objetos da visão estrutural para a visão temporal. Os objetos, após serem arrastados, são posicionados na origem do eixo do tempo na visão temporal podendo ser, em seguida, movidos para qualquer posição.

Na visão espacial, as janelas representam abstrações de dispositivos de entrada/saída denominados canais. Uma figura no interior das janela define o espaço que será ocupado por um componente do documento no dispositivo onde ele será exibido em um determinado

instante do tempo. O instante no tempo é definido por uma barra vertical apresentada na visão temporal.

As Subseções 4.4.1.1 e 4.4.1.2 detalham as principais características de cada visão.

Outros trabalhos, como por exemplo os editores dos sistemas CMIFed [RJMB93, HaRB93] e Mode [BHLM92], também usam múltiplas visões para editar e visualizar documentos.

O CMIFed é um ambiente de autoria e apresentação de documentos baseado no Amsterdam Hypermedia Model (AHM) [HaBR94]. O CMIFed provê três visões para edição e visualização dos documentos: hierárquica, canal e exibição. A *visão hierárquica* mostra o documento como uma coleção estruturada de blocos de dados que devem ser exibidos em série ou em paralelo. A estruturação dos blocos de dados na visão hierárquica gera implicitamente informações de sincronização: blocos de um mesmo nível da hierarquia são exibidos em paralelo e blocos em níveis diferentes da hierarquia são exibidos em série começando pela raiz e seguindo para os níveis inferiores. A *visão dos canais* mostra o mapeamento no tempo dos blocos de dados (folhas na visão hierárquica) nos canais,[9] mostrando então informações relativas à sincronização temporal dos blocos de dados do documento. Na visão dos canais, é permitida a especificação adicional de arcos de sincronização que explicitam restrições de sincronização adicionais aos blocos de dados mostrados nessa visão. O player mapeia uma especificação de apresentação de um documento em um ambiente de apresentação particular. O player é integrado às visões hierárquica e canal, permitindo ao usuário solicitar a exibição de trechos do documento a partir das outras duas visões. Na visão player, o autor do documento pode selecionar um nó e ativar um menu que disponibiliza funções para edição dos atributos dos nós, destacar o nó nas visões hierárquica e canal ou para escolha de um editor de mídia apropriado para o conteúdo do nó. O posicionamento dos canais de display (janelas) na tela é também realizado na visão player. Comparando o editor de documentos do sistema HyperProp com o CMIFed, observamos que a visão estrutural do HyperProp não mostra nenhuma informação relativa ao tempo, permitindo então combinar hiper-elos (relacionamentos

[9] Um *canal* no modelo de documentos AHM é um dispositivo de saída abstrato que é usado para exibir eventos. Um canal pode ser, por exemplo, uma janela na tela, ou um dispositivo de saída de áudio. O canal inclui informações default para a apresentação dos blocos de dados a ele associados, por exemplo fonte e estilo para canais de texto, ou o volume para um canal de áudio.

assíncronos) e o aninhamento de composições (que pode ser usado para definir uma estrutura hierárquica para os documentos) em uma representação gráfica simples e consistente. A visão estrutural, como será explicado na Seção 4.4.1.1, permite visualizar simultaneamente partes distintas da estrutura dos documentos, permitindo a edição de hiper-elos, o que não é possível de ser feito na visão hierárquica da versão do CMIFed descrita nas referências supracitadas. A visão hierárquica mostra apenas o aspecto composição da estrutura da apresentação dos documentos, não fornecendo uma visão estruturada dos hiper-elos [HaRB93]. A visão canal do CMIFed possui objetivos semelhantes aos da visão temporal do editor do HyperProp: apresentar relacionamentos síncronos entre os componentes do documento em uma interface baseada em um eixo do tempo. Entretanto, no CMIFed, os relacionamentos temporais são definidos por restrições temporais (arcos de sincronização) e não por elos baseados em condições-ações como no caso do HyperProp e do padrão MHEG. A visão espacial do editor do HyperProp provê uma simulação da exibição dos componentes, enquanto no CMIFed o player proporciona uma apresentação real dos componentes. No HyperProp a exibição é realizada por outro módulo do sistema, chamado executor, descrito no Capítulo 5. A separação entre a simulação e a execução da exibição permite que os autores possam verificar como será o comportamento da exibição do documento que está preparando em plataformas distintas da que usa para editar o documento, bastando para tal configurar os canais (no HyperProp os canais são abstrações de dispositivos) de acordo com a plataforma de exibição.

O editor de sincronização do sistema Mode [BHLM92] também usa três visões 2D para manipular os documentos. Na primeira delas, chamada de presentation view, é possível exibir e editar objetos independentes. A segunda visão, chamada time view, é usada para visualização e edição de relacionamentos temporais entre os objetos. A terceira visão é chamada layout view. Nesta visão, é possível editar o layout (posicionamento espacial das janelas e das características de apresentação) relativo a um dado instante do tempo da apresentação. Várias instâncias de cada uma das visões podem ser abertas em paralelo. As visões time e layout do Mode são semelhantes às visões temporal e espacial do editor do HyperProp. Na presentation view são definidos pontos de referência (âncoras no NCM), e é feita a associação entre pontos de referência através da definição de pontos de sincronização (uma simplificação dos elos NCM). No editor do HyperProp essas tarefas são

realizadas na visão estrutural em conjunto com os editores de mídia. A principal diferença entre os editores é que o do Mode não provê nenhuma facilidade para visualização da estrutura dos documentos.

4.1.1. Visão Estrutural

A visão estrutural, descrita em detalhes em [Much96], mostra um mapa dos documentos (composições aninhadas) manipulados por um usuário em uma sessão de trabalho. Caso se deseje visualizar com detalhes todos os níveis de aninhamento dos documentos, pode-se obter um diagrama bastante complexo, requerendo o emprego de métodos sofisticados no desenho dos mapas de documentos. Filtros para esconder informações e landmarks (nós especiais que sempre aparecem) são usualmente necessários.

Manter a legibilidade dos mapas é o principal objetivo ao se usar filtros para esconder informações. Para a construção desses filtros, foi utilizada uma extensão da estratégia de visões olho-de-peixe para modelos conceituais que oferecem nós de composição [Much96].

A motivação fundamental da estratégia de olho-de-peixe para documentos [Furn86] é balancear detalhes locais e o contexto global em relação ao nó selecionado (em foco) pelo usuário. Detalhes locais são necessários para dar aos usuários informações sobre suas possibilidades de navegação, dependentes da sua localização na estrutura do documento. O contexto global é importante para dar aos usuários informações sobre sua posição dentro da estrutura global do documento.

A estratégia básica do olho-de-peixe propõe uma função de grau de interesse que atribui a cada nó um valor representando seu interesse para o usuário. Esta função é decomposta em duas componentes. A primeira é chamada de importância a priori e descreve a importância intrínseca de um nó considerando a estrutura do documento. A segunda componente determina a distância entre o nó para o qual a função está sendo calculada e o nó em foco pelo usuário. A essência da visão olho-de-peixe é que a função de grau de interesse aumenta com a importância a priori (API) e diminui com a distância (D).

Na estratégia olho-de-peixe estendida, o grau de interesse do usuário para cada nó depende da perspectiva na qual está inserido. Um mesmo nó terá um valor da função de grau de

interesse para cada uma de suas perspectivas. A função de grau de interesse deve ser calculada para todas as perspectivas da estrutura do documento. Detalhes sobre essa função podem ser encontrados também na referência [Much96].

Uma vez que um mesmo nó pode ser inserido em diferentes composições, ele também pode ser desenhado em diferentes posições no mapa. Por outro lado, exibir um nó em uma de suas perspectivas não significa que será exibido em todas as outras. Como a posição de um nó em uma dada perspectiva depende de quais perspectivas estão sendo exibidas no mapa, os nós de um documento não têm uma posição fixa na tela. Para reduzir esse problema, o usuário pode definir algumas perspectivas como pontos de referência, isto é, perspectivas que serão sempre exibidas no mapa, pelo menos parcialmente, ajudando o usuário a melhorar o seu senso de localização. Esses pontos de referência são chamados *landmarks* e são específicos a cada usuário. Um usuário não precisa concordar com o conjunto de landmarks de qualquer outro usuário, cada usuário pode possuir o seu próprio conjunto.

As idéias propostas na referência [Much96] consideram o fato de que, em modelos com nós de composição aninhados, as composições estruturam o documento. Assim, é possível construir editores que exploram o recurso de “explodir” ou “implodir” nós de composição, mostrando ou escondendo seus componentes, dependendo do interesse atual do usuário. Esse recurso limita a quantidade de informação detalhada apresentada no mapa, melhorando sua legibilidade.

Para ilustrar melhor as técnicas de filtragem utilizadas pelo editor de estruturas, a Figura 31 e a Figura 32 apresentam um exemplo do uso de visões olho-de-peixe na implementação do browser de hiperbase do sistema HyperProp. O documento apresentado é representado por um nó de composição *A* (toda a janela). As relações de inclusão em nós de composição são representadas pela inclusão de retângulos (nós) em outros retângulos (nós de composição). As figuras mostram como o mapa é modificado dependendo do nó em foco e do nível de detalhe escolhido pelo usuário.

Para apresentar as visões olho-de-peixe, foi utilizada uma técnica de layout automático de grafos [SuMi91]. Como a estrutura dos nós e elos visíveis muda muito, o diagrama pode mudar freqüentemente de uma visão para outra. Mudanças repentinas e drásticas de

diagramas geralmente destróem o mapa mental do usuário, diminuem a eficiência do trabalho e podem causar desorientação. Com o objetivo de reduzir esses problemas, o editor mostra mudanças dos diagramas com animação. Esse recurso reduz a mudança visual instantânea, preservando o mapa mental do usuário e ajudando sua orientação.

A edição da estrutura de documentos tem sido investigada por vários projetistas de sistemas multimídia/hipermídia. Geralmente, eles não oferecem uma ferramenta genérica que pode ser moldada para satisfazer aos interesses do usuário quando edita ou navega pela estrutura de nós e elos. Normalmente, ou o sistema oferece um mapa global mostrando uma visão ponto-a-ponto de todos os nós e elos de um documento, ou mostra um mapa local indicando a posição do usuário, de onde ele veio e para onde ele pode ir, apresentando apenas os nós que são adjacentes ao nó corrente.

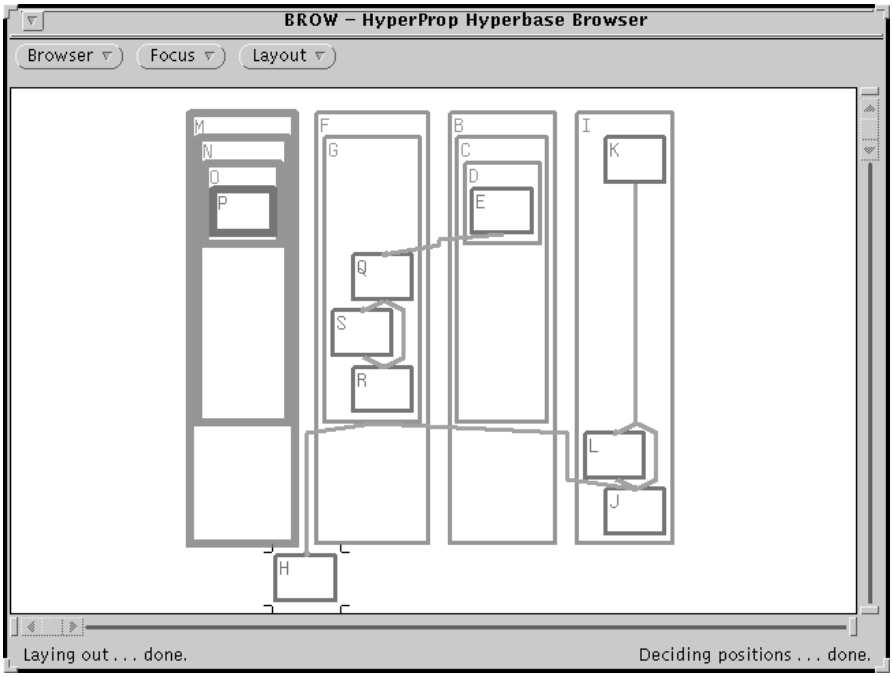
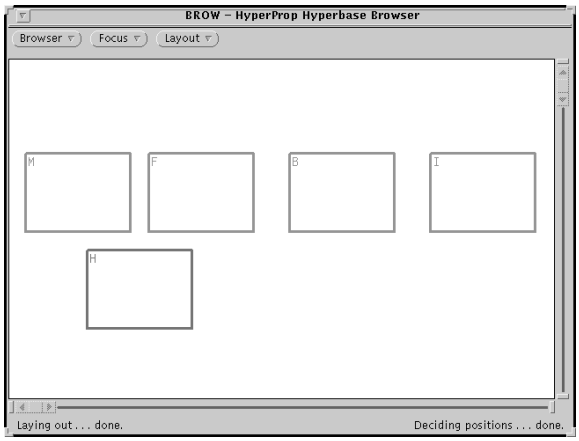
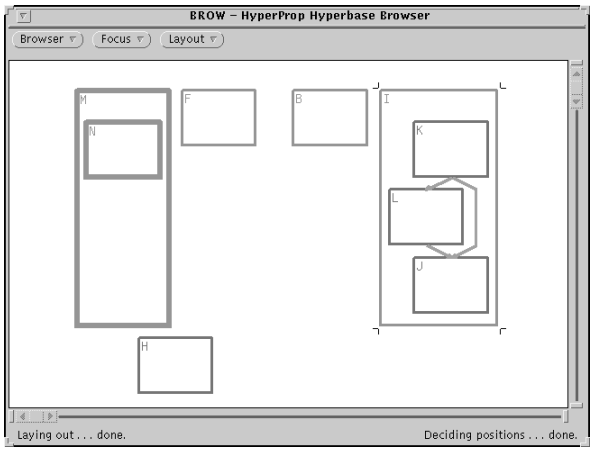


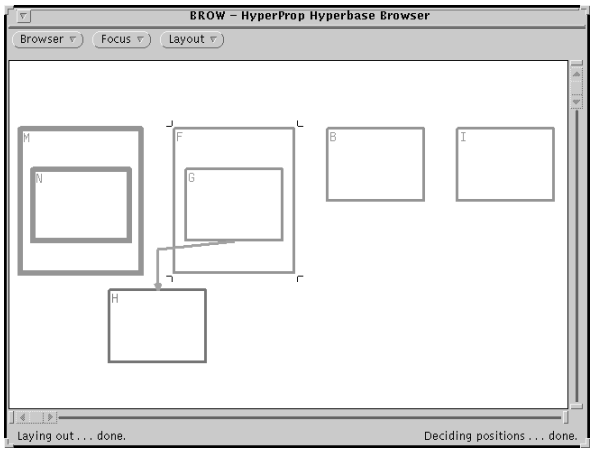
Figura 31 - Visão detalhada dos componentes aninhados do nó de composição *A*. <*M,N,O,P*> é um landmark.



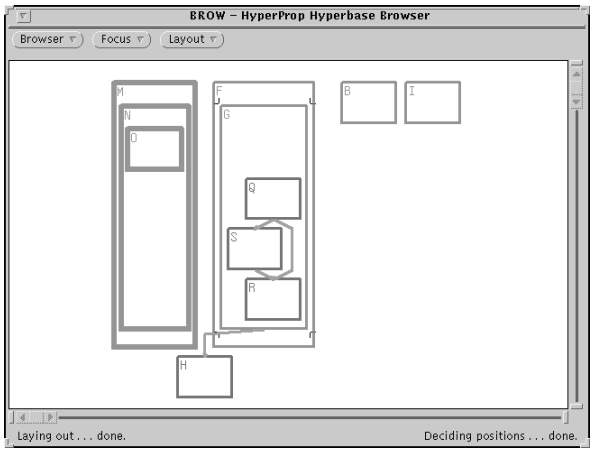
a) Visão quando o editor abre o nó A



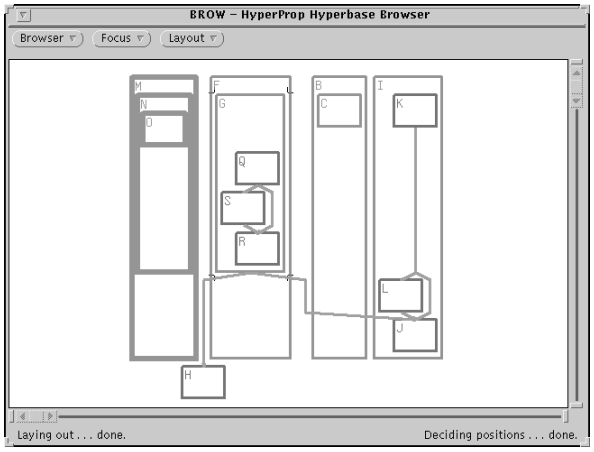
b) Nó *I* em foco



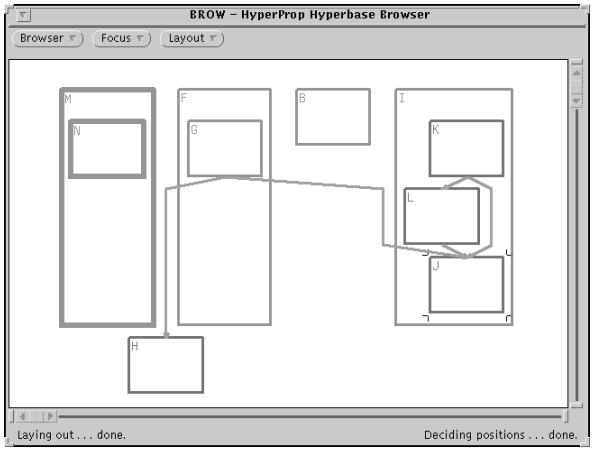
c) Nó *F* em foco



d) Nó *G* em foco



e) Nó *G* em foco (mais detalhes)



f) Nó *J* em foco

Figura 32 - Exemplo das visões olho-de-peixe de um nó de composição *A*.

Visões estruturais para documentos foram propostas pelos sistemas Neptune, Notecards, CYBERMAP, Storyspace, SHADOCS, Intermedia, entre outros. A abordagem mais interessante e que foi estendida no presente trabalho é a do CYBERMAP [Gloo91], que agrupa nós em estruturas chamadas HyperDrawers, através da análise de seu conteúdo e de

suas características, sem considerar os elos do documento. A desvantagem dessa técnica é que as relações modeladas por elos são ignoradas, além da criação dos HyperDrawers ser bastante complexa. Ao contrário, no modelo com composições aninhadas utilizado neste trabalho, os grupos de nós são dados diretamente pela hierarquia de composições e pelos elos. Esse fato permite a construção de visões que não exibem necessariamente todos os nós do documento de forma simples e preservando a exibição dos relacionamentos definidos nos elos.

4.1.2. Visões Temporal e Espacial

Um editor de sincronismo deve permitir ao autor especificar os eventos relevantes e definir as relações entre eles provendo uma interface gráfica para definição de características de apresentação dos documentos. Através dessa interface o editor de sincronismo permite a definição das posições temporais e espaciais de um objeto, em relação a um ponto específico no tempo ou a outros objetos, e também a definição de mudanças do comportamento na apresentação de um nó. Essas relações são capturadas pelas entidades elo e descritor, respectivamente.

O principal objetivo do editor de sincronismo, descrito em detalhes em [Cost96], é facilitar a edição dos aspectos de apresentação de documentos multimídia. Com esse propósito, a interface gráfica mostra duas visões dos documentos: a visão temporal (temporal view) e a visão espacial (spatial view).

A visão temporal, ilustrada pela Figura 33, mostra o posicionamento dos componentes de um documento ao longo de sua apresentação. Nesta visão, os objetos são desenhados nas áreas de áudio e display, que correspondem a abstrações dos dispositivos da plataforma de exibição. As representações dos nós são desenhadas como retângulos, cujos comprimentos correspondem às durações ótimas de suas exibições. Nessa visão, os elos, cujas extremidades são eventos previsíveis, isto é, eventos cujos momentos de ocorrência podem ser calculados a priori, são desenhados como linhas que conectam os objetos. Esse tipo de elo, conforme explicado no Capítulo 3, é chamado *elo de sincronismo* (sinc-elo).

A visão é focada em um objeto chamado *objeto base*, que é posicionado na origem do eixo do tempo. Os demais objetos ficam posicionados no tempo especificado em relação ao

objeto base, que corresponde ao objeto em foco na visão estrutural, anteriormente descrita. As regras para inclusão de objetos na time view são as seguintes:

- *Inclusão de nós antecessores*: Se um nó incluído na visão for destino de um e apenas um elo de sincronismo (relação 1:n), o nó origem desse elo é incluído na visão na posição determinada pelo elo.
- *Inclusão de nós posteriores*: Se um nó é incluído na visão, todos os nós de destino de seus elos de sincronismo que têm a condição de disparo satisfeita, isto é, os nós em que é possível determinar sua exibição no tempo, são também incluídos na visão.

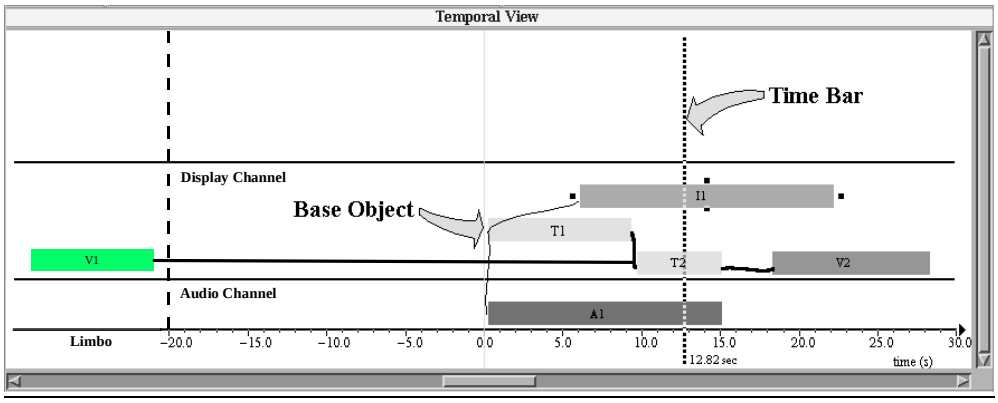


Figura 33 - Visão temporal

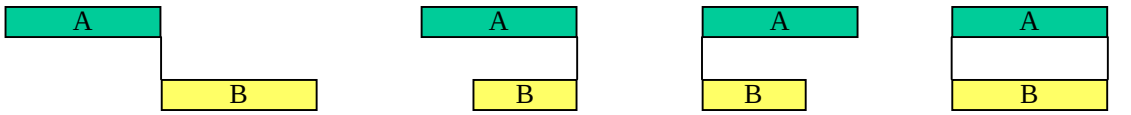
Note que essas regras são recursivas e são aplicadas a todos os nós incluídos na *time view*, iniciando-se pelo objeto base. A coleção de nós resultante da aplicação dessas regras de inclusão é chamada de *cadeia temporal*.

O usuário pode incluir explicitamente um nó na time view, para a definição de uma nova relação de sincronismo, simplesmente o arrastando (“dragging”) da visão estrutural para a visão temporal. Quando um nó *N* é incluído pelo usuário na time view, ele é inicialmente posicionado na origem do eixo do tempo. Um elo de sincronismo (relação 1:1), tendo o objeto base como origem e *N* como destino, é, então, automaticamente criado pelo editor.

O usuário pode trocar a posição relativa de objetos, desde que seja mantida a consistência do sincronismo, a qual é verificada pelo editor. Quando o usuário destrói um elo *E* na time view, os nós e elos que foram posicionados iniciando em *E* deixam de ser mostrados na visão temporal.

A visão temporal possui uma região, chamada de região de limbo, onde são colocados os nós cujo instante de apresentação não pode ser determinado com exatidão. Estes nós são arrastados da visão estrutural para a região de limbo na visão temporal. Quando um nó é colocado nesta região, nenhum elo de sincronização é criado entre ele e o objeto base na visão temporal. Os eventos associados aos nós colocados na região de limbo são considerados como eventos ocorridos em algum instante anterior ao do início da cadeia temporal corrente, mas cujo instante de ocorrência não pode ser determinado. O posicionamento de objetos na região de limbo permite manipular elos de sincronização mesmo quando não é possível calcular com precisão o momento da ocorrência de seus eventos de origem.

O usuário pode, opcionalmente, definir o sincronismo entre objetos utilizando funções de alto-nível da interface para especificar relacionamentos temporais entre intervalos [Alle83, Hamb72] que necessitam de um alinhamento preciso de eventos: *starts*, *equals*, *meets* e *finishes*. Essas relações provocam a criação automática de elos entre os nós envolvidos, como mostra a Figura 34. Note que o relacionamento *equals* requer que a exibição dos objetos envolvidos possua a mesma duração.



(a) A meets B (b) A finishes B (c) A starts B (d) A equals B

Figura 34 - Relacionamentos de alto-nível entre intervalos.

A time view possui um mecanismo de *scroll* que permite a visualização de todos os instantes de tempo de uma time chain. Ela possui ainda uma barra de tempo móvel (mostrada na Figura 33) que pode ser posicionada em qualquer instante. Se a barra de tempo estiver sobre um ou mais nós, esses nós serão mostrados na visão espacial, dentro dos canais onde eles serão apresentados.

Na *visão espacial* (ver Figura 35), o usuário edita o sincronismo espacial dos objetos posicionando-os em canais apropriados. Os canais são abstrações dos dispositivos de entrada e saída da plataforma onde será exibido o documento. Através desse posicionamento, o usuário define, por exemplo, o espaço no display que será usado para mostrar uma janela com uma imagem gráfica, o volume que será usado para tocar um nó de

áudio, etc. Quando o usuário muda o comportamento (posição, volume, etc.) de um objeto na visão espacial, o sistema inclui no descritor uma operação especificando a mudança de comportamento na exibição da representação do nó. Essa operação é inserida na lista de operações associada a um evento que ocorre no instante do tempo da apresentação do objeto definido pela posição da barra de tempo.

Figura 35 - Visão espacial.

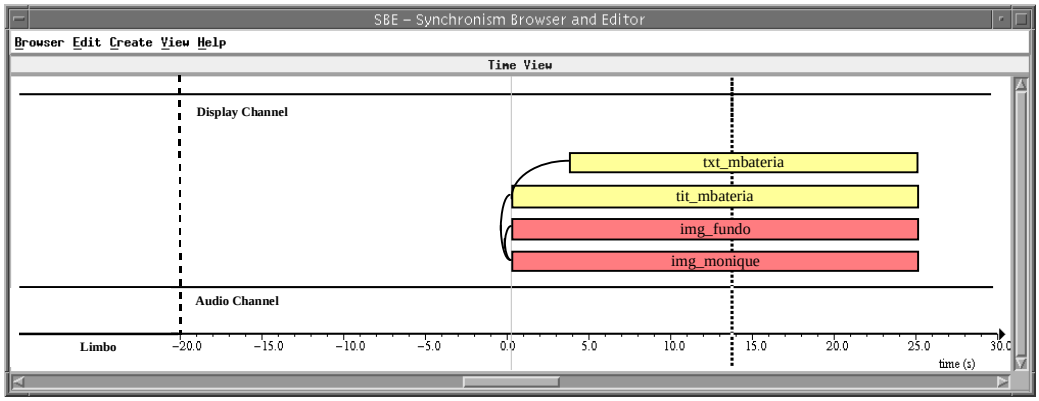
Se, na visão espacial, existirem dois ou mais objetos de áudio escalonados para serem exibidos no mesmo canal ao mesmo tempo, o tamanho dos retângulos indica como a mixagem dos sinais será feita. Analogamente, a ordem de sobreposição dos retângulos no canal de display determina como as janelas serão apresentadas no monitor de vídeo. A referência [Cost96] traz uma descrição completa da interface com o usuário do editor de sincronismo.

Um exemplo de especificação de uma mudança de comportamento usando as visões temporal e espacial é mostrado nas Figuras 8 e 9. A Figura 36 retrata a exibição de parte de um documento sobre o Carnaval Brasileiro mostrando características de apresentação diferentes de um nó imagem que contém uma foto (nó “img-monique”). A Figura 37 mostra as visões temporal e espacial deste documento. Na visão temporal (ver Figura 37-a) pode-se observar a duração no tempo da exibição dos objetos de representação envolvidos. No início da apresentação, o nó “img-monique” ocupa um espaço maior e depois passa a ocupar um espaço menor e é colocado em uma posição diferente na tela. Esta modificação no comportamento pode ser observada nas visões espaciais. A Figura 37-b mostra a disposição espacial dos nós de imagem “img-monique” e “img-fundo” e do nó de texto “tit-mbateria” no início de suas apresentações. A Figura 37-c mostra a disposição dos mesmos nós 13,8 segundos após o início da apresentação. O editor de documentos captura a modificação no comportamento do nó “img-monique” e inclui uma operação comandando a modificação no descritor associado à representação do nó.

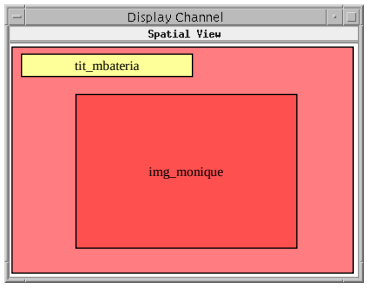
a) Apresentação no instante x segundos

b) Apresentação no instante (x + 11.8) segundos

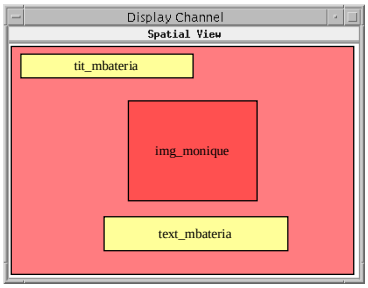
Figura 36 - Apresentação de documento sobre o Carnaval Brasileiro



a) Visão temporal



b) Visão espacial no instante x segundos



c) Visão espacial no instante (x + 11.8) segundos

Figura 37 - Visões temporal e espacial correspondentes a Figura 8.

No Capítulo 2 foram apresentados os principais métodos usados para descrição do sincronismo na apresentação de documentos, a saber, baseado em intervalos, hierárquico, eixo de tempo, script, redes de Petri, pontos de referência e em eventos. Os diferentes métodos de especificação possuem facilidades distintas que podem ser combinadas para facilitar a tarefa de autoria dos documentos. Por exemplo, o editor CMIFed [RJMB93, HaRB93] usa o método baseado em hierarquia em sua visão hierárquica combinado com uma mistura dos métodos baseados em eixo de tempo e em pontos de referência na sua

visão canal. O editor de documentos Firefly [BuZe92] adota uma combinação dos métodos baseado em pontos de referência e em eventos.

O editor de documentos descrito neste capítulo também usa uma combinação de métodos para especificação do sincronismo. A visão temporal permite que o sincronismo seja especificado através de funções de alto-nível que relacionam intervalos em conjunto com funções de nível mais baixo que permitem que a especificação seja feita através dos método baseado em eventos e em eixo de tempo. Estes paradigmas são então mapeados para em um modelo orientado a eventos, definidos nos elos e descritores.

Capítulo 5

Execução de Apresentações de Documentos

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.

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. Essas especificações, que foram assunto dos capítulos anteriores, servirão de entrada para o ambiente de execução, para que a apresentação, que é o resultado final desejado, ocorra.

É função do ambiente de armazenamento oferecer as ferramentas para armazenar de forma persistente e recuperar os objetos multimídia em um sistema de informação distribuído, deixando transparente a localização dos mesmos. Cabe a esses ambientes, também, oferecer uma qualidade de serviço (QoS) adequada para recuperação das diversas mídias, em especial as mídias contínuas, como áudio e vídeo.

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 garantir tais requisitos. 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 futuras (por exemplo, realizar pré busca), ou reagir em caso de alterações no comportamento esperado. Assim, as principais tarefas do ambiente de execução são 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 seria no formato da estrutura do modelo de dados interno ao 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, modelos de mais alto nível têm se mostrado mais adequados para utilização na autoria, sendo a utilização de editores gráficos na especificação uma tendência também observada. Como exemplos podem ser citados os sistemas Firefly [BuZe93], CMIF [RJMB93] e HyperProp, descrito nos Capítulos 3 e 4. A tradução da abstração de mais alto nível para o modelo de dados interno do ambiente de execução torna-se, assim, a ponte entre o ambiente de autoria e o de execução.

Em sua versão corrente, o ambiente de execução do HyperProp é implementado como um conjunto de métodos da base privada, controlando a apresentação de todos os documentos em uma sessão do usuário.

Este capítulo apresenta o ambiente de execução do HyperProp [Rodr97, RoSS97a, RoSS97b], cuja implementação foi o tema do trabalho apresentado na referência [Rodr97]. Para tal, o capítulo está organizado da seguinte forma. A Seção 5.1 aborda os requisitos de um ambiente de execução. A Seção 5.2 propõe uma arquitetura genérica para o ambiente. A Seção 5.3 apresenta o ambiente de execução do sistema HyperProp. A Seção 5.3.1 apresenta a arquitetura do ambiente. Na Seção 5.3.2 é feita uma discussão sobre a

semântica das entidades do NCM que capturam a especificação do sincronismo temporal e de como esta especificação pode ser usada para verificar a consistência[10] do sincronismo temporal em documentos NCM. A Seção 5.3.3 apresenta, resumidamente, duas propostas para o modelo de dados usado no ambiente de execução como base para controle da apresentação de documentos. Por fim, na Seção 5.3.4 são descritas as interfaces internas do ambiente e as interfaces externas com os demais subsistemas.

5.1. Requisitos de um Ambiente de Execução

O ambiente de execução é responsável por preparar e controlar a apresentação dos documentos. Para tanto, como mencionado, um plano de execução deve ser construído, baseado na especificação do documento, a fim de orientar tal tarefa. O plano de execução é um cronograma de eventos, onde a cada início ou fim de evento um instante esperado de ocorrência é associado. O início ou o fim de um evento é instantâneo sendo denominado *ponto de sincronização*.

O instante de ocorrência de um ponto de sincronização, relativo a outro 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. 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.

Usando e estendendo a taxinomia apresentada em [BuZe93], um ambiente de execução de documentos multimídia pode ser analisado quanto ao momento em que constrói o plano de execução, à flexibilidade na sua construção, aos relacionamentos temporais suportados, às

[10] No escopo deste trabalho, um documento é dito ser consistente quando sua apresentação termina decorrido um tempo finito após seu início.

mudanças de comportamento admitidas, às inconsistências analisadas e a como reage às variações imprevisíveis.

Um ambiente de execução 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]), apenas a fase de execução (Trellis [StFu90]), as fases de compilação e execução (Firefly [BuZe93], CMIF [RJMB93]), e as três fases juntas (HyperProp). O que os diferencia é o momento em que o plano de execução é construído, se antes da apresentação (na pré-compilação, ou na compilação, ou na pré-compilação e compilação); se durante a apresentação (na execução); ou em ambos os momentos (na pré-compilação e execução, ou na compilação e execução, ou ainda na pré-compilação, compilação e execução). Na Seção 5.2 é feita uma discussão sobre as três fases.

Independente de quando construído, o plano de execução pode ter, ou não, flexibilidade quanto ao momento 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, os 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, como especificado pelo NCM no Capítulo 3.

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

A geração do plano pode dar suporte às mudanças comportamentais especificadas na fase de autoria, tais como aumentar o volume do áudio em um dado instante, acelerar o vídeo, etc., bem como às mudanças de comportamento provenientes da interação dinâmica com o usuário no controle da taxa de apresentação do documento como um todo, denominadas meta-diretivas.

Durante a geração do plano, o ambiente de execução pode sinalizar inconsistências temporais e espaciais. No primeiro caso, está a detecção de falhas na especificação do autor quanto à disposição temporal na exibição do documento, que resulta em uma apresentação impossível de ser realizada. No outro caso, está a detecção de limitações na plataforma de exibição, como por exemplo, a ausência de suporte a áudio numa determinada estação (inconsistência de plataforma), ou o conflito por um determinado recurso.

Finalmente, os ambientes de execução podem ser classificados quanto à forma que controlam e atualizam o plano de execução quando da variação na taxa de apresentação esperada. Exemplos dessas variações imprevisíveis 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.

5.2. Arquitetura Genérica de um Ambiente de Execução

Em [BuZe93] é descrita uma arquitetura genérica para um ambiente de execução (formatador temporal na linguagem das autoras). Esta seção estende a arquitetura por elas introduzida, salientando as diversas interfaces internas e externas necessárias à discussão da Seção 5.3. A arquitetura possui quatro elementos básicos: o Pré-Compilador, o Compilador, o Executor e os Controladores de Exibição. A Figura 38 ilustra a arquitetura.

Embora realize funções do ambiente de execução, o Pré-Compilador é um elemento de suporte ao ambiente de autoria, verificando as consistências temporais e espaciais para cada uma das cadeias temporais parciais, independentemente, dada uma especificação de apresentação. A verificação pode ser feita de forma dinâmica, conforme o autor cria os relacionamentos, alertando para possíveis erros de especificação. O Pré-Compilador pode ou não alimentar o Compilador, ou o Executor, com as cadeias temporais parciais por ele analisadas. Quando a especificação da apresentação é feita independente da plataforma de exibição, não faz sentido o Pré-Compilador testar consistências de plataforma.

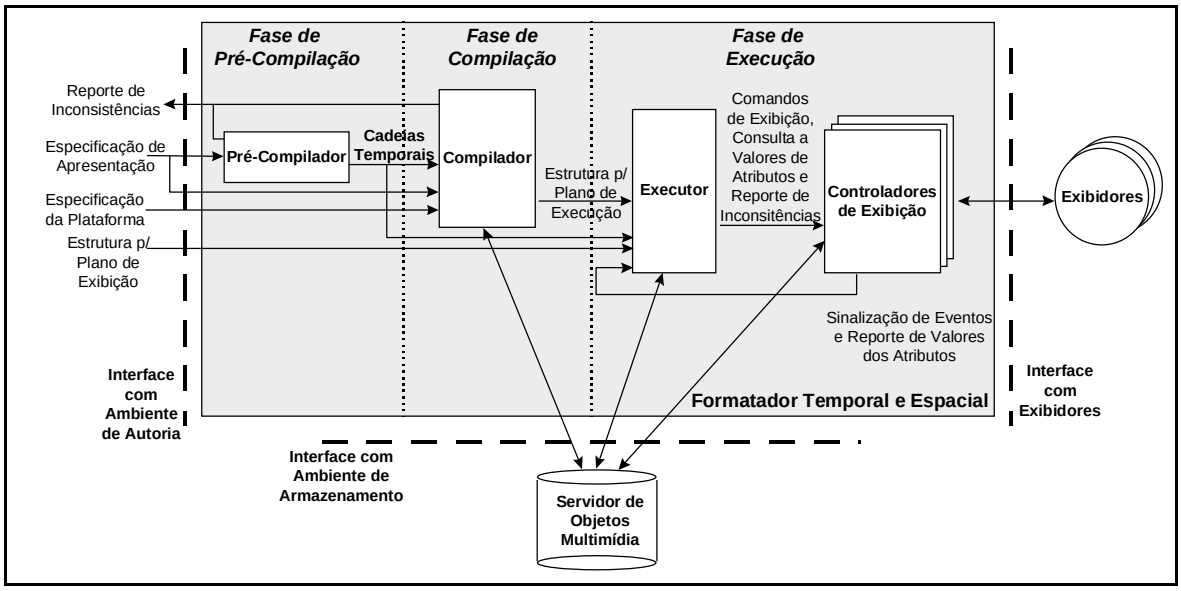


Figura 38 - Arquitetura Genérica do Ambiente de Execução

O Compilador é responsável por receber a especificação completa da 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 especificação da plataforma onde o documento será exibido, para que possam ser feitos os testes de consistência de plataforma. 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 compilação.

O Executor é o elemento destinado às questões que só podem ser resolvidas no momento da apresentação do documento e que, consequentemente, não puderam ser solucionadas pelo Pré-Compilador e pelo Compilador. Ao Executor cabe instanciar os Controladores de Exibição e reagir às sinalizações de ocorrência dos eventos por eles informadas, conforme a especificação do autor, bem como reagir às mudanças de comportamento realizadas pelo usuário em tempo de exibição. É através das sinalizações recebidas dos Controladores e de

informações de QoS de toda a plataforma (estação e rede) de tratamento do documento que o Executor pode atuar, através de comandos, ajustando em tempo de execução a exibição prevista. Como já mencionado, a especificação de apresentação pode ser feita direto 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 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.

Os Controladores de Exibição são responsáveis pela interface do sistema com os objetos em exibição. Deve haver uma instância de Controlador para cada objeto, e para cada composição de objetos do documento sendo exibida. Os Controladores recebem dos objetos as sinalizações de ocorrência de eventos, tais como interação do usuário ou apresentação 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. Também fica a cargo dos Controladores requisitar os conteúdos de dados do objeto a ser apresentado ao subsistema de armazenamento, negociando parâmetros de QoS para entrega dos mesmos.

5.3. O Ambiente de Execução do Sistema HyperProp

A presente seção descreve o ambiente de execução do HyperProp. Inicialmente é feita uma revisão das características e objetos do NCM necessárias à compreensão do ambiente de execução. Em seguida é apresentada a arquitetura do ambiente de execução. A Seção 5.3.2 apresenta uma especificação, usando a Técnica de Descrição Formal RT-LOTOS [CoOl94], das entidades NCM que capturam a especificação do sincronismo temporal. Esta seção discute também como esta especificação pode ser usada para verificar a consistência da especificação do sincronismo em documentos NCM. A Seção 5.3.3 traz uma discussão sobre propostas para estrutura de dados gerada pelo compilador e usada pelo executor do

sistema HyperProp como base para geração dos planos de exibição. Por fim, a Seção 5.3.4 apresenta uma descrição das interfaces entre os vários módulos do ambiente de execução.

5.3.1. Arquitetura

A arquitetura do ambiente de execução do sistema HyperProp segue a ilustrada na Figura 38, sendo composta pelos módulos Pré-Compilador, Compilador, Executor e pelos Controladores de Exibição. Na implementação atual, o Pré-Compilador realiza apenas os testes de consistência temporal das cadeias temporais parciais, 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 poderão servir de entrada para o Compilador, otimizando a tarefa de compilação.

Além dos componentes internos do ambiente de execução, a Figura 38 apresenta as interfaces com o subsistema de armazenamento (no caso, um servidor de dados NCM) e com os exibidores. O sistema HyperProp não implementa exibidores especiais de nós de conteúdo, apenas de nós de composição (necessários para exibição da estrutura do documento, conforme descrito no Capítulo 4). A idéia do sistema é incorporar exibidores já disponíveis comercialmente, através da implementação dos Controladores. Assim, para cada exibidor, deve ser implementado um Controlador que tenha conhecimento de seu funcionamento interno. Dessa forma, a *interface 3*, apresentada na Figura 38, entre os Controladores e os exibidores, não é relevante para o Executor, sendo dependente de cada exibidor. A *interface 2*, entre o Controlador e o Executor, assim como a *interface 1*, entre o Controlador e o Servidor NCM, são o assunto da Seção 5.3.4. Essa arquitetura faz com que o sistema atenda aos requisitos de interoperabilidade a que se propõe, para exibição dos documentos.

Quando, ao ambiente de execução do sistema HyperProp, é requisitada a apresentação de um determinado documento, o módulo Compilador busca, do Servidor NCM, todas as composições que definem o documento, recursivamente. As composições são, então, armazenadas no ambiente de execução, 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. De posse de todos os elos do documento, as expressões contidas nos mesmos são avaliadas, a fim de obter todos os eventos possíveis e, dessa forma, criar as cadeias temporais parciais. A cada evento da cadeia, o Compilador associa os elos cujas expressões devem ser avaliadas, no momento de sua ocorrência. A união de todas as cadeias temporais parciais, através de seus pontos de sincronização imprevisíveis, compõe uma máquina de estados de exibição. Cabe ao Compilador passar essa estrutura ao Executor, para que este construa o plano de execução. A referência [Rodr97] advoga que a estrutura tenha como modelo uma rede de Petri temporizada, TSPN (Time Stream Petri Net), conforme definido em [DiSé94], com uma pequena extensão, descrita na Seção 5.3.3. O emprego da técnica de descrição formal RT-LOTOS para e descrever as entidades que capturam o sincronismo temporal no NCM, Seção 5.3.2, abriu uma outra possibilidade para a estrutura base do executor: o DTA (Dynamic Timed Automaton) gerado pela ferramenta RTL [CoOl94] a partir da descrição de entidades NCM em RT-LOTOS. Ambas as estruturas estão sendo alvo de investigações. A versão corrente do executor [Rodr97] não implementa nenhuma delas, mas sim uma estrutura que permite, quando ocorre um evento, verificar se algum elo dispara e, como consequência, despacha as ações do elo disparado para os correspondentes eventos de destino. Embora tenha se avançado muito na direção de uma solução para a avaliação da consistência do sincronismo temporal e espacial dos documentos com a utilização da técnica de descrição formal RT-LOTOS, conforme discutido na Seção 5.3.2, a verificação de consistência é realizada na implementação corrente apenas na fase de pré-compilação e apenas para cadeias parciais e não para o documento como um todo.

Quando a ação de exibição de um determinado nó deve ser disparada (evento de exibição), 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 necessita consultar o 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. No caso em que o nó a ser exibido é de composição (visualização da estrutura de parte do documento), e o mesmo encontra-se na cache do ambiente, o Executor passa para o Controlador o próprio objeto de dados. Caso contrário, ele informa apenas o seu identificador. Em qualquer caso, o Executor informa apenas o identificador do descritor. Além dessas informações, o Executor passa também ao Controlador, a lista de eventos que este deve reportar, obtida através da estrutura de dados recebida do Compilador.

Cabe ao Controlador criar o objeto de representação, buscando o descritor (e o objeto de dados, no caso de só ter recebido a identificação do mesmo) no Servidor NCM. De posse do descritor, o Controlador sabe quais os eventos de mudança de comportamento é capaz de executar, devendo reportar suas respectivas ocorrências ao Executor, caso estas afetem o comportamento temporal do documento.

Além das funções já mencionadas, cabe ao Executor manter o plano de execução, ajustando os tamanhos dos objetos sendo apresentados, através do envio de mensagens aos Controladores. Para controlar os planos, o Executor recebe dos Controladores a sinalização dos eventos. A Seção 5.3.3 descreve a estrutura interna do Executor.

Em uma versão futura, o sistema pretende incorporar funções de pré carregamento dos objetos, a fim de evitar que retardos na rede e nos próprios sistemas operacionais impossibilitem que ações ocorram no instante desejado, como por exemplo, o início de apresentação de um nó. No momento, a questão é decidir se o elemento responsável pela função de pré carregamento será o Executor ou o Controlador. No segundo caso, o Executor instanciaria antecipadamente os Controladores, deixando a cargo destes determinar se será feita, e em que momento, a pré busca.

5.3.2. Semântica e consistência de documentos NCM — Fase de Compilação

Esta seção apresenta uma aplicação da Técnica de Descrição Formal RT-LOTOS na especificação de documentos baseados no modelo conceitual NCM [SCSS97]. A descrição

RT-LOTOS, automaticamente traduzida para um DTA (Dynamic Timed Automaton) pela ferramenta RTL [CoOl94], poderá ser usada para realizar testes de consistência temporal e espacial da apresentação de documentos NCM. Além disso, o DTA gerado pode servir como estrutura base para geração do plano de exibição pelo executor. Portanto, o processo de tradução do NCM em RT-LOTOS e deste no DTA, produz resultados equivalentes aos do módulo compilador do ambiente de execução.

RT-LOTOS [CoOl94] é uma extensão temporal da Técnica de Descrição Formal LOTOS [BoBr87, ISO89a]. Uma vez que a semântica do RT-LOTOS é formalmente definida, na referência supracitada, ao descrever documentos NCM usando RT-LOTOS estamos, como consequência, fazendo uma definição da semântica do modelo no aspecto sincronismo temporal.

O NCM adota um esquema que combina características dos modelos baseados em eventos e em pontos de referência [StNa95] para definição do sincronismo temporal. A flexibilidade que o modelo fornece apresenta o efeito colateral de permitir que os autores especifiquem documentos inconsistentes. No escopo deste trabalho, um documento é dito ser inconsistente quando, por algum motivo, sua apresentação, uma vez iniciada, não termina.

Em [CoOl96] é descrito um modelo para especificação formal da sincronização temporal de documentos multimídia. Segundo a proposta, um usuário formaliza seu documento através da composição hierárquica de um conjunto de objetos apresentação e restrição, que são então traduzidos em uma especificação formal RT-LOTOS. Uma importante contribuição da referência [CoOl96] é que o modelo proposto para os documentos pode ser verificado com relação a duas propriedades: consistência temporal quantitativa e qualitativa. A especificação de um dado documento apresenta uma inconsistência qualitativa quando sua apresentação não termina, independentemente de valores particulares para as durações das apresentações de seus componentes. Um documento apresenta uma inconsistência quantitativa quando sua apresentação não termina para alguns valores particulares de duração da apresentação de seus componentes. Tanto a inconsistência qualitativa quanto a quantitativa podem ser detectadas com o emprego de análise de alcançabilidade clássica desenvolvida para RT-LOTOS [CoOl95].

Uma vez definido um método de tradução de uma especificação NCM para processos RT-LOTOS abre-se a possibilidade de usar as técnicas apresentadas em [CoOl96] para verificação da consistência de documentos NCM. Cabe ressaltar que a modelagem proposta em [CoOl96] permite verificar o que chamaremos de *consistência intrínseca*, ou seja, ela permite verificar problemas com a especificação de um documento que independem da plataforma onde ele será exibido. A referência [CSOS97] estende a abordagem apresentada em [CoOl96] propondo um mecanismo para verificação formal da consistência de um documento em uma determinada plataforma de exibição que, obviamente, pode ser configurável. Como um trabalho futuro, a versão ora apresentada da especificação do NCM será estendida para permitir também que sejam feitas verificações de consistência de plataforma.

5.3.2.1. Especificação RT-LOTOS do NCM

Do ponto de vista temporal, a apresentação dos nós, e consequentemente dos documentos NCM, é definida pela ocorrência de uma coleção de eventos. O sincronismo temporal, isto é, o alinhamento no tempo dos eventos, é definido através dos elos. O mapeamento de um documento NCM para RT-LOTOS resulta em uma especificação composta por três processos básicos: nó, elo e composição. O *processo nó* define a seqüência de eventos que ocorre ao longo da apresentação de um nó. Um *processo elo* define como os eventos ocorridos em nós distintos sincronizam-se entre si. No NCM, a entidade nó de composição define um documento (agrupamento lógico de nós e elos), o *processo composição* é definido pela composição paralela dos processos RT-LOTOS *nó* associados a cada um dos nós pertencentes à composição (para os filhos que são nós de composição, um processo composição é instanciado, recursivamente), e processos RT-LOTOS *elo* correspondentes aos elos definidos na composição. Os processos *elo* são sincronizados com os processos *nó* nos eventos de origem e de destino do elo.

O mapeamento do NCM em RT-LOTOS baseia-se nas seguintes definições:

Seja E um conjunto de eventos (nomes de eventos), $pSet = E \times E \times E \times 2^E \times 2^E$ e $lSet = 2^E$.

Definição 1. Uma *apresentação de um nó p* é definida como uma tupla de eventos p = (start, end, req_stop, Eint, Eext) $\in$ *pSet*, onde:

- *start* é um evento que caracteriza o início da apresentação
- *end* é um evento que caracteriza o fim da apresentação
- *req_stop* é o evento que força o fim da apresentação
- *Eint* é o conjunto dos eventos internos da apresentação
- *Eext* é o conjunto dos eventos externos da apresentação

Definição 2. Uma ocorrência de um elo *l* é definida como um conjunto de eventos $evtS \in lSet$.

Definição 3. A apresentação de um documento multimídia, no NCM, eqüivale a apresentação de uma composição. A apresentação de uma composição é definida por uma árvore formalizada pela tupla MD = (p_0, *c*, *C*, *A*), onde:

- $C \in$ pSet $\cup$ lSet é o conjunto dos nós da árvore que eqüivalem a apresentações de nós e ocorrências de elos contidos na composição.
- *A* é o conjunto de arcos da árvore.
- $c \in$ C é a apresentação de uma representação do nó de composição.[11]
- $p_0 \in$ pSet é a raiz da árvore.

A árvore caracteriza os relacionamentos hierárquicos entre as instâncias de objetos nó ou elo. Por definição, as apresentações de nós terminais e as ocorrências de elos são sempre associadas com folhas na árvore. As apresentações de composições são associadas com estruturas intermediárias na árvore.

A técnica de descrição formal RT-LOTOS é então empregada para especificar o comportamento dos nós da árvore. Com este propósito, é definida uma função RTLproc que associa a cada um dos elementos da árvore uma expressão de comportamento RT-LOTOS.

Definição 4. A função $RTLproc : C \rightarrow P$ é definida da seguinte maneira:

- *P* é um conjunto cujos elementos são expressões de comportamento RT-LOTOS válidas.

[11] Um exemplo de ferramenta que apresenta uma representação pictorial de uma composição é apresentada no Capítulo 4.

- Se *p* é uma apresentação de um nó terminal, e portanto folha na árvore definida pelo aninhamento de apresentações de composições (isto é, $p \in$ pSet e sonsOf(p) = ∅), então *RTLproc(p)* é uma instância de um processo RT-LOTOS pertencente a *Biblioteca-de-Nós*.
- Se *p* é uma apresentação de uma composição (isto é, $p \in$ pSet e sonsOf(p) ≠ ∅), então *RTLproc(p)* é uma instância do processo *Comp-p*.
- Se *l* é um elo (isto é, $l \in$ *lSet*), então *RTLproc(l)* é uma instância de um processo RT-LOTOS pertencente a *Biblioteca-de-Elos*.

Onde

- *Biblioteca-de-Nós* e *Biblioteca-de-Elos* são duas bibliotecas de processos RT-LOTOS genéricos e pré-definidos formalizando apresentações de nó terminais e ocorrências de elos. As bibliotecas de nós e elos são assunto das Seções 3.5.3.2.1.2 e 3.5.3.2.1.3, respectivamente.
- *Comp-p* é o processo genérico RT-LOTOS, associado a uma apresentação *p* de um nó de composição *c*, que executa a composição dos processos RT-LOTOS associados aos filhos de *c* (apresentações de nós terminais e de composição e ocorrências de elos contidos na composição) e à apresentação de uma representação da composição *c*. O processo Comp-p é definido na Seção 5.3.2.1.1.

A semântica do documento multimídia caracteriza os instantes do tempo nos quais os diferentes eventos do documento devem ocorrer. Uma vez que processos RT-LOTOS são associados com os nós da hierarquia de apresentações de composições aninhadas, *a semântica do documento multimídia é fornecida pela semântica do processo RT-LOTOS associado à composição raiz do documento*.

5.3.2.1.1. O Processo Comp-p

O processo *Comp_p*, mostrado na Figura 39, após ser instanciado, fica aguardando que um outro processo RT-LOTOS ofereça o evento *start* para sincronização. O evento *start*

recebe, opcionalmente, um descritor fornecido pelo processo (normalmente um elo) que comandou o início da apresentação da composição oferecendo um evento *start* para sincronização. Note que, ao ser instanciado, o nó recebe como parâmetro um descritor fornecido por sua composição pai ou pelo usuário. Após ocorrido o evento *start*, o processo *Comp_p* inicializa o processo *body_comp_p* passando como parâmetro o descritor que deve ser usado para criar a representação do nó, o descritor do elo (se *descritor_elo* > 0) ou o descritor definido pela composição pai (se *descritor_elo* = 0, isto é se nenhum descritor foi passado via elo).

```
process Comp_p [start, end, req_stop, Eext] (descritor_no: Descritor) : exit :=
hide start_b in
start ?descritor_elo: Descritor ;
(       (
            (
                (  start_b ; end ; exit )
                |[start_b, end]|
                (
                    [descritor_elo > 0] -> body_comp_p [start_b, end, Eext] (descritor_elo)
                    []
                    [descritor_elo = 0] -> body_comp_p [start_b, end, Eext] (descritor_no)
                )
            ) [> req_stop ; end ; exit
        )
        ||| Comp_p [start, end, req_stop, Eext] (descritor_no)
)
endproc
```

Figura 39: Processo RT-LOTOS Comp_p.

A qualquer momento o processo *Comp_p* pode ser encerrado, bastando para tal que um processo force uma sincronização com o evento *req_stop*. Como esse evento esta posicionado após o operador disrupt "[>", ele força o término do processo RT-LOTOS correspondente a instância da composição que está sendo apresentada. Ao final do processo, a construção "||| *Comp_p*" faz com que sempre que um evento *start* for oferecido para um processo *Comp_p*, simultaneamente a instanciação de um *body_comp_p*, seja criada uma nova instância do processo *Comp_p*, que fica então aguardando um novo *start*. Esta construção permite que sejam criadas diferentes representações, opcionalmente com descritores diferentes e em paralelo, para a composição. Note que a instanciação do processo *Comp_p* cria o correspondente a um objeto de dados para a composição *p*,

enquanto sua ativação, via evento *start*, cria um objeto de representação para a composição. Portanto, através deste artifício, a especificação RT-LOTOS captura os conceitos de objeto de dados e objeto de representação do NCM.

O processo RT-LOTOS *body_comp_p*, mostrado na Figura 40, é quem, na realidade, faz a composição dos processos que modelam a apresentação dos nós e elos definidos no nó de composição *p*. Nesse processo, a expressão RT-LOTOS "(no1[Eno1] ||| ... ||| noi[Enoi])" define a composição paralela processos nó contidos na composição. Nesta expressão:

- *noi* é um processo RT-LOTOS definido na Biblioteca-de-nós modelando a apresentação de um nó terminal contido na composição ou um processo Comp_i modelando a apresentação de um nó de composição i contido em p.
- *Enoi* é o conjunto de eventos externos ao nó i. Conjunto este, formado pelas extremidades de elos visíveis para o nó, eventos marcando o início e o fim da exibição do nó e um evento que força o término da apresentação do nó. Eno1 ∪ ... ∪ Enoi = Eext ∪ Eint.

Os processos nó são todos instanciados e ficam prontos para executar em qualquer ordem, inclusive simultaneamente.

```
process body_comp_p [start_b, end, Eext] (descritor_no: Descritor) : exit :=
  hide Eint, req_end in
(
  (   ( no1[Eno1](descritor_no1) ||| ... ||| noi[Enoi] (descritor_noi)   )
      |[Eelo]|
      (elo1[Eelo1] ||| ... ||| eloj[Eeloj] )
  )  [>  end ; exit
)
|[req_end, end]|
( req_end ; end ; exit )
endproc
```

Figura 40: Processo RT-LOTOS body_comp_p.

Os processos elo, a semelhança dos nós, também são instanciados de forma concorrente pela expressão RT-LOTOS "(elo1[Eelo1]||| ... |||eloi[Eeloj])". Esta expressão instancia os processos elo, correspondentes aos elos definidos na composição, que ficam prontos para disparar em qualquer ordem. Nesta expressão:

- *eloj* é um processo RT-LOTOS definido na Biblioteca-de-elos. Os processos elo são definidos de acordo com o numero de extremidades de destino e de origem e com a expressão que define a condição de disparo do elo, conforme explica a Seção 5.3.2.1.3.
- *Eeloj* é o conjunto de extremidades do elo j.

O operador RT-LOTOS "|[Eelo]|" faz com que os operandos (no caso, a composição paralela dos nós e a composição paralela dos elos) executem concorrentemente, porém sincronizados nos eventos pertencentes ao conjunto de eventos Eelo, onde:

- *Eelo* = { Eelo1 ∪ ... ∪ Eeloj } ⊆ { Eext ∪ Eint ∪ start ∪ req_end }.

No processo *body_comp_p*, o operador "|[Eelo]|" faz com que os nós cujos eventos iniciais são extremidade de destino dos elos só comecem sua apresentação após o disparo dos elos correspondentes, ou seja, quando os elos oferecem os eventos de destino para sincronização ocorre um *rendez-vous* — um *encontro* — entre os eventos, o que é a essência do atributo ponto de encontro dos elos NCM. Os elos disparam quando ocorrem encontros entre os eventos de origem, isto é, quando os eventos de origem são oferecidos para sincronização pelos processos nó onde estão definidos e; obviamente, suas condições de disparo são satisfeitas. As condições de disparo são definidas pelas expressões RT-LOTOS que definem o comportamento dos processos elo. Assim, quando uma composição é instanciada, os eventos *start* são oferecidos para sincronização pelos processos nó, caso eles sejam destino de algum elo, os respectivos processos nó ficam parados a espera do *rendez-vous*, que ocorre quando o respectivo elo dispara. Os que não são destino de elo, "ocorrem" e provocam dois efeitos: liberam os eventos subsequentes do processo nó para ocorrem a seu devido tempo, como explica a Seção 5.3.2.1.2 e, quando for o caso, encontram-se com as origens de processos elo.

Cabe ressaltar que o conjunto de eventos *Eelo* inclui os eventos *start* e *req_end*. O evento *start* pode ser usado como origem de um elo de start-up, isto é, um elo que dispara sempre que a apresentação da composição é iniciada (ou seja, quando o objeto representação correspondente a composição é iniciado pela ação *start_b* no processo *Comp_p*). O evento *req_end* pode ser usado como destino de um elo que comanda o término da apresentação

da composição. O disparo de um elo com o *req_end* como destino libera a ação *end* colocada após o operador disrupt "[>" que força o término do processo body_comp_p.

5.3.2.1.2. Os processos da Biblioteca-de-Nós

Os processos RT-LOTOS nó têm seu comportamento especificado com base no número de eventos externos (extremidades de elos ativos) previsíveis e imprevisíveis que possuem. Os processos nó são assim denominados *Noi_j*, onde o *i* é definido pelo número de eventos externos previsíveis do nó e o *j* pelo número de eventos externos imprevisíveis. A Figura 41 mostra a definição genérica dos processos Nó. A *Biblioteca-de-Nós* consiste então de um conjunto de processos RT-LOTOS: No0_0, No0_1, No1_0, No1_1, No2_1, No1_2 etc.

O processo *Noi_j* é semelhante ao processo *Comp_p*. Sua finalidade é instanciar o processo *body_Noi_j* com os parâmetros definidos no descritor fornecido pelo elo que ativou o processo (capturados no evento *start*) ou pelo nó de composição que contém o processo nó. Note que a construção " ||| Noi_j [start, end, req_stop, e_1, … , e_{i+j}] (descritor_no, d_1, l_1, … , d_{i+j}, l_{i+j})" ao final do processo, torna possível a ativação de várias instâncias (representações do nó). O *req_stop*, como nos processos *Comp_p*, tem a função de permitir uma parada forçada do nó (no NCM esta facilidade simula a execução de ações stop).

O processo *body_Noi_j* funciona da seguinte forma: uma vez iniciado o processo começa a contagem dos delays associados aos eventos do nó. Para cada evento, um delay pode ser definido através do parâmetro d_k. Adicionalmente, o parâmetro l_k define um intervalo de tempo, após decorrido o delay, onde o evento pode ocorrer. Um evento previsível *i* necessariamente ocorre no intervalo $[d_i,d_i+l_i]$. Um evento imprevisível *j* pode ou não ocorrer no intervalo $[d_j,d_j+l_j]$. Quando o evento *end* ocorre (no intervalo [dno,dno+lno]), o processo *body_Noi_j* é encerrado. Portanto, com o emprego do parâmetro l, a especificação RT-LOTOS captura a flexibilidade na definição dos eventos NCM.

```
process Noi_j [start, end, req_stop, e1, … ,ei+j] (descritor_no: Descritor,
                                                   d1, l1, … , di+j, li+j:nat): exit :=
start ?descritor_elo:Descritor;
(   (
      (
         [delo > 0] -> body_ Noi_j [end, e1, … , ei+j] (descritor_elo, d1, l1, … , di+j, li+j)
         []
         [delo = 0] -> body_ Noi_j [end, e1, … , ei+j] (descritor_no, d1, l1, … , di+j, li+j)
      )  [>  req_stop ; end ; exit
   )  |||   Noi_j [start, end, req_stop, e1, … , ei+j] (descritor_no, d1, l1, … , di+j, li+j)
)
endproc

process body_noi_j [end, e1, … , ei+j] (dno, lno, d1, l1, … , di+j, li+j:nat): exit :=
hide end_no in
(
        (  ( delay (d1) e1{l1} ; exit )   ||| … ||| ( delay (di) ei{li} ; exit )
           |||  anchor [ei+1] (d i+1,l i+1)  |||  anchor [ei+j] (di+j,li+j)
           |||   ( delay (dno) end{lno} ; exit )
        )  [> end_no ; exit
)
|[end_no, end]|
(   end ; end_no ; exit   )
endproc

process anchor [user] (wait,window:nat) : exit :=
  delay(wait) (user{window}; exit [] delay(window) exit)
endproc
```

Figura 41: Processo RT-LOTOS Noi_j.

5.3.2.1.3. Os processos da Biblioteca-de-Elos

O nome dos processos da Biblioteca-de-elos é definido da seguinte forma: Elo*m_op_n*, onde *m* é o número de eventos de origem do elo, *n* é o número de eventos de destino e *op* é o operador cuja semântica é expressa pela expressão RT-LOTOS definida no corpo do processo. As Figuras 5, 6 e 7 mostram alguns exemplos de processos elo. A *Bibliote-de-Elos* é formada por um conjunto de processos semelhantes expressando a semântica das regras de disparo necessárias para modelar os elos contidos em um documento, em uma base privada ou em uma hiperbase pública.

Os processos RT-LOTOS da Figura 42 definem a semântica de elos 1 para n simples. O *v* antes do número de eventos de destino indica que o destino espera receber um descritor

quando o elo for disparado. Obviamente, pode ser passado um descritor nulo (parâmetro *d* igual a 0). O funcionamento desses elos é muito simples: quando o elo é instanciado, seu evento de origem é oferecido para sincronização; quando o processo nó de origem disponibiliza o referido evento para sincronização ocorre o encontro. Após o encontro, opcionalmente espera-se um delay e, também opcionalmente, espera-se outro delay (que pode ser diferente para cada um dos eventos de destino). Em seguida, o evento de destino é oferecido para sincronização.

```
process elo1_1 [e1, e2] (d: nat) : noexit :=
   e1; delay (d) e2{0}; elo1_1 [e1,e2] (d)
endproc

process elo1_v1 [e1, e2] (d: nat, descritor_dest: Descritor) : noexit :=
   e1; delay (d) e2{0} ! descritor_dest ; elo1_v1 [e1,e2] (d, descritor_dest)
endproc


process elo1_v4 [e1, e2, e3, e4, e5] (d1, d2, d3, d4, d5, descritor_dest2,
                                      descritor_dest3, descritor_dest4,
                                      descritor_dest5: Descritor) : exit :=
        e1 ; delay(d1) (
                        (delay(d2) e2{0} ! descritor_dest2 ; exit )
                        |||
                        (delay(d3) e3{0} ! descritor_dest3 ; exit )
                        |||
                        (delay(d4) e4{0} ! descritor_dest4 ; exit )
                        |||
                        (delay(d5) e5{0} ! descritor_dest5 ; exit )
                        )
         >> elo1_v4 [e1, e2, e3, e4, e5] (d1, d2, d3, d4, d5, , descritor_dest2,
                                          descritor_dest3, descritor_dest4,
                                          descritor_dest5)
endproc
```

Figura 42: Processos RT-LOTOS elo1_n.

Os processos RT-LOTOS mostrados na Figura 43 são outros exemplos de elos. O primeiro especifica a semântica de um elo *and*. A expressão RT-LOTOS deste processo determina que o evento de destino só fique disponível para sincronização após ocorridos os dois eventos de origem, respeitados os delays após a ocorrência das origens e antes da ocorrência dos destinos. O segundo processo define a semântica de um elo *or*. Nesse caso,

a expressão RT-LOTOS determina que o evento de destino é oferecido para sincronização após ocorrido um rendez-vous com qualquer um dos eventos de origem (ou mais de um caso ocorram simultaneamente).

```
process elo2_and_v1 [e1, e2, e3]
                        (d1: nat, d2: nat, d3: nat, descritor_dest: Descritor): noexit :=
(  ( e1 ; delay(d1) exit)  ||| ( e2  ; delay(d2) exit )  )   >>
        delay(d3) e3{0} ! descritor_dest;
        elo2_and_v1 [e1, e2, e3] (d1, d2, d3, descritor_dest)
endproc

process elo2_or_1 [e1, e2, e3] (d1, d2, d3: nat) : noexit :=
(   (  e1 ; delay(d1) exit  )    []   (  e2 ; delay(d2) exit ) )   >>  delay(d3) e3{0}  ;
        elo2_or_1 [e1, e2, e3] (d1, d2, d3)
endproc
```

Figura 43: Processos RT-LOTOS elo_and_ e elo_or_.

Por fim, o processo RT-LOTOS da Figura 44, mostra um processo RT-LOTOS correspondente a um elo com a expressão “(e1 and e2) or (e3 and e4)” em seu ponto de encontro.

```
process elo2_and_or_2_and_1 [e1, e2, e3,e4,e5] (d1, d2, d3,d4,d5: nat) : noexit :=
(   ( ( e1 ; delay(d1) exit)  ||| ( e2  ; delay(d2) exit ) )
   []
   ( ( e3 ; delay(d3) exit)  ||| ( e4  ; delay(d4) exit )  )
)   >>  delay(d5) e5{0}  ;
elo2_and_or_2_and_1 [e1, e2, e3,e4,e5] (d1, d2, d3,d4,d5)
endproc
```

Figura 44: Processo RT-LOTO elo_and_or_and.

5.3.2.2. Exemplo de especificação de um documento NCM usando RT-LOTOS

Esta seção exemplifica a formalização de um documento NCM usando a técnica de descrição formal RT-LOTOS. A estrutura do documento exemplo é mostrada na Figura 45.

A apresentação do documento (nó de composição A) começa com a exibição de uma representação pictorial do próprio nó A (o browser de estrutura mostra um mapa com a estrutura do documento, um desenho semelhante ao da Figura 1). O início da exibição do nó A (ocorrência do evento Ae1) provoca o disparo do elo l_4 e como consequência são iniciadas as apresentações dos nós D (perspectiva AD) e F (perspectiva AF). Ao longo da exibição do nó A, se o usuário selecionar uma região do mapa uma outra representação, ou

versão, do nó D (perspectiva ABD) é exibida. O final da apresentação da representação do nó ABD (evento ABDe2) faz com que o elo l1 dispare e o no C (perspectiva ABC) seja exibido.

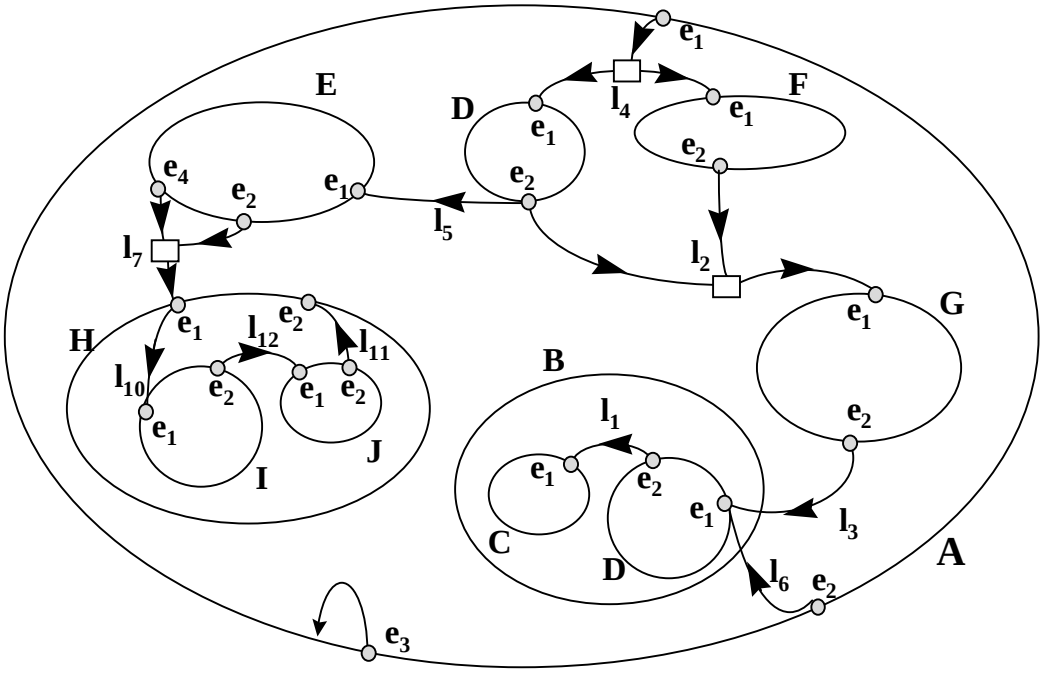


Figura 45: Estrutura do documento

Durante a exibição do nó AD, se o usuário selecionar uma âncora (evento ADe2), o elo l_5 dispara provocando o início da exibição do nó E (perspectiva AE). Depois que o nó F (perspectiva AF) acaba de ser exibido, caso o usuário tenha interagido durante a exibição de AD, isto é, se os eventos AFe2 e ADe2 tenham ambos ocorridos em qualquer ordem, a condição de disparo do elo l_2 é satisfeita, porém o evento de destino AGe1 só deve ocorrer 10 segundos após. Nesse caso provocando o início da exibição do nó G (perspectiva AG). Ao final da exibição do nó G o elo l_3 dispara e uma nova exibição do nó D (perspectiva ABD) é iniciada. Após o final desta nova versão do nó D o elo l_1 dispara e o nó C (perspectiva ABC) é exibido também.

Ao longo da exibição do nó AE, caso ocorra o evento AEe4 (interação com usuário) ou o evento AEe2 (evento previsível), uma representação da composição AH começa a ser exibida. O início da apresentação de AH implica no disparo do elo l_9 que implica na exibição do nó AHI. Durante a exibição do nó AHI, caso ocorra o evento imprevisível AHIe2

(interação com usuário), o nó AHJ é exibido. Ao final da exibição de AHJ, o evento AHJe2 força o término da representação da composição AH que contém a representação do nó AHI que terminou. O final da apresentação da representação pictorial do nó A, marcado pelo evento Ae3 força o final da apresentação da composição e consequentemente do documento.

Para testar o mapeamento de NCM em RT-LOTOS, o documento exemplo foi especificado aplicando-se o método descrito na Seção 5.3.2.1. Uma listagem do código da especificação é encontrada no Anexo B. O Anexo B inclui também uma listagem com o resultado da simulação da especificação RT-LOTOS do documento NCM exemplo. A simulação foi feita com a ferramenta RTL (RT-LOTOS Laboratory) [CoOl94] desenvolvida no LAAS-CNRS. A Figura 46 mostra graficamente o resultado da simulação no período de 190 a 240 segundos, que corresponde a uma exibição completa do documento.

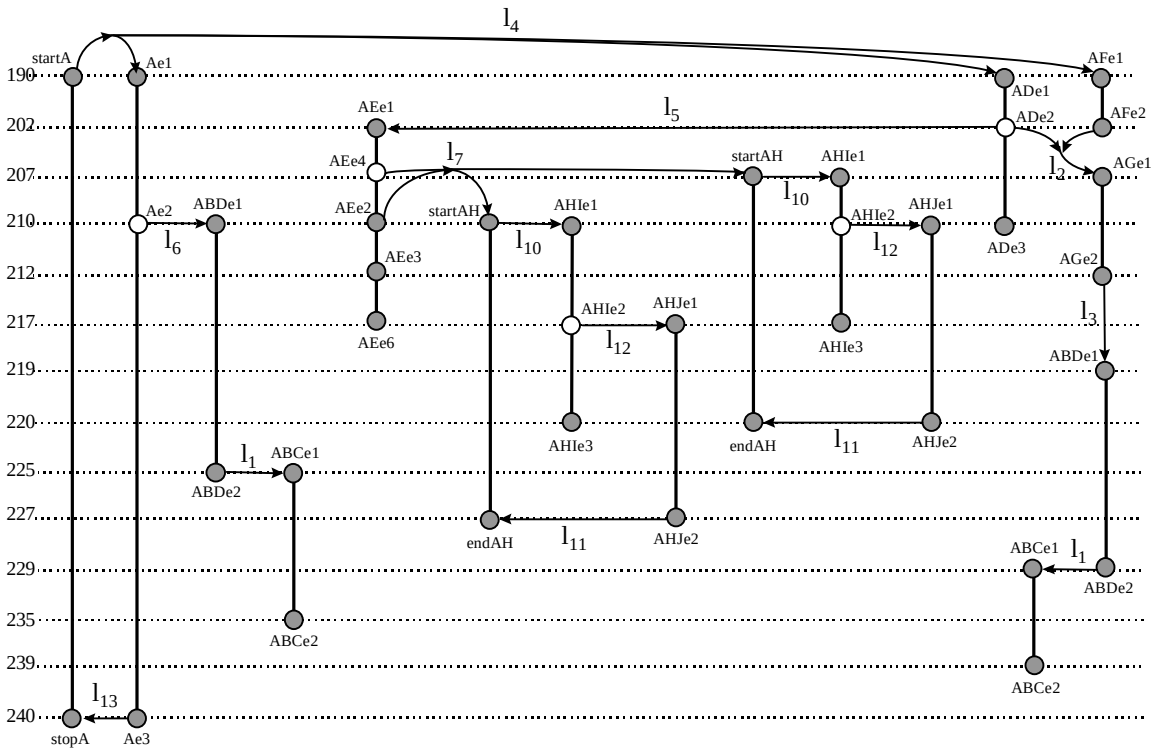


Figura 46: Estrutura da Apresentação do documento.

Este exemplo mostra claramente a viabilidade da aplicação da técnica apresentada em [CoOl96] para verificar a consistência de documentos NCM, bastando para tal integrar a

ferramenta RTL ao ambiente de edição e execução do sistema HyperProp. Como já mencionado, o método de verificação consiste em detectar tanto a inconsistência qualitativa quanto a quantitativa com o emprego de análise de alcançabilidade clássica em um DTA (Dynamic Timed Automaton) que a ferramenta RTL gera automaticamente a partir da especificação RT-LOTOS [CoOl95].

Como já diversas vezes mencionado, um outro trabalho futuro, aberto pelo mapeamento ora descrito, é o estudo da viabilidade do uso da estrutura gerada pela ferramenta RTL, o DTA, como estrutura base para geração dos planos de exibição pelo módulo responsável pelo controle da execução de documentos no sistema HyperProp, como será discutido na próxima seção.

5.3.3. Modelo de Dados das Cadeias Temporais

As referências [Rodr97, RoSS97a] propõem que o plano de execução seja modelado como uma rede de Petri temporizada, que é uma pequena extensão à TSPN [DiSé94]. A extensão consiste em associar aos arcos de saída de um lugar não só a duração mínima, ótima e máxima dos eventos (a tupla {min, ot, max}), mas também o tempo estimado para a duração, calculado pelo Compilador (tem-se assim a tupla {min, ot, esp, max}). Note que é feita uma passagem de um modelo de mais alto nível de abstração, o NCM, para a TSPN. A noção de estado global permite o fácil acompanhamento do comportamento dinâmico de um sistema modelado com TSPN. [WSSD96], ao contrário, utiliza a TSPN (na realidade uma extensão, I-HTSPN) no ambiente de autoria e o modelo MHEG no ambiente de execução. Uma crítica detalhada desse procedimento é encontrada em [RoSS97b].

A escolha de um modelo de fluxo de dados temporais foi feita após observar que a estrutura linear de uma *timeline* é inapropriada tanto para edição, conforme amplamente descrito na literatura, como também 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.

Nas TSPNs, eventos de seleção, indicando a interação com o usuário, são facilmente representados por arcos e lugares associados, permitindo que a concatenação das cadeias temporais ocorra naturalmente pela execução da rede de Petri. As cadeias que possuam relação com o lugar representando o evento terão, então, os seus tempos esperados, a serem associados aos arcos, calculados pelo Compilador, ou recalculados pelo Executor em tempo de exibição.

Eventos de apresentação são comparados 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 as consistências temporais e espaciais. O sistema ainda não implementa esse tipo de ajuste, sendo previsto o estudo de algoritmos para fazê-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 [Rodr97], e exigem o mesmo mecanismo comentado no parágrafo imediatamente anterior.

A Seção 5.3.2 mostrou uma descrição das entidades do NCM que capturam a especificação do sincronismo temporal dos documentos feita com o uso da Técnica de Descrição Formal RT-LOTOS [CoOl95]. A mesma seção apresentou os resultados do uso da ferramenta RTL (RT-LOTOS Laboratory) [CoOl94] para simular a apresentação de documentos NCM. Conforme explicado, a ferramenta gera um DTA (Dynamic Timed Automaton) a partir da especificação RT-LOTOS. O DTA foi então utilizado como estrutura base na realização das simulações, que produzem como resultado uma lista de eventos e seus respectivos tempos de ocorrência, ou seja um plano de exibição simulado. Cabe ressaltar que o DTA é a estrutura de dados base para a geração dos planos de exibição das apresentações de

documentos simuladas. Na arquitetura do ambiente de execução do HyperProp esta estrutura poderia ser então utilizada como saída do compilador e entrada para o executor, servindo de base para que este, em tempo de execução, gerasse os planos de exibição. Assim, a descrição do NCM em RT-LOTOS e o uso da ferramenta RTL, abriu uma linha de investigação bastante promissora que consiste em verificar a possibilidade de incorporar a ferramenta RTL ao ambiente de edição e execução do HyperProp. Neste caso a ferramenta, em conjunto com um tradutor NCM $\rightarrow$ RT-LOTOS, faria o papel do módulo compilador do ambiente, gerando a estrutura de dados base para o executor HyperProp, no caso um DTA.

5.3.4. Descrição das Mensagens Trocadas entre os Elementos do Sistema

Existem três grupos de mensagens salientadas na Figura 38. As mensagens trocadas entre o Compilador/Executor e o Servidor NCM (m1), as mensagens trocadas entre os Controladores e o Servidor NCM (m2), e as mensagens trocadas entre o Executor e os Controladores (m3). A interface 1 é uma interface externa ao ambiente de execução, enquanto a interface 2 é uma interface interna ao mesmo.

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 para a geração do plano de execução), e as requisições dos descritores pelo Executor.

No conjunto m2 estão as mensagens para busca dos objetos de dados e descritores que constituem os objetos de representação. A proposta é que uma interface para busca dos conteúdos, como a apresentada em [CSCS96] seja utilizada. A implementação atual traz os dados sem qualquer controle da qualidade de serviço.

Finalmente, no conjunto m3 estão as mensagens trocadas entre o Executor e os Controladores. Do Controlador para o Executor são enviadas mensagens de sinalização descrevendo os eventos ocorridos. No sentido inverso, existe uma mensagem para criação do Controlador, que passa como parâmetros o objeto de dados (ou o seu identificador, conforme mencionado na Seção 5.3.1), a identificação do descritor, a lista de eventos que

devem ser sinalizados e o tempo que falta para o comando de início de apresentação ser enviado. Também nesse sentido estão os comandos para iniciar, parar e abortar a exibição de um nó. Futuramente, serão acrescidas mensagens com os comandos de suspender uma exibição, e comandos de get e set para, respectivamente, consultar ou armazenar um valor em um atributo do objeto sendo exibido. O adiantamento e retrocesso de uma exibição será feito através de comandos de set no atributo velocidade do objeto de representação. A definição das diversas interfaces, em detalhes, pode ser encontrada em [Rodr97].

Capítulo 6

Trabalhos Relacionados

Existem inúmeras propostas de modelos para documentos multimídia. Com base no exposto no Capítulo 2 e em outros estudos [BlSt96, PéLi96], conclui-se que os modelos que apresentam maior poder de expressão são os baseados nos conceitos de composições, pontos de referência e eventos. Daí a escolha desta abordagem para o tratamento do problema de sincronização no NCM e nos trabalhos relacionados descritos ao longo deste capítulo para efeito de comparação com o NCM.

No Capítulo 2 foram feitas inúmeras referências a modelos que adotam outros paradigmas para especificação do sincronismo temporal: eixos de tempo, intervalos, scripts, redes de Petri etc. que não serão aqui comentados.

Após a descrição de alguns exemplos de modelos, o esquema de classificação apresentado no Capítulo 2 é empregado para uma comparação dos modelos apresentados.

6.1. Modelos e Sistemas

6.1.1. Mode

O sistema Mode (Multimedia Objects in a Distributed Environment) [BlHL92] baseia-se nas classes de objetos: de informação, de apresentação e de transporte.

Os objetos que pertencem à classe informação (chamados objetos informação) são as unidades básicas de informação do ambiente. Para apresentar um objeto informação são gerados objetos da classe apresentação (objetos apresentação). Objetos apresentação contêm todos os dados e métodos necessários para sua exibição; eles podem existir de forma independente dos objetos informação. Se um objeto deve ser apresentado

remotamente, tanto o objeto informação quanto os objetos apresentação apropriados devem ser transportados para o nó remoto.

Quando um objeto informação ou apresentação precisa ser transportado, ele é convertido em um objeto da classe transporte (objeto transporte) enquanto durar o transporte, e é desconvertido pelo receptor. Objetos transporte consistem de todos os métodos e dados necessários para transportar, comprimir e descomprimir um objeto.

Uma apresentação multimídia sincronizada (objeto multimídia) pode ser composta de objetos monomídia, de objetos multimídia aninhados, e de informação de sincronização, o que permite a criação de hierarquias de sincronização.

Para suportar as diferentes formas de apresentação de um objeto informação foi definido o objeto básico, que consiste de uma referência para um objeto informação e uma coleção de atributos de apresentação. Com o emprego desse mecanismo, 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.

Existem objetos básicos dinâmicos e estáticos. A apresentação de um objeto básico dinâmico é composta de uma seqüência de objetos apresentação (por exemplo quadros de uma seqüência de vídeo), apresentados periodicamente. A sincronização desta seqüência monomídia é denominada sincronização intra-objeto. O índice de cada objeto apresentação é denominado ponto de referência. A apresentação de um objeto básico estático (um objeto cuja mídia não depende do tempo, como um pedaço de texto ou uma figura) possui apenas dois pontos de referência, o início e o fim da apresentação.

A sincronização entre apresentações de objetos básicos (denominada sincronização inter-objetos) é definida usando pontos de referência. A descrição de um ponto de referência junto com o objeto básico correspondente é denominada elemento de sincronização, denotada na forma de um BasicObject-ReferencePoint. Dois ou mais elementos de sincronização podem ser combinados em um ponto de sincronização. Nenhum objeto pode

continuar a sua exibição até que todos os objetos associados ao ponto de sincronização atinjam os seus respectivos elementos de sincronização. A definição completa da sincronização inter-objetos é descrita pela lista de todos os pontos de sincronização.

Dois tipos especiais de objetos são adicionalmente definidos: temporizadores e objetos interativos. Os temporizadores podem ser considerados como objetos dinâmicos virtuais onde os pontos de referência são milisegundos. Temporizadores são usados para modelar retardos absolutos, por exemplo, uma apresentação de slides onde os slides mudam a cada 10 segundos. Objetos interativos modelam a comunicação com os usuários, são sincronizados apenas em seu início ou fim, e representam o tipo mais importante de objetos com duração de apresentação imprevisível. Os objetos interativos podem invocar métodos para iniciar, suspender, retomar ou encerrar a exibição de um objeto básico ou multimídia, simular o avanço rápido ou retorno destes objetos, ou desviar a apresentação para um ponto específico em um objeto multimídia.

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 sincronização. O *synchronizer* também implementa meta-diretivas geradas pela interação do usuário com os objetos.

O sistema é limitado com relação à definição de relacionamentos baseados no conceito hipermídia (interatividade do usuário de maneira completamente imprevisível) pois a captura de interações com o usuário é restrita aos objetos interativos (botões) que não são vinculados ao conteúdo dos objetos monomídia. Portanto, o sistema não implementa o conceito de âncora em toda a sua extensão. O sistema também é restrito no aspecto estruturação lógica pois os objetos multimídia, apesar de permitirem a construção de

hierarquias de sincronização, apresentam uma granularidade grossa, não permitindo a oferecendo apenas um ponto de entrada na composição.

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 sincronização aliados à flexibilidade na especificação permitem um ajuste da taxa de exibição dos objetos na fase de execução.

6.1.2. Firefly

O sistema Firefly [BuZe92, BuZe93a, BuZe93b] divide a especificação dos documento em dois níveis: especificação a nível de mídia e especificação a nível de documento.

A *especificação a nível de mídia* descreve o comportamento temporal da apresentação dos segmentos de informação que compõem os documentos, como trechos de vídeo, gravações de áudio, arquivos de texto, programas e documentos multimídia. Nesse nível, a descrição de um segmento individual é denominada *media item* e inclui: eventos, durações, custos e uma referência para o segmento de informação "concreto" descrito pelo media item.

Os *Eventos* representam pontos no tempo ao longo da apresentação do media item. Todos os media itens possuem um evento inicial e um evento final que marcam o início e o fim da apresentação do media item. Além desses, um media item pode possuir zero ou mais eventos internos especificados pelo autor. Um evento é previsível ou imprevisível. Um evento previsível marca pontos no tempo que podem ser determinados a priori, como o momento no qual um quadro específico em um vídeo será exibido. Deve-se ressaltar, entretanto, que os instantes de tempo marcados por eventos previsíveis só podem ser determinados sob condições ideais e não levam em conta retardos imprevisíveis, como os devidos a carga excessiva na estação onde será apresentado o documento. Eventos imprevisíveis marcam pontos no tempo, durante a apresentação dos segmentos, que não podem ser determinados a priori. São exemplos de eventos imprevisíveis o momento que um programa em execução atinge um determinado estado ou o momento no qual o usuário seleciona um objeto que está sendo exibido na tela da estação de trabalho.

Uma *duração* especifica o tempo decorrido entre dois eventos temporalmente adjacentes no mesmo media item. A duração entre dois eventos adjacentes e_i e e_{i+1}, é representada pelos valores: $mimDuration_{i,i+1}$ (valor mínimo para o tempo decorrido entre os dois eventos), $optDuration_{i,i+1}$(valor ótimo, ou desejado, para o tempo decorrido) e $maxDuration_{i,i+1}$(valor máximo). Alguns segmentos, como textos e imagens estáticas, normalmente não possuem a noção de tempo em suas especificações. Os autores devem então utilizar a estrutura media item para impor um comportamento temporal a tais segmentos. Os media itens resultantes da imposição poderão ter durações previsíveis ou imprevisíveis. Uma duração é previsível se seu comprimento pode ser determinado a priori. Exemplos incluem trechos de áudio, vídeo e animações. Tipicamente, o comprimento de uma duração previsível é uma função da quantidade de dados e da taxa segundo a qual as unidades de dados são apresentadas. Outro exemplo de duração previsível é a apresentação de um segmento de imagem ou texto por um intervalo de tempo específico. Por outro lado, o comprimento de uma duração imprevisível só pode ser determinado quando a apresentação do segmento acaba. Segmentos com durações imprevisíveis incluem programas com tempo de execução imprevisíveis, como buscas em bancos de dados ou simulações; comunicações em tempo real, como conversações telefônicas; e apresentações de textos ou imagens que devem ser explicitamente encerradas por um comando do usuário.

O modelo permite a associação de *custos* a durações ajustáveis para que possam ser associados níveis de preferência aos possíveis valores da duração. A seleção de uma duração diferente da ótima reduz a qualidade da apresentação do media item; o custo pode ser visto como uma medida objetiva dessa degradação. O modelo permite a associação de dois custos à duração entre dois eventos adjacentes e_i e e_{i+1}: $costShrink_{i+1}$ (custo da diminuição da duração através da seleção de um valor entre o ótimo e o mínimo) e costStretch (custo do aumento da duração pela seleção de um valor entre o ótimo e o máximo).

A *especificação a nível de documento* descreve o comportamento temporal da apresentação de um documento e consiste de media itens, restrições temporais, operações, durações e custos.

A especificação do documento inclui media itens que descrevem os segmentos que compõem o documento. Um media item pode aparecer em mais de um documento ou mais de uma vez no mesmo documento. Cada ocorrência de um media item é referenciada como uma *instância* do media item.

Restrições temporais representam explicitamente os relacionamentos temporais entre os eventos do documento especificando a ordenação temporal de pares de eventos em um ou mais media itens. Cada restrição temporal associa um evento fonte a um evento destino. As restrições temporais são classificadas em: igualdades temporais (como as que exigem que os eventos fonte e destino ocorram simultaneamente ou que o fonte preceda o destino de um valor de tempo fixo) e desigualdades temporais (como as exigem que o evento fonte preceda o destino por um valor de tempo não especificado, ou que o fonte preceda o destino por no mínimo 15 s e no máximo 20 s, por exemplo.

Os autores podem associar *operações* aos eventos para controlar o comportamento da apresentação de um media item. As operações são divididas em dois grupos: operações que modificam o tempo, como aumentar-velocidade-exibição ou suspender-exibição; e operações que não modificam o tempo da apresentação dos documentos, como aumenta-volume, diminui-janela etc. A especificação das operação no nível do documento torna possível aos autores controlar de forma diferenciada as várias instâncias de um media item.

Os autores podem opcionalmente especificar durações e custos na especificação a nível de documento. Essas durações e custos são então associadas com instâncias específicas de um media item, permitindo que o comportamento de cada instância seja personalizado.

O sistema Firefly oferece duas ferramentas para realizar as tarefas de criação, visualização e edição de especificações temporais: um visualizador de media itens e um editor interativo de documentos.

A Figura 47 mostra dois visualizadores da mídia áudio. Na parte superior da figura aparece o editor de áudio TiogaVoice. A parte inferior mostra o visualizador temporal de media itens.

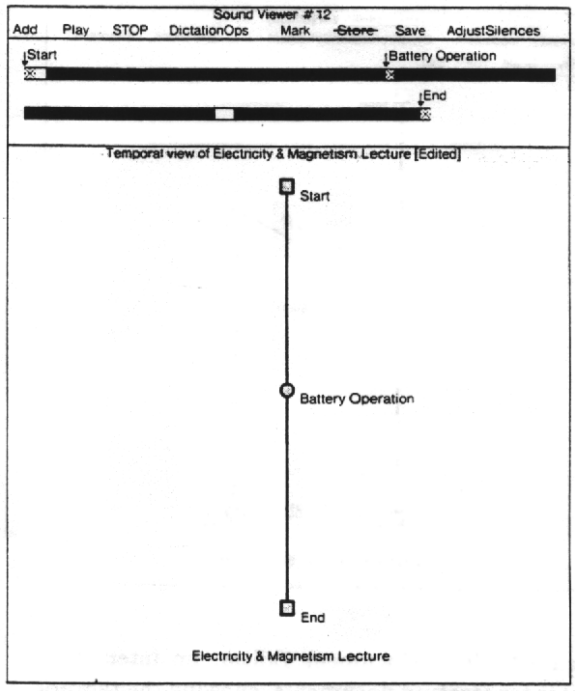


Figura 47: Visualizador de item de mídia do sistema Firefly.

No exemplo da Figura 47 o visualizador de mídia mostra um media item áudio chamado "Electricity & Magnetism Lecture". Os eventos de início e de fim de exibição são representados por nós retangulares que possuem os nomes Start e End. O evento interno "Battery Operation" é representado por um nó circular posicionado entre os eventos de início e fim do media item.

Tanto o visualizador de media item, quanto o editor interativo de documentos fornecem uma visualização que utiliza uma notação em grafos. Segundo essa notação, os nós retangulares representam os eventos de início e de fim de um media item, enquanto os nós circulares representam eventos internos. As linhas representam o tempo decorrido entre dois eventos, onde seu comprimento é proporcional à duração entre os eventos.

A interface gráfica da outra ferramenta do sistema Firefly, o editor interativo de documentos, é mostrada na Figura 48.

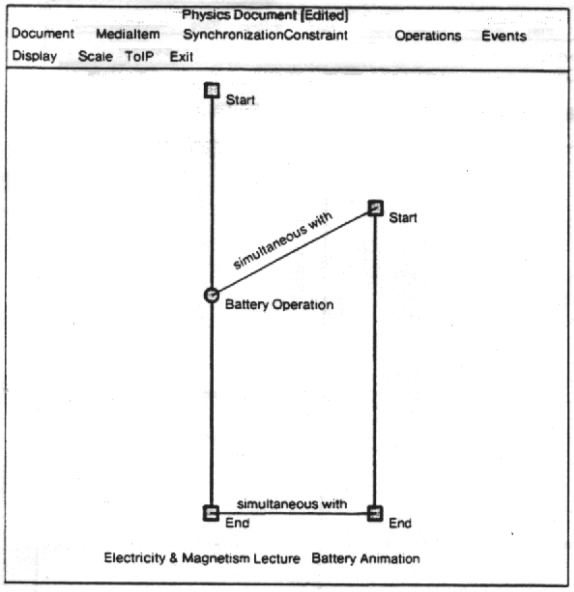


Figura 48: Editor interativo de documentos do sistema Firefly

O editor, ilustrado na Figura 48, possui duas regiões de interface. A região de menu, que contém funções para criação e edição de documentos, e a região de edição gráfica, que contém uma visão temporal do documento na notação de grafos.

A Figura 48 mostra um exemplo de documento que contém dois media itens: um áudio intitulado “Electricity & Magnetism Lecture”, e um vídeo intitulado “Battery Animation”, que contém uma animação. Para exibir a animação durante a exibição do áudio, são utilizados dois relacionamentos temporais de sincronização. O primeiro relacionamento, de igualdade temporal, conecta o evento “Battery Operation” do áudio com o evento de início (Start) da animação, especificando que eles devem ocorrer ao mesmo tempo. O segundo relacionamento, também de igualdade temporal, conecta o evento de fim do áudio (End) com o evento de fim (End) da animação, especificando que eles devem ocorrer ao mesmo tempo, ou seja, o áudio e a animação devem terminar juntos.

Na interface do editor, um media item pode ser inserido em um documento ativando a opção de menu. Eventos assíncronos também podem ser definidos dentro dos media itens, através da interface com o editor do sistema Firefly.

A apresentação dos documentos definidos no sistema é controlada por um formatador que apresenta arquitetura idêntica à ilustrada no Capítulo 5, 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 eventos, previsíveis antes do início da exibição, que possuam algum encadeamento temporal. As cadeias temporais auxiliares são encadeamentos de eventos previsíveis, a partir de um evento imprevisível.

O compilador Firefly recebe a especificação de apresentação de um documento e obtém a duração entre cada dois pontos de sincronização temporais adjacentes em cada segmento de mídia a ser apresentado. 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. O compilador também recebe uma lista de operações, que especificam mudanças de comportamento aplicadas a cada ponto de sincronização. O sistema Firefly não permite o aninhamento de composições, e mesmo a composição definida pelo agrupamento de media itens em um documento é muito restrita.

Para construir as cadeias temporais, o compilador aplica um algoritmo que obtêm as componentes conexas do documento. Inicialmente, cada evento é colocado como uma componente conexa separada. Dois eventos 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 evento 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 eventos. 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 evento 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 eventos aos quais o algoritmo não conseguiu atribuir um instante esperado e os conflitos de eventos

(eventos de um mesmo segmento de mídia, que receberam o mesmo instante esperado de ocorrência).

Por fim, o compilador, em cada componente conexa, visita os eventos 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 e sinalizar a ocorrência de eventos imprevisíveis. A Figura 49 ilustra a arquitetura do executor do Firefly.

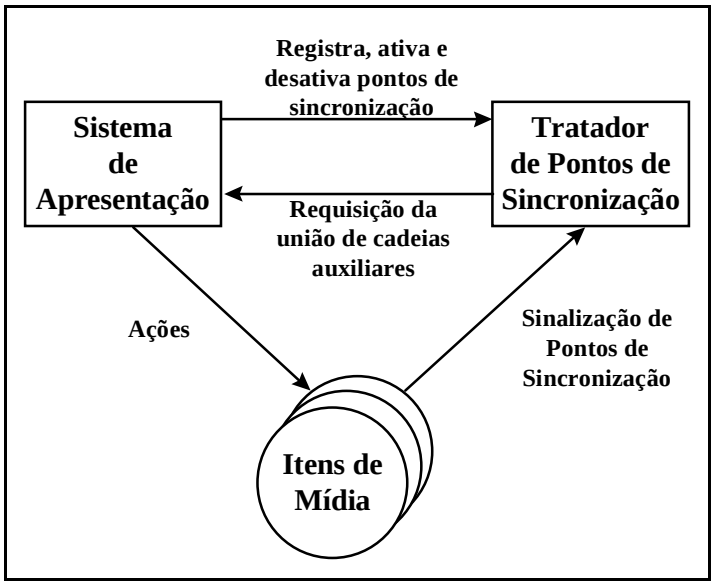


Figura 49: Executor do Firefly.

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 eventos previsíveis.

O sistema também permite que qualquer evento imprevisível em um segmento de mídia seja habilitado ou inibido. Quando um segmento de mídia detecta a ocorrência de um evento imprevisível, ele o notifica ao tratador de eventos, que mantém uma lista dos eventos habilitados e inibidos. Se o evento notificado estiver habilitado, ele é passado ao sistema de apresentação, para que a cadeia temporal auxiliar com início nesse evento seja unida à cadeia principal. Antes da união, o sistema de apresentação calcula tempos absolutos para os eventos da cadeia auxiliar, que originalmente possuíam valores relativos.

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.

6.1.3. 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 (Hierarchical Time Stream Petri Net) [SéSW95], definido com o objetivo de melhorar a estruturação durante a fase de autoria dos documentos hipermídia. A idéia básica do modelo 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 documento. 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, ou seja, a granularidade das composições é grossa.

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 pelo modelo anterior.

Na I-HTSPN 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 I-HTSPN, 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 50 apresenta o esquema de funcionamento da ferramenta I-HTSPN.

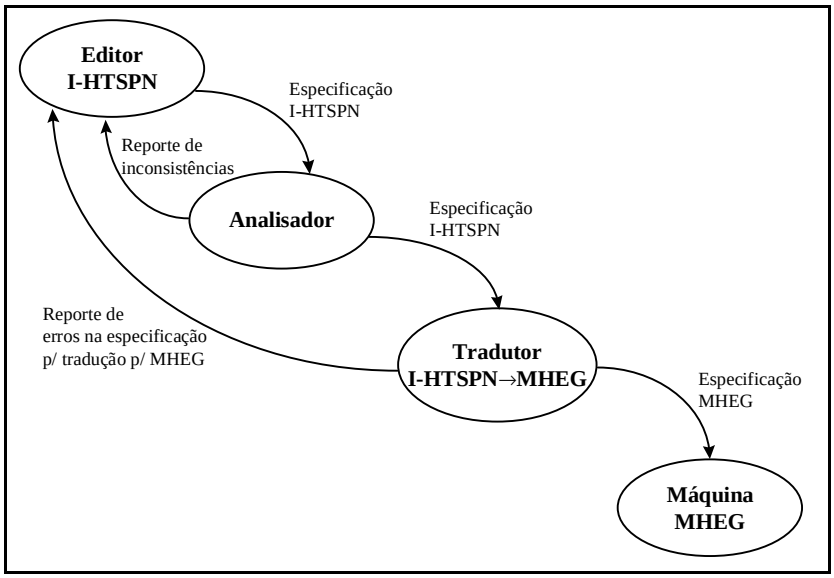


Figura 50: Esquema de funcionamento da ferramenta I-HTSPN

6.1.4. CMIF

O modelo CMIF — CWI[12] Multimedia Interchange Format [BuRL91, RJMB93], descreve um modelo para representação e manipulação de documentos multimídia. O modelo CMIF concentra-se na organização temporal das informações e na construção de documentos de grande porte formados por agrupamentos de documentos menores (estruturação hierárquica). Os documentos modelados com o CMIF são exibidos como se fossem seqüências de vídeo — o usuário inicia, suspende, reinicia, encerra etc. a apresentação do documento.

O modelo CMIF permite que uma apresentação seja recursivamente construída por uma série de subapresentações. Uma apresentação no CMIF pode ser denominada: *apresentação composta*, quando ela contém outras apresentações aninhadas; *apresentação atômica*, quando ela não contém nenhuma apresentação na sua subárvore de aninhamento.

Um documento é, portanto, uma árvore com hierarquia de apresentações compostas, onde as folhas dessa árvore são apresentações atômicas [RJMB93].

[12] CWI — Centre for mathematics and Computer Science in Amsterdam.

Um *evento*, no modelo conceitual CMIF, é definido como o menor fragmento de mídia que pode ser manipulado pelo sistema. São exemplos de eventos: pequenos fragmentos de vídeo, áudio, imagem ou texto. Uma apresentação atômica é uma coleção de eventos. *Marcadores* são utilizados como pontos de ancoragem das conexões entre componentes do documento. Esses marcadores ficam armazenados juntamente com os dados.

Os eventos que compõem uma apresentação atômica estão relacionados através de relacionamentos temporais ("timing constraints"). São usados dois mecanismos para definir os relacionamentos temporais: composição paralela e seqüencial; e arcos de sincronização, ou simplesmente sinc-arcs.

A própria construção da árvore de hierarquia de uma apresentação define os relacionamentos temporais através dos mecanismos de composições paralela e seqüencial. O posicionamento dos nós dentro da árvore de apresentação define o comportamento temporal de sua exibição.

Além da definição de relacionamentos temporais a partir da estrutura hierárquica dos componentes de uma apresentação, através do modelo CMIF, pode-se definir relacionamentos entre quaisquer nós de uma mesma sub-árvore através dos *sinc-arcs* (arcos de sincronização). Um *sinc-arc* é uma relação entre marcadores em dois eventos de uma mesma apresentação atômica, que especifica um determinado valor de retardo. Suponha, como exemplo de sua utilização, que em uma sub-árvore de uma apresentação atômica foi especificado que um nó de vídeo deve ser apresentado em paralelo com uma música (através do posicionamento paralelo dos dois nós na árvore), e que nós desejamos retardar a exibição do início da música de 2 segundos. Isto pode ser feito pela adição de um sinc-arc do evento inicial do nó de vídeo para o evento inicial do nó de áudio (música), atribuindo a ele um retardo de 2 segundos.

O modelo CMIF também define os *elos de hipermídia* (*hyperlinks*), que são relacionamentos entre fragmentos de informações que dependem de um evento de interação do usuário. Âncoras são definidas como uma parte de um item de dados. Os hyperlinks são conexões entre duas âncoras, origem e destino do elo.

Outro conceito importante do CMIF é o conceito de canais. Um *canal* é uma abstração de características específicas de um dispositivo de saída da plataforma de exibição, e é constituído por um conjunto de eventos. Cada canal possui um tipo de mídia específico que deve ser do mesmo tipo dos eventos que estão situados nele. Um canal de áudio define, por exemplo, qual o volume de exibição dos componentes de dados que estão contidos nele. Um canal de imagens gráficas pode especificar qual a posição na tela ocupada pelos seus componentes, etc.

Todos os elementos do modelo CMIF (eventos, canais, nós de composição, sinc-arcs, âncoras e elos) possuem uma lista de atributos que descrevem suas propriedades, como: nome, duração de exibição, etc. Essas propriedades estão vinculadas ao tipo de mídia e à natureza de cada elemento.

Com o objetivo de editar e exibir documentos CMIF, foi construído o ambiente CMIFed (CMIF Editor) [RJMB93].

O ambiente de autoria hipermídia do CMIFed descreve uma interface com o usuário que oferece três diferentes visões de um mesmo documento multimídia. Utilizando-se dessas visões, o autor pode manipular todos estágios da tarefa de criação da apresentação de um documento em um único ambiente de interface integrado. As visões do ambiente CMIFed são: a *hierarchy view*; a *channel view*; e o *player*.

A *hierarchy view* oferece a edição e visualização da estrutura da apresentação. Nesta view o autor define a estrutura do documento. Na hierarchy view, são especificadas as relações temporais de composição paralela (componentes posicionados lado a lado) e composição seqüencial (seqüência definida pelo posicionamento vertical com os nós na parte superior sendo apresentados antes que os da parte inferior). O autor compõe seu documento montando a estrutura hierárquica definida pelo aninhamento das subapresentações (inclusão de um retângulo em outro). A Figura 51 mostra a janela de interface que fornece a visão hierárquica (hierarchy view) de um documento no CMIFed.

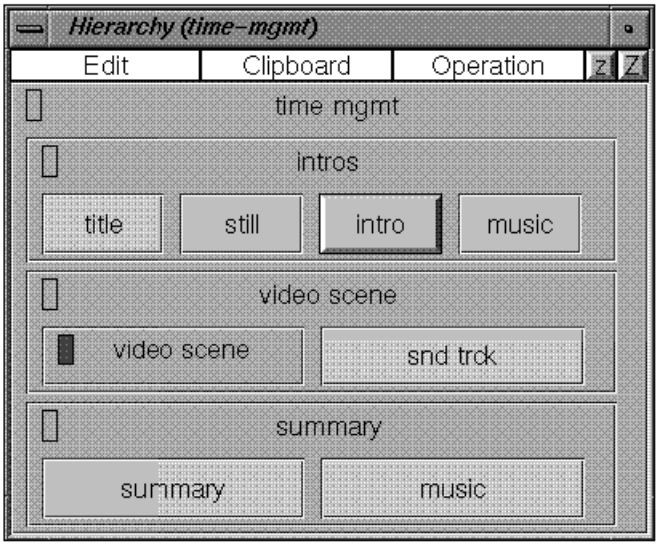


Figura 51: A *hierarchy view* do CMIFed

A *channel view* fornece uma visão baseada nos canais criados pelo autor. A Figura 52 mostra como exemplo, a janela do CMIFed que fornece a visão de canais (channel view) da apresentação "time-mgmt". Nessa figura, são mostrados os canais utilizados na apresentação time-mgmt, que são: Video, Title text, Body text, Sound e Image. O posicionamento dos eventos dentro dos canais está relacionado aos instantes de suas ocorrências. Os eventos são alinhados seqüencialmente no tempo dentro dessa visão. A ordem de suas disposições dentro dos canais no sentido vertical define o alinhamento temporal (de cima para baixo). Ou seja, o evento "shot 1"ocorre antes do evento "shot 2", que ocorre antes do evento "shot 3".

A channel view permite a definição de relacionamentos temporais com o emprego de sinc-arcs, que podem ser criados e alterados dentro dessa visão. No exemplo da Figura 52, foi criado um sinc-arc do evento "still" do canal "Image", para o evento "title" do canal "Title text". Esse sinc-arc é representado por uma seta que conecta os dois eventos.

A terceira visão da apresentação, fornecida pelo *player* (exibidor), mostra o efeito da exibição de uma determinada apresentação em uma plataforma específica. O player permite que o autor possa editar os aspectos de layout da apresentação, como a geometria das janelas usadas por canais que utilizam o dispositivo de vídeo. O autor também pode definir

e posicionar os pontos de ancoragem em imagens estáticas da apresentação. Um exemplo de player é mostrado na Figura 2.4.

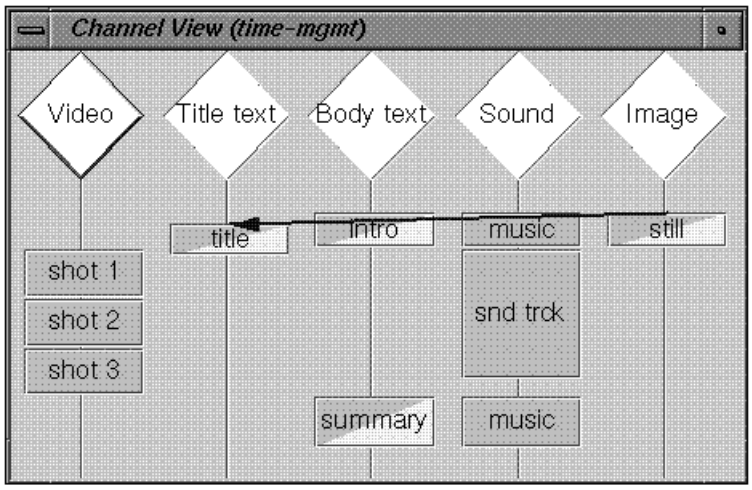


Figura 52: A *channel view* do CMIFed

O player possui uma janela de controle (posicionada no canto inferior esquerdo). A função dessa janela é controlar o tempo de exibição que está sendo mostrado na sua visão. O layout das janelas pode ser modificado, e as alterações de comportamento dos nós são armazenadas dentro dos canais das mídias.

A terceira visão da apresentação é fornecida pelo formatador CMIFed, denominado *player*, que é o elemento responsável pelo controle da apresentação do documento. A Figura 53 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.

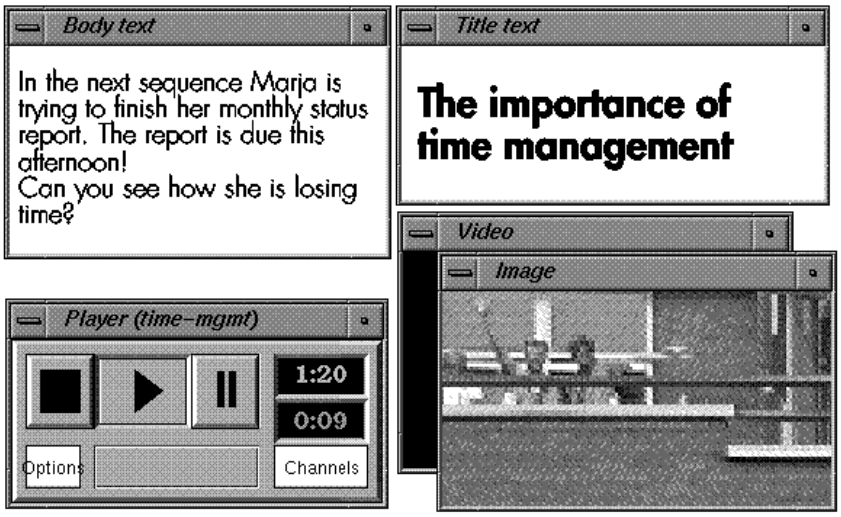


Figura 53 - Interface do executor CMIF

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. 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.

A principal restrição apresentada pelo CMIF é a impossibilidade da definição de relacionamentos entre eventos localizados em composições distintas, uma vez que os sinc-arcs só permitem relacionamentos entre eventos irmãos na estrutura hierárquica. Esta limitação do modelo de documentos CMIF levou os pesquisadores do CWI a desenvolver uma extensão deste modelo, incorporando ao mesmo, facilidades definidas no modelo Dexter. O novo modelo criado foi denominado Amsterdam Hypermedia Model e suas principais características são descritas na seção seguinte.

6.1.4.1. Amsterdam Hypermedia Model

O Amsterdam Hypermedia Model (AHM) [HaBR94, HaBR93] usa o conceito de evento do CMIF para definir o sincronismo da apresentação dos documentos. Os *eventos* do AHM são associados com a exibição de intervalos (blocos de dados).

No AHM a apresentação dos segmentos de um documento pode ser alinhada no tempo através da uma composição hierárquica que permite comandar a exibição paralela ou seqüencial dos segmentos de um documento. O modelo permite a definição de restrições temporais relacionando pais e filhos ou irmãos na hierarquia, as restrições são do tipo a exibição do filho δ deve começar 10 segundos após o início da exibição de seu pai. A especificação de sincronismo baseada na hierarquia é complementada com arcos de sincronização que permitem a definição de restrições temporais entre um ponto de referência origem e um ponto de referência destino que podem estar em componentes localizados em composições distintas. A restrição é definida na composição que é o ancestral comum mais próximo, na hierarquia de composições, dos componentes relacionados.

A interação do usuário com o documento é tratada por esse modelo com o emprego de hyperlinks que definem relacionamentos um-para-um entre âncoras, componentes atômicos ou compostos. Uma extensão ao CMIF é o conceito de contexto de âncoras que permite

especificar que partes da apresentação permanecem, ou não, sendo exibidas quando um link é disparado.

A idéia de composição de componentes atômicos (blocos de dados) formando componentes compostos que podem, inclusive, conter outros componentes compostos é usada no AHM para facilitar o reuso de apresentações. Nesse aspecto merece destaque a facilidade que o modelo oferece de testar se há conflito na utilização dos recursos (canais) quando uma composição é incluída em um documento (que na realidade é outra composição). O principal problema na definição das composições AHM 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). Ao contrário do NCM, no AHM existem relacionamentos definidos nas composições e outros definidos nos arcos. Quando definidos em arcos, não permitem um reuso estruturado.

6.1.5. MHEG

O padrão MHEG — Multimedia and Hypermedia Expert Group [MHEG95], tem por objetivo definir uma codificação padronizada para objetos multimídia e hipermídia em sua forma final permitindo seu intercâmbio entre serviços e aplicações através de qualquer tipo de meio de transmissão.

O MHEG tem a intenção de permitir o intercâmbio de objetos, nesse sentido ele atua como um *container*. Os objetos intercambiados devem estar em sua forma final, isto é, prontos para serem apresentados, nesse sentido o MHEG atua como um descritor da futura apresentação do objeto. A ferramenta utilizada para permitir a realização desses papeis é a *composição*.

O padrão segue o paradigma de orientação a objetos na descrição dos elementos que o compõem. No contexto do MHEG, o termo “classe” denota uma estrutura de informação multimídia/hipermídia intercambiável, a partir da qual objetos MHEG podem ser instanciados. Como os objetos MHEG devem ser considerados como dados e não como código executável, não são definidas interfaces na forma de assinaturas de métodos.

Assume-se que a semântica associada com o uso das classes MHEG é definida no nível de aplicação e não no nível MHEG. A Figura 54 lista as classes MHEG.

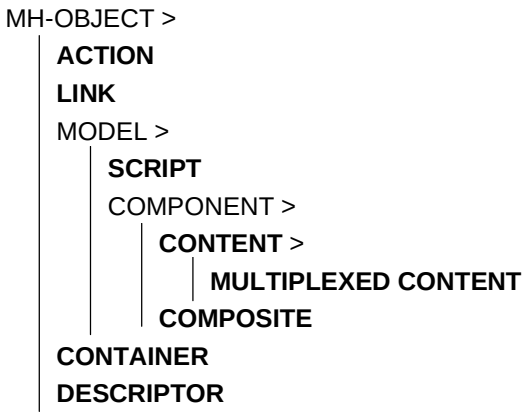


Figura 54: Hierarquia de Classes MHEG. Na figura, > significa "possui as seguintes subclasses" e apenas as classes enfatizadas podem ser intercambiadas.

MHEG model é uma classe abstrata cujas características são herdadas pelas classes: script, content, multiplexed content e composite. A partir das classes modelo são instanciados *objetos modelo* que são os objetos intercambiáveis. Para permitir o reuso em diferentes contextos dos dados intercambiados nos objetos modelo, é possível instanciar diversos *run-time objects* a partir de um único objeto modelo. A apresentação ou ativação de um rt-object não afeta o estado objeto modelo a partir do qual ele foi instanciado, o rt-object corresponde a uma visão específica de um dado ou composição modelo. A Figura 55 ilustra o relacionamento entre classes e objetos MHEG.

O MHEG utiliza a técnica de composição para agrupar um conjunto de objetos para fins de intercâmbio permitindo a especificação dos recursos mínimos de sistema necessários para transportar e exibir o objeto composto. Para tal, o container (ou objeto composto usado para representar os documentos) inclui um *objeto descritor* que resume as informações que o servidor ou uma aplicação qualquer irá necessitar para organizar e otimizar a transmissão e o gerenciamento dos objetos em uma apresentação em tempo real.

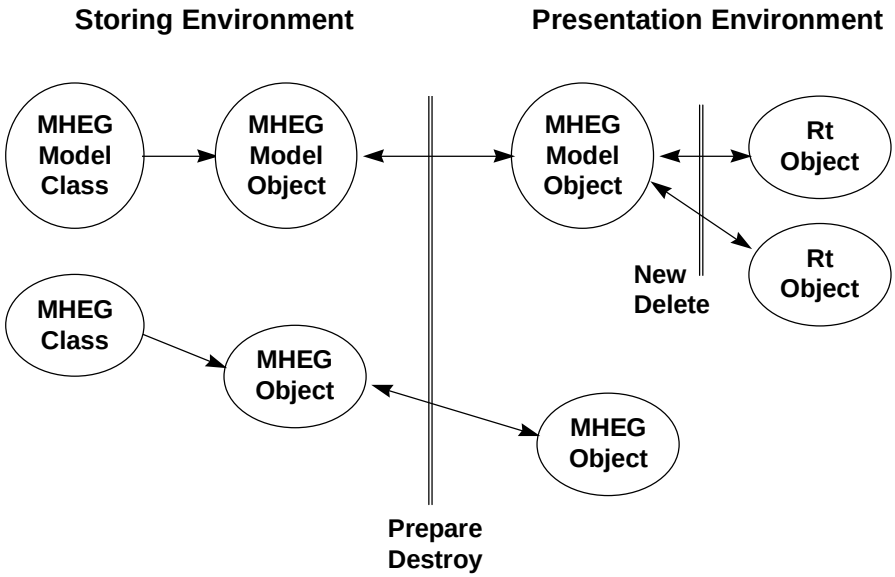


Figura 55: Relacionamento entre classes e objetos MHEG.

O objeto descritor fornece, em arquivos read-me com formato legível para humanos, informações definidas pelas aplicações fonte que são lidas pelas aplicações e máquinas MHEG destino (responsáveis pela manipulação dos objetos MHEG). Dentre as informações fornecidas estão: uma descrição da forma como será utilizado o espaço genérico MHEG, uma lista de "hooks" que descrevem os formatos dos *content objects* que carregam os dados do documento (objeto composto MHEG), uma lista dos objetos e instruções associadas definindo níveis aceitáveis de degradação da qualidade ou de abandono da apresentação e uma lista de ações que espera-se que a máquina MHEG que irá exibir o documento possa executar.

O padrão MHEG possui uma estrutura denominada *generic space* que permite especificar o tempo e o espaço para apresentação dos componentes dos documentos. Todos os tamanhos e posições codificados nos objetos MHEG são expressos em coordenadas do espaço genérico MHEG. É responsabilidade das aplicações definir o relacionamento entre o espaço genérico MHEG e o mundo físico. O espaço genérico possui quatro eixos: um eixo temporal e os três eixos espaciais *x, y, z* empregados com seu significado matemático usual.

O MHEG possui um mecanismo de *canal virtual* que define um espaço lógico onde os rt-components são apresentados e percebidos pelos usuários. Os canais são associados a uma instância do espaço genérico MHEG especificada no objeto descritor. O mapeamento do

canal virtual em um dispositivo físico como um aparelho de TV, uma câmera etc. em um sistema de apresentação é deixado para a aplicação, embora possam ser colocadas instruções sobre a configuração esperada no objeto descritor.

A classe *action* define uma estrutura que especifica um conjunto sincronizado de ações elementares a serem aplicadas a um ou mais destinos. O padrão define ações elementares que afetam os seguintes comportamentos dos objetos MHEG:

- *Preparação*: “prepare” e “destroy” são aplicadas para adicionar e remover objetos MHEG no sistema.
- *Criação de rt-objects*: “new” e “delete”.
- *Apresentação*: “run” e “stop” são usadas para controlar a progressão de time-based rt-components.
- *Rendition*: são fornecidas ações para controlar a projeção de rt-components no sistema, por exemplo, “set speed” para time-based mídias e “set size” para mídias visíveis.
- *Interação*: são fornecidas ações para controlar a interação com um rt-component no sistema. Por exemplo, “set selectable”, “set modifiable”, “get seletion status” etc.

A técnica de composição também é usada para descrever as apresentações, que variam de objetos monomídia simples a objetos multimídia complexos e interações hipermídia. Um *composite object* define uma lista de elementos agrupados para apresentação. A apresentação de uma composição consiste da apresentação de seus elementos. Um elemento de um rt-composite é um *socket*. Rt-components são plugados nos sockets que podem ser de três tipos:

- *empty*: componente nulo plugado.
- *presentable*: rt-content ou rt-multiplexed content plugado.
- *structural*: rt-composite plugado.

Os rt-componentes plugados nos sockets podem ser removidos ou substituídos através da ação *plug*. Um atributo denominado *original perception* é usado para inicializar o espaço

da composição onde os sockets filhos são posicionados. Quando uma composição está executando, sempre que sua posição temporal corrente cruzar a posição temporal onde está posicionado um socket, uma ação *run* é automaticamente direcionada para o objeto "plugado" no socket. No MHEG a interação com o usuário baseia-se nos comportamentos de seleção e modificação dos rt-components e sockets.

O *composite object* do MHEG inclue uma especificação da apresentação de seu conteúdo em termos de tamanho, velocidade, volume etc. Um dado *content object* incluído em um composite pode ser apresentado de várias formas. Cada uma delas é denominada *presentable* e possui seu próprio estado de apresentação. O composite object inclui também relações no tempo e no espaço entre os diferentes presentables, e a especificação de ações que devem ser executadas quando determinadas condições são satisfeitas. As condições são especificadas em termos de modificações no estado dos sockets ou do estado de um objeto intercambiado. Estes relacionamentos que especificam o comportamento da composição são definidos em link objects.

Os *link objects* são usados para definição das relações espaciais, temporais e condicionais entre objetos MHEG. Um elo contém uma condição, que quando satisfeita, provoca o processamento de ações nos destinos o que produz o efeito desejado do elo. Os elos MHEG são dirigidos por eventos. Cada elo descreve os eventos que podem dispará-lo e o efeito que o disparo provocará. Um evento é gerado por uma mudança no comportamento de um objeto, rt-object ou canal. Por exemplo, um objeto tornou-se "ready", ou o volume de um rt-content aumentou. Quando os elos são preparados eles passam a ser disparáveis. A condição de disparo de um elo pode ser: uma condição genérica única que, por sua vez, pode ser uma trigguer condition ou uma constraint condition (quando o elo é preparado sua condição é avaliada) ou ainda, uma combinação lógica de condições genéricas. Um exemplo de condição é: Source value: "get audible volume of rt-content N" (é sempre uma ação "get"); Previous condition: == 10 (pode ser "any" ou "negation") e Current condition: < 5 (pode ser uma ação "get").

O padrão MHEG descreve uma série de *comportamentos* (behaviours) que os content objects e os presentables devem exibir sob o controle de uma máquina MHEG. Esses comportamentos são especificados em termos de ações e do efeito que as ações impõem no

estado dos objetos. O padrão não define como os objetos são manipulados pela máquina, mas sim o efeito esperado nos seguintes *estados*: preparação, execução, modificação, timestone, interação e composição. O estado de *preparação,* cujos valores são "pronto" ou "não-pronto" pode ser usado, por exemplo, para indicar se um objeto foi carregado na memória principal e está pronto para ser usado. O estado de *execução*, com os valores "executando" ou "não-executando" indica que um presentable está sob o controle do processo "run" da máquina MHEG. O estado *modificação*, com os valores "não-modificavel", "modificável", "modificando" e "modificado", indicando a possibilidade de modificar, ou se o objeto está sendo ou foi modificado, permite, por exemplo, que o usuário modifique o valor de um campo que contém um texto em um objeto MHEG. O MHEG permite que o projetista de um objeto especifique momentos especiais ao longo do "tempo de vida" de um objeto, esses momentos são denominados timestones, o estado *timestone* guarda o valor do último timestone ocorrido. O estado de um elemento de *interação* indica o valor de um identificador de interação, especificado pelo projetista do objeto e gerado pela máquina MHEG, quando o objeto é selecionado pelo usuário. O estado da *composição* é um valor arbitrário atribuído por uma ação associada a um elo, essa facilidade torna possível ao projetista de um objeto MHEG fornecer informação de estado de uma composição.

Apesar de não definir uma linguagem de script, o MHEG fornece a classe script para encapsular um script e uma indicação da linguagem usada. O MHEG assume que a linguagem de script usada em um objeto script é capaz de referenciar objetos MHEG, rt-objects e acessar seus atributos.

Os elos MHEG são semelhantes aos do NCM (baseados em eventos). Porém, MHEG define um evento primitivo como um instante no eixo temporal de um dos seguintes tipos:

- temporal: timestone ou delay
- action: resultado do processamento de uma ação
- interaction: provocado por interação do usuário com a aplicação

Os eventos são associados ao estado de um objeto, ou seja, as âncoras no MHEG são sempre todo o conteúdo de um nó (o equivalente a âncora λ) no NCM. Esta restrição

inviabiliza a captura de seleções de regiões diferentes de um mesmo nó. Enquanto no NCM basta associar duas âncoras as respectivas regiões e associá-las a eventos de seleção, no MHEG é necessário dividir o conteúdo do nó em diferentes content objects e associá-los a sockets diferentes de um composite object.

O MHEG define duas composições container (unidade de intercâmbio) e composite (unidade de apresentação). O NCM possui apenas uma composição, agrupamento de nós e elos, usados tanto como unidade de intercâmbio quanto de apresentação. Porém, com relação ao aspecto apresentação, toda a especificação de sincronismo no NCM é capturada pelos elos, não existe como no MHEG a noção de posicionamentos dos elementos da composição em relação à progressão da apresentação da composição (posicionamento dos sockets no espaço genérico do composite object). Entretanto, cabe ressaltar que o mesmo efeito pode ser reproduzido no NCM com o emprego de um elo com o evento "início de apresentação da composição" como condição de disparo e operadores retardo aplicados aos destinos do elo (uma ação inicia para cada componente da composição).

A idéia de contextualização da apresentação de componentes, no MHEG pode ser implementada com o emprego de ações "plug" que associa contents ou composites aos sockets de um composite e "set" que modificam o comportamento dos objetos. Porém as facilidades disponibilizadas para tal são de muito baixo nível, não existe, como no NCM, uma associação descritor-nó dependente da navegação.

6.1.6. PREMO

O PREMO (Presentation Environments for Multimedia Objects) [HeRL95] é um padrão para apresentação de objetos multimídia desenvolvido pelo ISO/IEC JTC1/SC24. O PREMO é adequado para desenvolvedores de aplicações que desejam incluir efeitos multimídia em suas aplicações sem restringir-se a modelos que definem apenas como codificar e transferir documentos multimídia. Esse padrão é centrado no desenvolvimento de uma ferramenta de programação para desenvolvimento de aplicações multimídia.

Do ponto de vista editorial o PREMO está atualmente dividido em quatro partes: Fundamentos do PREMO — Parte 1, Componente Base — Parte 2, Componente de

Modelagem, Apresentação e Interação — Parte 3, e Serviços de Sistema Multimídia — Parte 4. Estruturalmente o PREMO é dividido em componentes. Um *componente* é uma coleção de tipos de objetos e tipos de dados, a partir dos quais objetos e outras estruturas podem ser instanciados. Os objetos internos a um componente são projetados para cooperar uns com os outros e oferecer um conjunto bem definido de capacidades funcionais que pode ser usado por objetos externos ao componente. Um componente pode oferecer *serviços* que podem ser utilizados em um ambiente distribuído, ou eles podem ser usados como um conjunto de objetos ligados ("linkados") diretamente a uma aplicação.

A Parte 1 do padrão, denominada Fundamentos do PREMO [PRE95a], apresenta uma visão geral do PREMO descrevendo seu escopo, justificativa, e explicando os conceitos básicos do ambiente. A principal contribuição dessa parte do padrão é a definição do modelo de objetos adotado na definição do ambiente. O modelo de objetos baseia-se nos conceitos de subtipos e herança. O modelo de objetos do PREMO originou-se de uma adaptação do modelo de objetos distribuídos desenvolvido pelo consórcio OMG. No PREMO uma forte ênfase é dada à habilidade dos objetos de serem ativos. Conceitualmente, diferentes mídias (por exemplo, uma seqüência de vídeo e a correspondente trilha sonora) são consideradas como atividades paralelas que devem alcançar pontos de sincronização distintos possivelmente definidos pelo usuário. Assim, a sincronização é modelada usando as noções de sincronização entre processos concorrentes, no PREMO, objetos ativos. As operações dos objetos do PREMO podem ser:

- Síncronas: o chamador é suspenso até que o chamado atenda o serviço solicitado.
- Assíncronas: o chamador não é suspenso, e as solicitações de serviço são enfileiradas no chamado. Nesse caso não são permitidos valores de retorno.
- Amostradas: o chamador não é suspenso, mas as solicitações de serviço não são enfileiradas no chamado. Ao invés disso, as solicitações sobrepõem-se enquanto o chamado não as atende.

A Parte 2 [PREM95b], que descreve o Componente Base, lista um conjunto inicial de tipos de objetos e dados, úteis para construção, apresentação e interação com informação multimídia. O comportamento dos objetos base é definido através da especificação do tipo denominado *PREMOObject*, que é o supertipo comum de todos os tipos de objetos do

PREMO. Esse tipo de objeto define operações para iniciação, destruição, consultas de tipo etc. O Componente Base é uma coleção dos seguintes objetos:

- *Enhanced Data Objects*: a semântica associada a esses objetos define as interfaces para construção e modificação de um objeto de dados específico.
- *Event Handler Objects*: fornece as operações necessárias para implementação do gerenciamento de eventos no PREMO. A característica essencial do modelo de eventos do PREMO é a separação entre a fonte e o receptor dos eventos. As fontes difundem os eventos sem ter nenhum conhecimento dos objetos que os receberão; isso é feito através do encaminhamento dos eventos para os objetos *EventHandler*. Os interessados em receber os eventos registram-se no *EventHandler*, enviando uma solicitação baseada no tipo do evento e opcionalmente em restrições mais complexas. Os receptores são notificados pelo *EventHandler* da chegada de eventos. O padrão define dois importantes subtipos do *EventHandler*, denominados *synchronization-points* e *and-synchronization-points*. A característica adicional desses tipos é a possibilidade de condicionar o despacho dos eventos, abrindo a possibilidade de retardar o envio da notificação aos recipientes dos eventos.
- *Controller Objects*: a função de um *Controller* é coordenar a cooperação entre objetos. Um *Controller* é uma máquina de estados finitos autônoma e programável. As transições são disparadas por mensagens enviadas por outros objetos. As ações do *Controller* correspondem ao envio de mensagens a outros objetos.
- *Clock Objects*: fornecem uma interface uniforme para a visão que o sistema possui do relógio de tempo real.
- *Synchronization Objects*: as facilidades de sincronização incluídas no PREMO adotam um modelo de sincronização baseado em eventos. Nesse modelo, considera-se que cada objeto sincronizável progride autonomamente em um espaço de coordenadas unidimensional interno, que pode ser definido por números inteiros, reais ou no tempo (por exemplo, coordenadas inteiras podem ser usadas para identificar os quadros em uma seqüência de vídeo). Pontos nesse espaço são

denominados *pontos de referência*; para cada ponto de referência, um objeto e uma operação pode ser registrada, junto com uma instância de evento (o evento de sincronização). A operação é invocada pelo objeto sincronizável quando o ponto de referência é atravessado, usando o evento armazenado no ponto de referência como argumento. Tipicamente, o ponto de referência irá comunicar a ocorrência do evento a um EventHandler ou a um Controller. O tipo objeto sincronizável serve de supertipo para vários tipos específicos de mídia: vídeo, áudio, animações etc. Combinando objetos com sincronização baseada em eventos a ClockObjects através de herança múltipla podem ser definidos objetos com sincronização baseada no tempo. O objeto resultante dessa combinação, denominado *TimeSynchronizable*, possuem um atributo velocidade, que caracteriza o passo da progressão do objeto em seu espaço de coordenadas no tempo.

- *Aggregate Objects*: definem os tipos de objetos no PREMO, oferecendo facilidades tradicionais de agregação como listas, vetores, tabelas etc.
- *Producer Objects*: fornecem um encapsulamento para definição do processamento de objetos de dados para produção de objetos de dados refinados ou transformados. Objetos Producer podem receber objetos de dados de um número qualquer de fontes e entregar objetos de dados para um número qualquer de destinos.
- *Porter Object*: é o objeto base que interconecta os objetos PREMO a sistemas e ambientes externos, por exemplo, dispositivos físicos. Um objeto porter pode ser usado para exportar a saída do PREMO (um objeto *OutputPorter*) ou para importar dados para o PREMO (um objeto *InputPorter*). Subtipos desse objeto fornecem interfaces consistentes para "dispositivos virtuais" que representam dispositivos externos específicos, por exemplo, o frame buffer do display, um relógio, um microfone, um alto-falante etc. Objetos Porter podem também ser conectados a EventHandlers como possíveis fontes de eventos.

A Parte 3 apresenta o Componente de Modelagem, Apresentação e Interação [PREM95c]. Esse componente combina o controle de mídias com modelagem e geometria. O objetivo desse componente é definir uma família de tipos de objetos que descrevam a modelagem, o

rendering e a interação em termos abstratos. O componente é considerado abstrato no sentido que espera-se que sejam derivados componentes concretos de modelagem e apresentação através do refinamento dos tipos de objetos nele definidos. O modelo para apresentação descrito na Parte 3 baseia-se em outro padrão definido no SC24 chamado Computer Graphics Reference Model [CGRM92]. A idéia básica da Parte 3 é que uma apresentação multimídia é considerada como uma série de passos de transformação entre a aplicação e o operador. O termo transformação é empregado em um sentido genérico incluindo transformações entre modelos de cores, algoritmos de codificação e decodificação etc. Os passos de transformação são modelados no PREMO em termos de cinco níveis abstratos chamados *ambientes*. A saída de cada um dos ambientes é, resumidamente, a seguinte:

- *Ambiente de Construção*: nesse ambiente, os dados da aplicação que serão exibidos são preparados como um modelo a partir do qual podem ser produzidas cenas para apresentação. A aplicação só pode editar o modelo nesse ambiente.
- *Ambiente Virtual*: nesse ambiente uma cena do modelo é produzida. A cena consiste de um conjunto de primitivas de saída virtuais prontas para apresentação. A geometria dessas primitivas é completamente definida em todas as dimensões de forma que as cenas são geometricamente completas.
- *Ambiente de Visualização*: nesse ambiente, uma imagem da cena é gerada por projeção. A imagem consiste de um conjunto de primitivas de saída de visualização prontas para realização. As primitivas de saída no ambiente de visualização devem possuir uma dimensão geométrica mais refinada que no ambiente virtual.
- *Ambiente Lógico*: a saída desse ambiente é uma versão apresentável da imagem completamente pronta para realização. A imagem apresentável consiste de um conjunto de primitivas de saída lógicas.
- *Ambiente de Realização*: na saída desse ambiente corresponde a exibição de uma imagem apresentável. A exibição consiste de um conjunto de primitivas de saída de realização. A exibição não corresponde necessariamente a uma exibição em uma tela, pode ser em um dispositivo de áudio ou em um driver lógico.

O Componente Serviços de Sistema Multimídia (SSM), Parte 4 [PREM94], fornece uma infra-estrutura para construção de plataformas para computação multimídia distribuída que suportem aplicações multimídia interativas lidando com mídias sincronizadas em um ambiente distribuído heterogêneo. O Componente Serviços de Sistema Multimídia define a si próprio como um "middleware", isto é, um componente de software localizado na região que fica entre o sistema operacional e as aplicações específicas. O SSM não contém nenhuma noção de modelos geométricos ou de apresentação, ou de composição de entidades de apresentação abstratas; ele concentra-se no gerenciamento de baixo nível dos dispositivos multimídia, suas conexões, distribuição etc. Os objetos que constam do esquema do SSM descrevem um grafo de fluxo de dados. Os nós do grafo, formados pelos *device objects*, fornecem uma abstração para execução das mídias. Os arcos do grafo, denominados *connection objects*, fornecem uma abstração para o transporte de streams (cadeias de bytes). Os tipos device e connection herdam características do tipo *resource*. O tipo resource permite que um cliente descreva sua espectativa sobre a qualidade da media stream manipulada. As métricas de qualidade, por exemplo a banda passante, aplicam-se tanto aos objetos device quanto aos connection. Os objetos device transmitem e recebem o media streams através de uma *device port*. A interface para o objeto device fornece métodos através dos quais um cliente pode descobrir as portas disponíveis. Um objeto device que importa um stream é denominado *display device*, um objeto device que exporta um stream é denominado *capture device*; um objeto device que importa e exporta um stream é denominado *filter device*. A interface do tipo device fornece métodos para que os clientes descubram os objetos *format* e *protocol* disponíveis em cada porta. O tipo format descreve o contexto da mídia (formato de codificação, por exemplo, vídeo MPEG) e o tipo protocol descreve o contexto do transporte do stream. O objeto connection é uma abstração do mecanismo de transporte. Esse objeto fornece operações para *attach* e *detach* uma porta de dispositivo a um media stream. A operação attach testa a compatibilidade entre os objetos format e protocol disponíveis nas portas que irão participar da conexão. Para controlar o fluxo da media stream, o componente fornece o tipo de objeto *Stream*. A interface básica desse tipo prevê operações para suspender e retomar o fluxo do stream.

Tanto o PREMO quanto o NCM baseiam o controle e a especificação do sincronismo temporal no conceito de evento. O PREMO utiliza um "modelo de evento" que separa a

fonte dos receptores dos eventos. As fontes de eventos reportam os eventos capturados pelos event handlers para os recipientes apropriados registrados nos event handlers. Um subtipo do objeto event handler, chamado ANDSynchronizationPoint, permite a definição de um conjunto de eventos fonte que devem ocorrer antes que o objeto ANDSynchronizationPoint sinalize o fato para os recipientes. O objeto ANDSynchronizationPoint executa uma operação lógica *and* nos eventos de origem.

Os elos NCM atuam como PREMO synchronization point objects, entretanto o NCM permite a definição de uma expressão lógica baseada na ocorrência dos eventos de origem bem mais genérica que o ANDSynchronizationPoint do PREMO. O NCM permite o uso de operadores and, or, not e retardo nas expressões que habilitam o disparo de seus elos, enquanto o PREMO só permite o uso da lógica and. A funcionalidade dos controller objects do PREMO é desempenhada no NCM pelos objetos representação de composições, cujo conjunto de elos, ao receber notificações de ocorrências de eventos, dispara ações que controlam a sincronização entre os vários objetos de uma apresentação. Os objetos representação de nós terminais possuem funções equivalentes às dos synchronizable objects do PREMO. No NCM, a função dos porter objects é desempenhada pelos canais, que modelam dispositivos de E/S virtuais. Os aggregate objects do PREMO, uma vez que podem conter outros aggregate objects, seriam o equivalente, porém bastante simplificado, das composições NCM. O PREMO simplesmente disponibiliza esta classe de objeto base, porém, ao contrário do NCM cujo elemento principal são as composições, não dá muito enfoque ao uso destes objetos. A contribuição do PREMO é muito centrada na modelagem dos dispositivos de E/S, das conexões entre estes dispositivos e no tratamento dos eventos gerados ao longo de suas execuções. O aspecto composição só é abordado com a definição dos aggregate objects, porém não são definidas facilidades como os conceitos de perspectiva e herança de comportamento definido em composições. O padrão também não apresenta propostas para o problema de contextualização da apresentação de componentes, como faz o NCM com a associação de descritores a nós dependente da navegação.

6.2. Comparação entre os Modelos Descritos

Esta seção apresenta um quadro comparativo entre os sistemas descritos neste capítulo e o NCM. Os modelos são comparados com base nas características apresentadas no Capítulo 2, sendo divididos em quatro tabelas, uma para cada aspecto do esquema de avaliação (apresentação de componentes, relações temporais entre componentes, composições e especificação do ambiente). As características dos modelos apresentados nesta seção são baseadas não somente em elementos implementados, mas também nas potencialidades de implementação futura.

Tabela 1: Comparação entre os modelos apresentados no aspecto especificação da apresentação de componentes.

Especificação da Apresentação de Componentes							
	Granularidade		Duração		Flexibilidade		QoS
	Grossa	Fina	Previsível	Imprevisível	Contínua	Discreta	
Mode	+	+	+	–	+	+	+
Firefly	+	+	+	+	+	–	+
I-HTSPN	+	–	+	+	+	–	–
CMIF/AHM	+	+	+	–	–	–	–
MHEG	+	–	+	+	–	+	+
PREMO	+	+	+	–	–	–	+
NCM	+	+	+	+	+	+	+

Tabela 2: Comparação entre os modelos apresentados com relação ao aspecto relacionamentos temporais entre componentes.

Relacionamentos Temporais entre Componentes									
	Granularidade			Tipo			Flexibilidade		QoS
	Ponto	Intervalo	Composto	Ordenação	Duração	Complexa	Contínua	Discreta	
Mode	+	–	+	+	–	–	–	–	+
Firefly	+	–	–	+	+	–	–	–	–
I-HTSPN	+	+	+	+	–	–	+	–	–
CMIF/AHM	+	+	–	+	–	–	–	+	–
MHEG	+	+	+	+	–	+	–	–	–
PREMO	+	–	+	+	–	–	–	–	–
NCM	+	+	+	+	–	+	–	–	–

Tabela 3: Comparação entre os modelos apresentados com relação ao emprego de composições na estruturação dos documentos.

Composições						
	Granularidade		Tipo de Componentes		Aninhamento	Herança
	Grossa	Fina	Previsíveis	Imprevisíveis		
Mode	+	–	+	+	+	–
Firefly	+	–	+	+	–	–
I-HTSPN	+	–	+	+	+	–
CMIF/AHM	+	+	+	+	+	–
MHEG	+	+	+	+	+	–
PREMO	+	–	+	–	+	–
NCM	+	+	+	+	+	+

Tabela 4: Comparação entre os modelos apresentados com relação às facilidades para especificação do ambiente onde se dará a apresentação dos documentos.

Descrição do Ambiente				
	Mudanças no Comportamento	Sincronização Espacial	Estação de Trabalho	Suporte de Comunicação
Mode	–	+	–	+
Firefly	+	+	–	–
I-HTSPN	–	+	–	–
CMIF/AHM	–	+	–	–
MHEG	+	+	+	–
PREMO	–	+	+	+
NCM	+	+	+	–

Nas composições reside uma das diferenças mais fundamentais entre o modelo NCM e os outros modelos descritos. A menos do MHEG[13] e do NCM, todos os modelos descritos utilizam as composições apenas para a estruturação da apresentação dos documentos, considerando 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,

[13] As composições do padrão MHEG (composite e container) baseiam-se no conceito de composição do NCM [Casa91].

perdendo assim a estruturação lógica. No NCM as composições representam, de fato, uma estruturação lógica do documento.

Capítulo 7

Conclusões e Perspectivas

Um documento multimídia é por natureza uma composição. É uma entidade formada pela composição de fragmentos de informação representada através de diferentes mídias. Ao utilizar diversos fragmentos de informação para compor um documento, seu autor registra idéias que deseja difundir. A expressão das idéias no documento implica em uma ordenação lógica dos fragmentos de informação. Esta ordenação define o tempo e o espaço que deve ser ocupado pela apresentação de cada um dos componentes. Assim, um modelo conceitual de documentos multimídia deve incluir estruturas que capturem todos esses aspectos dos documentos.

Uma característica dos documentos multimídia que merece ser mencionada é o tamanho dos seus componentes (o armazenamento de vídeos demanda usualmente centenas de mega bytes ou mesmo giga bytes), assim uma meta que deve ser perseguida na modelagem destes documentos é evitar a replicação desnecessária de componentes a nível de documentos e do sistema que os manipula. Tendo isto em mente e tentando incorporar ao modelo o menor número de estruturas possível, foi elaborado este trabalho, cujas principais contribuições são apresentadas na seção que segue.

7.1. Contribuições

Quando se iniciou o trabalho que resultou nesta tese, o NCM já havia sido objeto de outros estudos [Casa91, SoCR95] que, entretanto, não abordaram com profundidade o aspecto de apresentação dos documentos NCM. Estes estudos concentraram-se nos aspectos estruturais [Casa91] e no controle de versões e suporte ao trabalho cooperativo [SoCR95]. Assim, a principal contribuição deste trabalho foi dotar o NCM de estruturas que permitem

ao modelo capturar especificações do sincronismo temporal e espacial nas definições de documentos. Cabe, entretanto, ressaltar que as novas facilidades incorporadas ao modelo impactaram as estruturas anteriormente definidas, exigindo uma readequação das mesmas.

Para tratar o aspecto tempo na definição dos documentos, foi introduzido no NCM o conceito de evento. Para definir relacionamentos entre eventos foi redefinido o conceito de elo NCM, que passou a capturar um relacionamento de *n* eventos de origem com *m* eventos de destino. Ações são executadas nos eventos de destino, caso uma expressão lógica, baseada na ocorrência dos eventos de origem, seja satisfeita. Com base nestas duas estruturas — eventos e elos — o sincronismo temporal da apresentação dos documentos foi capturado pelo NCM. A semântica da apresentação dos documentos NCM foi então definida pela ocorrência de uma coleção de eventos alinhados no tempo através dos elos, conforme formalmente especificado no Capítulo 5. Com algumas variações, o conceito de evento como apresentação ou seleção de regiões já havia sido utilizado em outros modelos de documentos [BHLM92, BuZe92, HaBR94, MHEG95]. O único modelo, do nosso conhecimento, que associa eventos a seleção, apresentação ou modificação de regiões é o MHEG. Entretanto, nesse modelo as regiões são sempre nós (o equivalente à região λ no NCM). Portanto, a menos de alguma publicação a qual não tivemos acesso, no contexto de modelagem de documentos multimídia, o conceito de evento, como definido no NCM [SoSC95], é uma contribuição original deste trabalho.

A especificação do sincronismo espacial no NCM é capturada pela entidade descritor. Além da configuração inicial das características espaciais da apresentação dos componentes dos documentos, o descritor permite ao autor programar mudanças no comportamento da apresentação em resposta à ocorrência de eventos. A separação da especificação das características de apresentação dos dados a serem apresentados abre a possibilidade de se especificar formas alternativas para sua apresentação. Esta facilidade torna possível aos autores definir apresentações contextualizadas para documentos ou mesmo para componentes de documentos. A separação entre descritor (características de apresentação) e nós (dados) pressupõe a definição de mecanismos que permitam a associação de um nó a um descritor que dá origem a uma apresentação (ou representação) específica do nó. No NCM, a associação depende do caminho percorrido, via navegação na estrutura do

documento, para atingir um determinado nó. O NCM permite a associação de um descritor a cada evento de destino de um elo, assim, se o acesso ao nó ocorre em consequência de uma navegação através do elo, o descritor a ele associado no respectivo elo é agregado ao nó para criar uma representação do mesmo. Se o acesso a um nó se dá por navegação em profundidade na estrutura de contextos aninhados, o contexto pai pode definir o descritor a ser associado ao nó. Nesse caso, o conceito de perspectiva permite definir alternativas diferentes de apresentações para um nó, isto é, o modelo permite que descritores diferentes sejam associados a perspectivas diferentes de um mesmo nó. O descritor pode ser especificado pelo usuário que deseja criar a representação do nó ou, caso nenhuma das alternativas defina o descritor, um descritor default, associado à classe do nó, é usado. Variações da idéia de descritor são também usadas em outros modelos [BHLM92, HaBR94, BuZe93a, WSSD96, MHEG95], porém, nenhum deles define, como faz o NCM, um esquema dinâmico e dependente da navegação para a agregação dos descritores aos nós. Do ponto de vista de contextualização da apresentação de nós, outra contribuição do NCM é o conceito de elo visível. Dependo da perspectiva através da qual se observa um nó (e consequentemente do caminho navegado até atingir-se o nó) determinados elos são ou não ativos (habilitados para disparar). Conforme menciona a referência [Zell95], estas são também contribuições originais do NCM ao problema, na época aberto, de contextualização da apresentação de documentos multimídia.

Ainda com relação ao NCM, considerando que o trabalho aqui apresentado é uma extensão à versão do modelo descrita em [SoCR95], cabe explicitar as principais contribuições acrescentadas:

- Definição mais precisa dos conceitos de conteúdo, região, unidades de informação, unidades de informação marcadas e identificação de âncoras.
- Especificação dos métodos e atributos para os objetos de composição, em geral.
- Especificação dos métodos para os diversos nós de contexto.
- Especificação dos métodos de exibição e edição da estrutura e sincronismo dos nós e elos.
- Relacionamento dos exibidores e editores de estrutura e sincronismo com a arquitetura em camadas.

- Atualização da definição dos conceitos de perspectiva, elo, elo visível e entidade virtual.
- Inclusão de novas classes para definição do modelo de apresentação, entre elas: script, ponto de encontro, lista de operações e descritor.
- Redefinição da classe elo para especificação de relações n:m entre eventos.
- Definição das ações e condições de um elo e descritor.
- Inclusão de novas classes de elos: Sinc-elo, hiper-elo e elo genérico.
- Relacionamento das diversas novas classes incluídas com a arquitetura HyperProp.
- Definição da hierarquia de classes do NCM baseada nos objetos de armazenamento, de dados e de representação.
- Definição do controle de versões dos objetos de representação.
- Redefinição das operações de controle de versões, levando em conta o requisito de vários objetos de representação poderem ser derivados de um único objeto de dados.

Com o intuito de validar o modelo elaborado, o trabalho, aqui apresentado, avançou na direção da elaboração de ambientes que permitissem a edição a execução de apresentações de documentos NCM.

A idéia que norteou o desenvolvimento do ambiente de edição foi a de que os documentos multimídia são estruturas complexas, sendo necessário tratá-los segundo visões distintas. Três visões foram consideradas relevantes para o processo de edição: estrutura (aninhamento de composições e elos), tempo (sincronismo temporal) e espaço (sincronismo espacial). Foi então implementada uma interface [MSCS97, CMSS96, Much96, Cost96] que permite a visualização e edição de documentos NCM segundo três janelas integradas que fornecem aos usuários as visões estrutural, temporal e espacial de um documento. A implementação da ferramenta de edição se mostrou de extrema importância, pois permitiu depurar o modelo e provou sua viabilidade e potencialidade em termos de autoria.

Foi projetado também um ambiente para execução de apresentações de documentos NCM, cuja arquitetura [RoSS97a, RoSS97b] foi detalhada e implementa [Rodr97].

Uma outra contribuição importante desta tese foi que, ao definir o mapeamento de NCM em RT-LOTOS, foi proposto um método que permite aplicar a Técnica de Descrição Formal RT-LOTOS para especificar modelos de documentos baseados em composições aninhadas e verificar a consistência, inclusive de plataforma, de documentos baseados neste tipo de modelo [SCSS97, CSOS97].

7.2. Trabalhos Futuros

Por fim, como não poderia deixar de ser, o trabalho de pesquisa cujos resultados foram aqui apresentados abriu várias perspectivas de trabalhos futuros. Dentre eles merecem destaque:

- O emprego de técnicas de descrição formal para verificar a consistência de documentos produzidos nas ferramentas de autoria. A idéia é acrescentar na ferramenta de autoria facilidades para, por exemplo, traduzir um documento NCM em uma especificação RT-LOTOS que, por sua vez, é traduzida em um DTA pela ferramenta RTL. A mesma ferramenta permite empregar técnicas de análise de alcançabilidade [CoOl95] para verificar a consistência na especificação do sincronismo.
- Analisar a viabilidade de usar a estrutura de dados resultante do mapeamento NCM → RT-LOTOS → DTA como estrutura base para o módulo executor do ambiente de execução do HyperProp. Como esta estrutura serviu de base para a geração dos planos de exibição simulados, ela se coloca como candidata para estrutura base para geração dos planos de exibição em tempo de execução pelo executor HyperProp.
- Do mesmo modo que no ambiente de autoria, é importante que o executor 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 importantes trabalhos a serem realizados.
- A implementação de uma nova versão da ferramenta de autoria, desta feita com o apoio de especialistas na área de interfaces usuário-computador, e a edição de documentos de grande porte por usuários finais para testar, com este tipo de usuário, o modelo e o ambiente de edição e execução do HyperProp.

- Embora o NCM permita a especificação flexível do sincronismo temporal de seus documentos, as ferramentas de edição e execução atuais não fazem uso desta facilidade do modelo. O estudo de algoritmos de elasticidade para as fases de autoria e execução e sua implementação na ferramenta de edição e no executor do HyperProp são um importante trabalho futuro. Algumas opções de técnicas citadas na literatura incluem programação linear, programação dinâmica, ordenação topológica, caminho mais curto, entre outras. Cabe destacar que esta flexibilidade tem que ser estudada em conjunto com as técnicas que serão empregadas para verificar a consistência dos documentos, pois a possibilidade de definir apresentações flexíveis dificulta enormemente a verificação da consistência dos documentos.
- Um trabalho futuro consiste em incorporar, ao ambiente de execução do sistema HyperProp, funções de pré-carregamento dos objetos, a fim de evitar que retardos na rede e nos próprios sistemas operacionais prejudiquem a qualidade da apresentação dos documentos. Neste sentido, o uso do plano de exibição para buscar os componentes que serão apresentados no futuro é uma abordagem bastante promissora que, entretanto, apresenta problemas com relação às cadeias temporais cujo primeiro evento é imprevisível. Neste caso é preciso associar uma probabilidade de ocorrência aos eventos iniciais da cadeia e, com base nesta probabilidade, decidir se vale a pena buscar antecipadamente componentes cujas exibições estão a ela associadas. Uma outra abordagem interessante para este problema seria adaptar a estratégia visão olho-de-peixe, que no Capítulo 4 é usada para decidir que componentes de um documento complexo devem ser exibidos para o usuário na visão estrutural, para buscar antecipadamente os componentes "mais próximos" do nó em foco, no caso, o nó que está sendo apresentado em um determinado instante.
- Analisar esquemas de armazenamento dos componentes de documentos multimídia baseados em composições aninhadas em sistemas com servidores distribuídos é também um trabalho de pesquisa bastante interessante. Mais especificamente, poderia ser investigado como as informações estruturais e de sincronismo

fornecidas pelos nós de composição e elos podem ser utilizadas para decidir onde e como alocar os vários componentes do documento.

- A definição e implementação de mecanismos que garantam a segurança dos documentos NCM manipulados no sistema HyperProp nos aspectos confidencialidade e integridade se constitui também em um interessante trabalho a ser realizado no futuro.

Anexo A

Especificação Formal da Sintaxe do NCM em ASN.1 (Abstract Syntax Notation One)

-- 1 Module Entidade

```
NCM-Entidade  {entidade(-- 1)}

DEFINITIONS ::= BEGIN

EXPORTS
        Uid,
        Acl
        Descricao;

IMPORTS ;

Uid ::=  OBJECT IDENTIFIER

Acl ::= SEQUENCE OF Ac

Ac ::= CHOICE
{
        ac-user         Ac-user,
        ac-grupo        Grupo
}

Ac-user ::= SEQUENCE
{
        user            GraphicString,
        permissoes      Permissoes
}

Ac-grupo ::= SEQUENCE
```

```
{
        grupo           Grupo,
        permissoes      Permissoes
}

Grupo ::= SEQUENCE OF GraphicString

Permissoes ::= SEQUENCE
{
        read    BOOLEAN,
        write   BOOLEAN,
        play    BOOLEAN
}

Descricao ::= SEQUENCE
{
        dono            GraphicString,
        criacao         UTCTime,
        modificacao     UTCTime
}

END
```

-- 2 Module Ancora

```
NCM-Ancora  {ancora(-- 2)}

DEFINITIONS ::= BEGIN

EXPORTS
                Ancora;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)};

Ancora ::= SEQUENCE
{
        ancora-id       Uid,
        regiao          Regiao
}

Regiao ::= CHOICE
{
        lambda                  Lambda,
        unid-inf-marc           Unid-inf-marc
}

Lambda ::= 'lambda'

Unid-inf-marc ::= SEQUENCE
{
        inicio          INTEGER,
        fim             INTEGER
}
```

END

-- 3 Module Elo

```
NCM-Elo  {elo(-- 3)}

DEFINITIONS ::= BEGIN

EXPORTS
                Elo,
                Extremidade,
                Perspectiva,
                Tipo-evento;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)}
                no-id
                        FROM NCM-No  {no(-- 7)}
                Ancora
                        FROM NCM-Ancora  {ancora(-- 2)}
                No-descritor
                        FROM NCM-No  {no(-- 7)}
                Ponto-de-encontro
                        FROM NCM- Ponto-de-encontro {ponto-de-encontro(-- 5)};

Elo ::= SEQUENCE
{
        elo-id                          Uid,
        extremidades-origem             Extremidades-origem,
        extremidades-destino            Extremidades-destino,
        ponto-de-encontro               Ponto-de-encontro
}

Extremidades-origem ::= SEQUENCE OF Extremidade

Extremidades-destino ::= SEQUENCE OF Extremidade-dest

Extremidade  ::= SEQUENCE
{
        perspectiva             Perspectiva,
        ancora                  Ancora,
        tipo-evento             Tipo-evento
}

Extremidade-dest  ::= SEQUENCE
{
        descritor-extr          No-descritor    OPTIONAL,
        extremidade             Extremidade
}

Perspectiva ::= SEQUENCE OF no-id

Tipo-evento ::= ENUMERATED
{
        apresentacao    (1),
```

```
        selecao         (2),
        atribuicao      (3)
}
```

-- 4 Module Script

```
NCM-Script  {script(-- 4)}

DEFINITIONS ::= BEGIN

EXPORTS
                Script;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)};

Script ::= SEQUENCE
{
        script-id       Uid,
        script-tipo             Script-tipo,
        script-data             Script-data
}

Script-tipo ::= CHOICE
{
        ponto-de-encontro       GraphicString ::= ‘pe’,
        lista-de-operacoes      GraphicString ::= ‘lo’,
        externo         GraphicString
}

Script-data ::= SEQUENCE OF GraphicString

END
```

-- 5 Module Ponto-de-encontro

```
NCM-Ponto-de-encontro {ponto-de-encontro(-- 5)}

DEFINITIONS ::= BEGIN

EXPORTS
                Ponto-de-encontro
                Valor-estado,
                Operador-logico,
                Retardo,
                Condicao-previa,
                Condicao-corrente,
                Acoes;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)}
                Extremidade
```

```
                    FROM NCM-Elo  {elo(-- 3)}
            Script-tipo
                    FROM NCM-Script  {script(-- 4)};

Ponto-de-encontro ::= SEQUENCE
{
        pe-id                   Uid
        script-tipo             Script-tipo ::= 'pe',
        condicoes-pe            Condicoes-pe,
        acoes-pe        Acoes,
}

Condicoes-pe ::= CHOICE
{
        combinacao-logica               Combinacao-logica,
        condicao-generica               Condicao-generica,
}

Combinacao-logica ::= SEQUENCE
{
        operador-logico         Operador-logico OPTIONAL,
        condicoes                       SEQUENCE OF Condicoes-pe,
}

Condicao-generica ::= SEQUENCE
{
        valor-origem            Valor-origem,
        retardo                 Retardo OPTIONAL,
        condicao-previa  Condicao-previa,
        condicao-corrente       Condicao-corrente,
}

Valor-origem ::= SEQUENCE
{
        extremidade             Extremidade,
        valor-estado            Valor-estado
}

Valor-estado ::= CHOICE
{
        valor-estado-apr        Valor-estado-apr,
        valor-estado-sel        Valor-estado-sel,
        valor-estado-atr        Valor-estado-atr
}

Valor-estado-apr ::= ENUMERATED
{
        dormindo        (1),
        preparado       (2),
        ocorrendo       (3),
        suspenso        (4)
}

Valor-estado-sel ::= ENUMERATED
{
```

```
        preparado       (1),
        ocorrendo       (2)
}

Valor-estado-atr ::= ENUMERATED
{
        preparado       (1),
        ocorrendo       (2)
}

Condicao-previa ::= CHOICE
{
        condicao-avaliada       Condicao-Avaliada,
        verdade         BOOLEAN     ::= TRUE
}

Condicao-corrente ::= CHOICE
{
        condicao-avaliada       Condicao-Avaliada,
        verdade         BOOLEAN     ::= TRUE
}

Condicao-Avaliada ::= SEQUENCE
{
        operador-comparacao             Operador-comparacao
        valor-comparacao                Valor-estado
}

Acoes ::= SEQUENCE OF Acao

Acao ::= CHOICE

Acao ::= SEQUENCE
{
        extremidade     Extremidade,
        retardo         Retardo OPTIONAL,
        acao            Acao-tipo
}

Retardo ::= INTEGER

Acao-tipo ::=   'inicia' | 'suspende' | 'reassume' | 'termina' | 'prepara' |  'ativa' |  'habilita' |
                'inibe' | 'aguarde' |  'atribui'

Operador-logico ::= ENUMERATED
{
        and     (1),
        or      (2),
        not     (3)
}

Operador-comparacao ::= ENUMERATED
{
        igual                   (1),
        diferente               (2),
```

```
        maior                   (3),
        maior-ou-igual  (4),
        menor                   (5),
        menor-ou-igual  (6),
}

END
```

-- 6 Module Lista-de-operacoes

```
NCM-Lista-de-operacoes {lista-de-operacoes(-- 6)}

DEFINITIONS ::= BEGIN

EXPORTS
                Lista-de-operacoes;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)}
                Script-tipo
                        FROM NCM-Script  {script(-- 4)}
                Ancora
                        FROM NCM-Ancora  {ancora(-- 2)}
                Tipo-evento
                        FROM NCM-Elo  {elo(-- 3)}
                Valor-estado,
                Operador-logico,
                Retardo,
                Condicao-previa,
                Condicao-corrente,
                Acoes
                        FROM NCM-Ponto-de-encontro  {ponto-de-encontro(-- 5)};

Lista-de-operacoes ::= SEQUENCE
{
        lo-id                   Uid,
        script-tipo             Script-tipo ::= 'lo',
        lo                      L-operacoes
}

L-operacoes ::= SEQUENCE OF Operacoes

Operacoes ::= SEQUENCE
{
        condicoes-lo            Condicoes-lo,
        acoes-lo        Acoes,
}

Condicoes-lo ::= CHOICE
{
        combinacao-logica-lo            Combinacao-logica-lo,
        condicao-generica-lo            Condicao-generica-lo,
}

Combinacao-logica-lo ::= SEQUENCE
```

```
{
        operador-logico-lo              Operador-logico OPTIONAL,
        condicoes-lo                    SEQUENCE OF Condicoes-lo,
}

Condicao-generica-lo ::= SEQUENCE
{
        valor-origem-lo   Valor-origem-lo,
        retardo                 Retardo OPTIONAL,
        condicao-previa   Condicao-previa,
        condicao-corrente       Condicao-corrente,
}

Valor-origem-lo ::= SEQUENCE
{
        ancora                  Ancora,
        tipo-evento             Tipo-evento
        valor-estado            Valor-estado
}

END
```

-- 7 Module No

```
NCM-No  {no(-- 7)}

DEFINITIONS ::= BEGIN

EXPORTS
                No,
                No-id,
                No-descritor
                No-ancoras;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)}
                Descritor
                        FROM NCM-Descritor  {descritor(-- 9)};

No ::= SEQUENCE
{
        no-id                   No-id,
        no-ancoras              No-ancoras,
        descritor               No-descritor
}

No-id ::= Uid

No-ancoras ::= SEQUENCE OF Ancoras

No-descritor ::= Descritor

END
```

-- 8 Module NoTerminal

```
NCM-NoTerminal  {noTerminal(-- 8)}

DEFINITIONS ::= BEGIN

EXPORTS
                NoTerminal;

IMPORTS
                No-id,
                No-ancoras,
                No-descritor
                        FROM NCM-No  {no(-- 7)};

NoTerminal ::= SEQUENCE
{
        noTerminal-id                   No-id,
        noTerminal-tipo         NoTerminal-tipo,
        noTerminal-ancoras              No-ancoras,
        noTerminal-dados                NoTerminal-Dados,
        descritor-no                    No-descritor
}

NoTerminal-tipo ::= ENUMERATE
{
        texto   (1),
        audio   (2),
        video   (3)
}

NoTerminal-dados ::= CHOICE
{
        dados-incluido  s       Dados-incluidos,
        dados-referencia Referencia-externa
}

Dados-incluidos ::= CHOICE
{
        octet-string    OctetString
        bit-strint      BitString
}

Referencia-externa ::= GraficString

END
```

-- 9 Module NoComposicao

```
NCM-NoComposicao  {noComposicao(-- 9)}

DEFINITIONS ::= BEGIN

EXPORTS
                NoComposicao;

IMPORTS
                No-id,
```

```
                No-ancoras,
                No-descritor
                        FROM NCM-No  {no(-- 7)}
                Elo
                        FROM NCM-Elo  {elo(-- 3)}
                NoTerminal
                        FROM NCM-No  {noTerminal(-- 8)},

NoComposicao ::= SEQUENCE
{
        noComposicao-id                 No-id,
        descritor-composicao            No-descritor,
        noComposicao-ancoras        No-ancoras,
        elos                            Elos,
        elementos                       Elementos
}

Elos ::= SEQUENCE OF Elo

Elementos ::= SEQUENCE OF Elemento

Elemento ::= SEQUENCE
{
        descritor-elemento      No-descritor    OPTIONAL,
        componente              Componente
}

Componente ::= CHOICE
{
        noTerminal              NoTerminal,
        noComposicao    NoComposicao
}

END
```

-- 10 Module NoContexto

```
NCM-NoContexto  {noContexto(-- 10)}

DEFINITIONS ::= BEGIN

EXPORTS
                NoContexto,
                NoContexto-id;

IMPORTS
                No-id,
                No-ancoras,
                No-descritor
                        FROM NCM-No  {no(-- 7)}
                Elos
                        FROM NCM-NoComposicao  {noComposicao(-- 9)}
                NoTerminal
                        FROM NCM-No  {noTerminal(-- 8)},

NoContexto ::= SEQUENCE
```

```
{
        noContexto-id                   No-id,
        descritor-Contexto              No-descritor,
        noContexto-ancoras              No-ancoras,
        elos                            Elos,
        contexto-elementos              Contexto-elementos
}

Contexto-elementos ::= SET OF Contexto-elemento

Contexto-elemento::= SEQUENCE
{
        descritor-elemento      No-descritor    OPTIONAL,
        componente              Componente
}

Componente ::= CHOICE
{
        noTerminal              NoTerminal,
        noComposicao    NoComposicao
}

END
```

-- 11 Module NoContextoUsuario

```
NCM-NoContextoUsuario {noContextoUsuario(-- 15)}

DEFINITIONS ::= BEGIN

EXPORTS
                NoContextoUsuario;

IMPORTS
                NoContexto
                        FROM NCM-NoContexto  {noContexto(-- 10)}

noContextoUsuario       NoContextoUsuario       ::=     NoContexto

END
```

-- 12 Module NoContextoVersoes

```
NCM-NoContextoVersoes {noContextoVersoes(-- 12)}

DEFINITIONS ::= BEGIN

EXPORTS
                NoContextoVersoes;

IMPORTS
                NoContexto
                        FROM NCM-NoContexto  {noContexto(-- 10)}

noContextoVersoes       NoContextoVersoes       ::=     NoContexto
```

```
END
```

-- 13 Module Trilha

```
NCM-Trilha  {trilha(-- 13)}

DEFINITIONS ::= BEGIN

EXPORTS
                Trilha;

IMPORTS
                No-id,
                No-ancoras,
                No-descritor
                        FROM NCM-No  {no(-- 7)}
                NoContexto-id
                        FROM NCM-NoContexto  {noContexto(-- 10)}
                Elos
                        FROM NCM-NoComposicao  {noComposicao(-- 9)}
                Perspectiva
                        FROM NCM-Elo  {elo(-- 3)};

Trilha ::= SEQUENCE
{
        trilha-id               No-id,
        contexto-associado              NoContexto-id,
        entidade-corrente               Trilha-elemento
        descritor-trilha                No-descritor,
        trilha-ancoras                  No-ancoras,
        trilha-elos                     Elos,
        trilha-elementos        Trilha-Elementos
}

Trilha-Elementos ::= SEQUENCE OF Trilha-elemento

Trilha-elemento ::= Perspectiva

END
```

-- 14 Module BasePrivada

```
NCM-BasePrivada  {basePrivada(-- 14)}

DEFINITIONS ::= BEGIN

EXPORTS
                BasePrivada,

IMPORTS
                No-id,
                No-ancoras,
                No-descritor
                        FROM NCM-No  {no(-- 7)}
                Elos
                        FROM NCM-NoComposicao  {noComposicao(-- 9)}
```

```
                NoContextoUsuario
                        FROM NCM-NoContextoUsuario  {noContextoUsuario(-- 11)}
                NoTerminal
                        FROM NCM-No  {noTerminal(-- 8)},

BasePrivada ::= SEQUENCE
{
        basePrivada-id                  No-id,
        descritor-basePrivada           No-descritor,
        basePrivada-ancoras             No-ancoras,
        basePrivada-elos        Elos,
        basePrivada-elementos           BasePrivada-elementos
}

BasePrivada-elementos ::= SET OF BasePrivada-elemento

BasePrivada-elemento ::= CHOICE
{
        noTerminal              NoTerminal,
        noContextoUsuario       NoContextoUsuario,
        base-privada            BasePrivada
}

END
```

-- 15 Module HiperbasePublica

```
NCM-HiperbasePublica  {hiperbasePublica(-- 15)}

DEFINITIONS ::= BEGIN

EXPORTS
                HiperbasePublica;

IMPORTS
                NoContexto
                        FROM NCM-NoContexto  {noContexto(-- 10)}

hibperbasePublica       HiperbasePublica        ::=     NoContexto

END
```

-- 16 Module Anotacao

```
NCM-Anotacao  {anotacao(-- 14)}

DEFINITIONS ::= BEGIN

EXPORTS
                Anotacao,

IMPORTS
                No-id,
                No-ancoras,
                No-descritor
                        FROM NCM-No  {no(-- 7)}
```

```
                Elos
                        FROM NCM-NoComposicao  {noComposicao(-- 9)}
                NoContextoUsuario
                        FROM NCM-NoContextoUsuario  {noContextoUsuario(-- 11)}
                Trilha
                        FROM NCM-Trilha  {trilha (-- 13)}
                NoTerminal
                        FROM NCM-No  {noTerminal(-- 8)},

Anotacao ::= SEQUENCE
{
        anotacao-id                     No-id,
        descritor-anotacao              No-descritor,
        anotacao-ancoras        No-ancoras,
        anotacao-elos                   Elos,
        anotacao-elementos              Anotacao-elementos
}

Anotacao-elementos ::= SET OF Anotacao-elemento

Anotacao-elemento ::= CHOICE
{
        noTerminal              NoTerminal,
        noContextoUsuario       NoContextoUsuario,
        trilha                  Trilha
}

END
```

-- 17 Module Descritor

```
NCM-Descritor  {descritor(-- 9)}

DEFINITIONS ::= BEGIN

EXPORTS
                Descritor;

IMPORTS
                Uid
                        FROM NCM-Entidade  {entidade(-- 1)}
                Referencia-externa
                        FROM NCM-NoTerminal  {noTerminal(-- 8)}
                Lista-de-operaces
                        FROM NCM-Lista-de-operacoes  {lista-de-operacoes(-- 6)};

Descritor ::= SEQUENCE
{
        descritor-id            Descritor-id,
        iniciacao               Lista-de-operacoes,
        termino         Lista-de-operacoes,
        operacoes               Lista-de-operacoes
        exibidor        Referencia-externa,
        posicao         Posicao  OPTIONAL,
        tamanho                 Tamanho         OPTIONAL,
        duracao         INTEGER
```

```
}

Descritor-id ::= Uid

Posicao ::= SEQUENCE
{
        posicao-x               INTEGER,
        posicao-y               INTEGER,
        posicao-z               INTEGER,
}

Tamanho ::= SEQUENCE
{
        tamanho-x               INTEGER,
        tamanho-y               INTEGER,
        tamanho-z               INTEGER,
}

END
```

Anexo B

Especificação Formal da Semântica do NCM

7.3. Exemplo de especificação de um documento NCM usando RT-LOTOS

Nesta seção um documento NCM exemplo é especificado formalmente usando a técnica de descrição formal RT-LOTOS.

7.3.1. Descrição do Documento

A apresentação do documento (nó de composição A) começa com a exibição de uma representação pictorial do próprio nó A (o browser de estrutura mostra um mapa com a estrutura do documento, um desenho semelhante ao da Figura 1). O início da exibição do nó A (ocorrência do evento Ae1) provoca o disparo do elo l_4 e como consequência são iniciadas as apresentações dos nós D (perspectiva AD) e F (perspectiva AF). Ao longo da exibição do nó A, se o usuário selecionar uma região do mapa uma outra representação, ou versão, do nó D (perspectiva ABD) é exibida. O final da apresentação da representação do nó ABD (evento ABDe2) faz com que o elo l1 dispare e o no C (perspectiva ABC) seja exibido.
Durante a exibição do nó AD, se o usuário selecionar uma âncora (evento ADe2), o elo l_5 dispara provocando o início da exibição do nó E (perspectiva AE). Depois que o nó F (perspectiva AF) acaba de ser exibido, caso o usuário tenha interagido durante a exibição de AD, isto é, se os eventos AFe2 e ADe2 tenham ambos ocorridos em qualquer ordem, a condição de disparo do elo l_2 é satisfeita, porém o evento de destino AGe1 só deve ocorrer 10 segundos após. Nesse caso provocando o início da exibição do nó G (perspectiva AG). Ao final da exibição do nó G o elo l_3 dispara e uma nova exibição do nó D (perspectiva ABD) é iniciada. Após o final desta nova versão do nó D o elo l_1 dispara e o nó C (perspectiva ABC) é exibido também.
Ao longo da exibição do nó AE, caso ocorra o evento AEe4 (interação com usuário) ou o evento AEe2 (evento previsível), uma representação da composição AH começa a ser exibida. O início da apresentação de AH implica no disparo do elo l_9 que implica na exibição do nó AHI. Durante a exibição do nó AHI, caso ocorra o evento imprevisível AHIe2 (interação com usuário), o nó AHJ é exibido. Ao final da exibição de AHJ, o evento AHJe2 força o término da representação da composição AH que contém a representação do nó AHI que terminou.
O final da apresentação da representação pictorial do nó A, marcado pelo evento Ae3 força o final da apresentação da composição e consequentemente do documento.

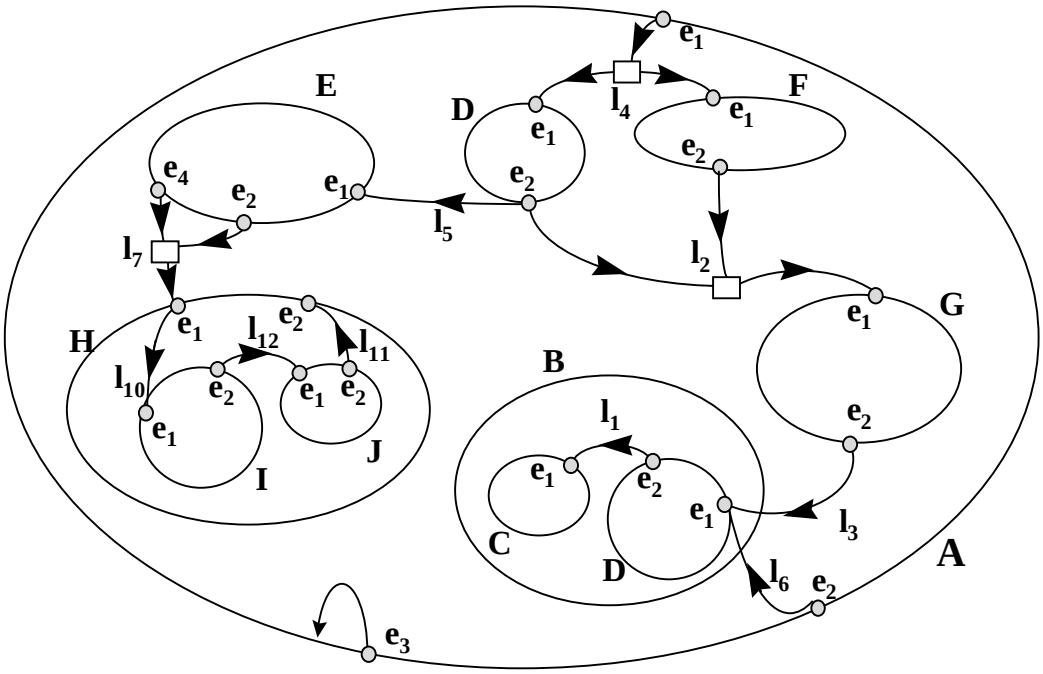


Figura 1 - Estrutura do documento

7.3.2. Especificação RT-LOTOS do Documento

```
specification NCM [userA, userAD, userAEe4, userAEe5, userAHI] : exit
type natural is boolean
  sorts nat
  opns
    +   :nat,nat->nat
    -   :nat,nat->nat
    *   :nat,nat->nat
    min :nat,nat->nat
    max :nat,nat->nat
    <   :nat,nat->bool
    >   :nat,nat->bool
    <=  :nat,nat->bool
    >=  :nat,nat->bool
    div  :nat,nat->nat
    mod  :nat,nat->nat
    divs :nat,nat->nat
endtype

behaviour
  let dnoA : time=40 in
  let lnoA : time=0 in
  let delonoA : time=50 in
  let lelonoA : time = 0 in

 hide startA, endA, stopA in
```

```
compA [startA, endA, stopA, userA,userAD, userAEe4, userAEe5, userAHI] (dnoA, lnoA)
|[ startA, endA, stopA, userA, userAD, userAEe4, userAEe5, userAHI]|
(
        Users [userA,userAD, userAEe4, userAEe5, userAHI]
        |||
        ( startA !0 !0 ; endA ; Loop [startA, endA] (delonoA, lelonoA) )
 )

where


process Users [user1, user2, user3, user4, user5] : noexit:=
        Interaction [user1]
        |||
        Interaction [user2]
        |||
        Interaction [user3]
        |||
        Interaction [user4]
        |||
        Interaction [user5]
endproc


process Interaction [user]: noexit:=
 user ;  delay(1) Interaction [user]
endproc

process Loop [e1, e2] (delonoA, lelonoA:nat) : noexit :=
 e1 !delonoA !lelonoA ;  e2 ; Loop [e1, e2] (delonoA, lelonoA)
endproc
process  compA [start, end, req_stop, userA, userAD, userAEe4, userAEe5, userAHI]
                                                                (dno, lno: nat) : exit :=
hide start_b in
start ?delo:nat ?lelo:nat ;
(
        (
            (
                (  start_b ; end ; exit )
                |[start_b, end]|
                (
                        [delo > 0] -> body_compA [start_b, end, userA, userAD,
                                                userAEe4, userAEe5, userAHI]
                                                                (delo, lelo)
                        []
                        [delo = 0] -> body_compA [start_b, end, userA, userAD,
                                                userAEe4, userAEe5, userAHI]
                                                                (dno, lno)
                )
            ) [> req_stop ; end ; exit
```

```
        )
        ||| compA [start, end, req_stop, userA,userAD, userAEe4, userAEe5, userAHI] (dno, lno)
)
endproc

process  body_compA [start, end, userA,userAD, userAEe4, userAEe5, userAHI]
                                                                    (dnoA, lnoA: nat) : exit :=
  (* os parametros dno definem a duracao do no', periodo de tempo entre o start e o end*)
  (* os parametros de definem o delay entre o inicio do no' - start - e a ocorrencia do evento *)
  let dAe2 : time=20 in
  let lAe2: time=1 in
  let dnoAD : time=20 in
  let dADe2 : time=10 in
  let lADe2 : time=5 in
  let dnoAE : time=15 in
  let dAEe2 : time=8 in
  let dAEe3 : time=10 in
  let dAEe4 : time=5 in
  let lAEe4 : time=2 in
  let dAEe5 : time=11 in
  let lAEe5 : time=3 in
  let dnoAF : time=12 in
  let dnoAG : time=5 in
  let dl2 : time=5 in
  let dl3 : time=7 in
  let dnoABDl6 : time=15 in
  let dnoABDl3 : time=10 in
  let dl8 : time=25 in
```

```
  hide    Ae1, Ae3, stopA, startAB, endAB, stopAB, ABDe1, startAH, endAH, stopAH,
ADe1, ADe3, stopAD, AEe1, AEe6, stopAE, AEe2, AEe3, AFe1, AFe2, stopAF,
AGe1, AGe2, stopAG, req_end   in
(
        (
                (
                        no0_1 [Ae1, Ae3, stopA, userA] (dnoA,lnoA,dAe2,lAe2)          (* no' A *)
                        |||
                        compAB [startAB, endAB, stopAB, ABDe1] (0,0)        (*composicaoAB
*)
                        |||
                        compAH [startAH, endAH, stopAH, userAHI](0,0) (*composicao AH*)
                        |||
                        no0_1 [ADe1, ADe3, stopAD, userAD] (dnoAD,0,dADe2,lADe2)
                                                                        (* no' AD *)
                        |||
                        no2_2 [AEe1, AEe6, stopAE, AEe2, AEe3, userAEe4, userAEe5]
                                (dnoAE,0,dAEe2,0,dAEe3,0,dAEe4,lAEe4,dAEe5,lAEe5)
                                                                        (* no' AE *)
                    |||
                        no0_0 [AFe1, AFe2, stopAF] (dnoAF, 0)                (* no' AF*)
                        |||
                        no0_0 [AGe1, AGe2, stopAG] (dnoAG, 0)                (* no' AG*)
                )
        |[ Ae1, AFe1, ADe1, startAB, Ae3, AFe2, userAD, AGe1, AEe1,
           AGe2, userA, ABDe1, userAEe4, AEe2, startAH, userAEe5,
           stopAB, stopAH, stopA, stopAD, stopAE, stopAF, stopAG ]|
                (
                        elo1_v4 [start, Ae1, AFe1, ADe1, startAB] (0,0,0,0,0,0,0,0,0,0,0,0,0)
                                                                        (*elo lstart*)
                        |||
                        elo1_1 [Ae3, req_end] (0)                        (* elo lend*)
                        |||
                        (       elo2_and_v1 [userAD, AFe2, AGe1] (0,0,dl2,0,0)          (* elo l2*)
                                |[userAD]|
                                elo1_v1 [userAD, AEe1] (0,0,0)                          (* elo l5*)
                        )
                    |||
                        elo1_v1 [AGe2, ABDe1] (dl3, dnoABDl3,0)                (* elo l3*)
                        |||
                        elo1_v1 [userA, ABDe1] (0, dnoABDl6,0)                        (* elo l6*)
                        |||
                        elo2_or_v1 [userAEe4, AEe2, startAH] (0,0,0,5,0)        (* elo l7*)
                )
        ) [> end ; exit
 )
|[end, req_end]|
( req_end ; end ; exit)
endproc (* process compA *)
```

```
process  compAB [start,end, req_stop, BDe1] (dno, lno:nat) : noexit :=
   hide start_b in
start ?delo:nat ?lelo:nat ;
(
        (
            (
                ( start_b ; end ; exit )
                |[start_b, end]|
                (
                        [delo > 0] -> body_compAB [start_b, end, BDe1] (delo, lelo)
                        []
                        [delo = 0] -> body_compAB [start_b, end, BDe1] (dno, lno)
                )
            ) [> req_stop ; end ; exit
        )
        ||| compAB [start, end, req_stop, BDe1] (dno, lno)
)
endproc

process body_compAB [start, end, BDe1] (dno, lno:nat) : exit :=
    let dnoBC : time=10 in
    hide BDe2, stopBD, BCe1, BCe2, stopBC, req_end in
 (
        (
                (
                        no0_0 [BDe1, BDe2, stopBD]  (0,0)               (* no' BD *)
                        |||
                        no0_0 [BCe1, BCe2, stopBC] (dnoBC, 0)          (* no' BC *)
                )
        |[BDe2, BCe1, stopBD, stopBC]|
                (
                        elo1_v1 [BDe2, BCe1] (0,0,0)              (* elo l1*)
                )
        )   [> end ; exit
 )
|[end, req_end]|
( req_end ; end ; exit )
endproc (*process body_comAB *)
```

```
process  compAH [start, end, req_stop, HIe2] (dno, lno:nat) : noexit :=
   hide start_b in
start ?delo:nat ?lelo:nat ;
(
          (
               (
                    (  start_b ; end ; exit )
                    |[start_b, end]|
                    (
                              [delo > 0] -> body_compAH [start_b, end, HIe2] (delo, lelo)
                              []
                              [delo = 0] -> body_compAH [start_b, end, HIe2] (dno, lno)
                    )
               ) [> req_stop ; end ; exit
          )
          |||  compAH [start, end, req_stop, HIe2] (dno, lno)
          )
endproc

process body_compAH [start, end, HIe2] (dno, lno:nat) : exit :=
   let dnoHI : time=10 in
   let dHIe2 : time=3 in
   let lHIe2 : time=5 in
   let dnoHJ : time=10 in
 hide HIe1, HIe3, stopHI, HJe1, HJe2, stopHJ,  req_end in
 (
          (
                    (
                              no0_1 [HIe1, HIe3, stopHI, HIe2] (dnoHI,0,dHIe2,lHIe2)   (* no' HI *)
                              |||
                              no0_0 [HJe1, HJe2, stopHJ] (dnoHJ, 0)                              (* no' HJ*)
                    )
          |[ HIe1, HIe2, HJe1, HJe2, stopHI, stopHJ ]|
                    (
                              elo1_v1 [start, HIe1] (0,0,0)      (* elo start*)
                              |||
                              elo1_1 [HJe2, req_end] (0)                  (* elo end*)
                              |||
                              elo1_v1 [HIe2, HJe1] (0,0,0)      (* elo l12*)
                    )
          )   [> end ; exit
 )
|[end, req_end]|
(  req_end ; end ; exit )
endproc (*process body_comAH *)




process no0_0 [start, end, req_stop] (dno: nat, lno: nat): exit :=
```

```
        start ?delo: nat ?lelo:nat;
        (
                (
                    (
                        [delo > 0] -> body_no0_0 [end] (delo, lelo)
                        []
                        [delo = 0] -> body_no0_0 [end] (dno, lno)
                    )  [> req_stop ; end ; exit
                )
                |||
                no0_0 [start, end, req_stop] (dno, lno)
        )
endproc

process body_no0_0 [end] (d: nat, l: nat) : exit :=
delay (d) end{l} ; exit
endproc


process no0_1 [start, end, req_stop, e1] (dno:nat, lno:nat, d1:nat, l1:nat): exit :=
        start ?delo: nat ?lelo:nat;
        (
                (
                    (
                        [delo > 0] -> body_no0_1 [end,e1] (delo, lelo, d1,l1)
                        []
                        [delo = 0] -> body_no0_1 [end,e1] (dno, lno,d1,l1)
                    )   [>  req_stop ; end ; exit
                )
                |||
                no0_1 [start, end, req_stop, e1] (dno, lno,d1,l1)
        )
endproc
```

```
process body_no0_1 [end,e1] (d:nat, l:nat, d1:nat, l1:nat): exit :=
        hide end_no in
        (
                (
                        ( delay (d) end{l} ; exit )
                        |||
                        anchor [e1] (d1,l1)
                ) [> end_no ; exit
        )
        |[end_no, end]|
        (   end ; end_no ; exit   )
endproc


process no2_2 [start, end, req_stop, e1, e2, e3, e4]
                                (dno, lno, d1, l1, d2, l2, d3, l3, d4, l4:nat): exit :=
start ?delo: nat ?lelo:nat;
(
        (
            (
                [delo > 0] -> body_no2_2 [end, e1, e2, e3, e4]
                                                (delo, lelo, d1,l1, d2, l2, d3, l3, d4, l4)
                []
                [delo = 0] -> body_no2_2 [end, e1, e2, e3, e4]
                                                (dno, lno, d1,l1, d2, l2, d3, l3, d4, l4)
            )   [>  req_stop ; end ; exit
        )
        |||
        no2_2 [start, end, req_stop, e1, e2, e3, e4] (dno, lno,d1,l1, d2, l2, d3, l3, d4, l4)
)
endproc
```

```
process body_no2_2 [end,e1, e2, e3, e4] (dno, lno, d1, l1, d2, l2, d3, l3, d4, l4:nat): exit :=
        hide end_no in
        (
                (
                        ( delay (d1) e1{l1} ; exit )
                        |||
                        ( delay (d2) e2{l2} ; exit )
                        |||
                        anchor [e3] (d3,l3)
                        |||
                        anchor [e4] (d4,l4)
                        |||
                        ( delay (dno) end{lno} ; exit )
                ) [> end_no ; exit
        )
        |[end_no, end]|
(   end ; end_no ; exit   )
endproc


process anchor [user] (wait,window:nat) : exit :=
  delay(wait) (user{window}; exit [] delay(window) exit)
endproc


process r_anchor [user] (wait,window:nat) : exit :=
  delay(wait) anytime[user] [> delay(window) i; exit
  where

  process anytime[user] : noexit :=
    user; anytime[user]
  endproc

endproc


process elo1_1 [e1, e2] (d: nat) : noexit :=
   e1; delay (d) e2{0}; elo1_1 [e1,e2] (d)
endproc

process elo1_v1 [e1, e2] (d: nat, d_target: nat, l_target: nat) : noexit :=
   e1; delay (d) e2{0} !d_target !l_target ; elo1_v1 [e1,e2] (d,d_target,l_target)
endproc
```

```
process elo1_2 [e1, e2, e3] (d1, d2, d3: nat) : exit :=
        e1 ;
        delay(d1) (
                        (delay(d2) e2{0} ; exit )
                        |||
                        (delay(d3) e3{0} ; exit )
                )
         >> elo1_v2 [e1, e2, e3] (d1, d2, d3)
endproc


process elo1_v2 [e1, e2, e3] (d1, d2, d3, d_target2, l_target2, d_target3, l_target3: nat) : exit :=
        e1 ;
        delay(d1) (
                        (delay(d2) e2{0} ! d_target2 ! l_target2 ; exit )
                        |||
                        (delay(d3) e3{0} ! d_target3 ! l_target3; exit )
                )
         >> elo1_v2 [e1, e2, e3] (d1, d2, d3, d_target2, l_target2, d_target3, l_target3)
endproc


process elo1_v3 [e1, e2, e3, e4] (d1, d2, d3, d4, d_target2, l_target2,
                                d_target3, l_target3, d_target4, l_target4: nat) : exit :=
        e1 ;
        delay(d1) (
                        (delay(d2) e2{0} ! d_target2 ! l_target2 ; exit )
                        |||
                        (delay(d3) e3{0} ! d_target3 ! l_target3; exit )
                        |||
                        (delay(d4) e4{0} ! d_target4 ! l_target4; exit )
                )
         >> elo1_v3 [e1, e2, e3, e4] (d1, d2, d3, d4, d_target2, l_target2,
                                        d_target3, l_target3, d_target4, l_target4)
endproc
```

```
process elo1_v4 [e1, e2, e3, e4, e5] (d1, d2, d3, d4, d5, d_target2, l_target2,
                                      d_target3, l_target3, d_target4, l_target4,
                                      d_target5, l_target5: nat) : exit :=
        e1 ;
        delay(d1) (
                        (delay(d2) e2{0} ! d_target2 ! l_target2 ; exit )
                        |||
                        (delay(d3) e3{0} ! d_target3 ! l_target3 ; exit )
                        |||
                        (delay(d4) e4{0} ! d_target4 ! l_target4 ; exit )
                        |||
                        (delay(d5) e5{0} ! d_target5 ! l_target5 ; exit )
                )
         >> elo1_v4 [e1, e2, e3, e4, e5] (d1, d2, d3, d4, d5, d_target2, l_target2,
                                          d_target3, l_target3, d_target4, l_target4,
                                          d_target5, l_target5)
endproc


process elo2_and_v1 [e1, e2, e3] (d1: nat, d2: nat, d3: nat, d_target: nat, l_target: nat) : exit :=
(  ( e1 ; delay(d1) exit)  ||| ( e2  ; delay(d2) exit ) )   >> delay(d3) e3{0} !d_target !l_target ;
elo2_and_v1 [e1, e2, e3] (d1, d2, d3,d_target,l_target)
endproc


process elo2_or_1 [e1, e2, e3] (d1, d2, d3: nat) : noexit :=
        (   (  e1 ; delay(d1) exit  )    []   (  e2 ; delay(d2) exit ) )   >>  delay(d3) e3{0}  ;
        elo2_or_1 [e1, e2, e3] (d1, d2, d3)
endproc


process elo2_or_v1 [e1, e2, e3] (d1, d2, d3, d_target, l_target: nat) : noexit :=
        (  ( e1 ; delay(d1) exit )  []  (  e2 ; delay(d2) exit  ) )
        >>  delay(d3) e3{0} !d_target !l_target ;
        elo2_or_v1 [e1, e2, e3] (d1, d2, d3, d_target, l_target)
endproc


endspec
```

7.3.3. Resultado de uma Simulação da Apresentação Exemplo

Time Action State
0 i(startA<0,0>) 1
0 i(start_b) 2
0 i(startAB<0,0>) 3
0 i(Ae1<0,0>) 4
0 i(ADe1<0,0>) 5
0 i(AFe1<0,0>) 6
0 i(exit) 7
0 10 8
10 2 9
12 i(AFe2) 10
12 userAD 11
12 i(exit) 12
12 i(AEe1<0,0>) 13
12 1 14
13 4 15
17 i(AGe1<0,0>) 16
17 userAEe4 17
17 i(exit) 18
17 i(startAH<5,0>) 19
17 i(start_b) 20
17 i(HIe1<0,0>) 21
17 1 22
18 2 23
20 i(AEe2) 24
20 i(ADe3) 25
20 i(exit) 26
20 i(end_no) 27
20 i(startAH<5,0>) 28
20 i(start_b) 29
20 i(HIe1<0,0>) 30
20 userA 31
20 i(ABDe1<15,0>) 32
20 userAHI 33
20 i(HJe1<0,0>) 34
20 1 35
21 1 36
22 i(AGe2) 37
22 i(AEe3) 38
22 1 39
23 3 40
26 1 41
27 i(AEe6) 42
27 i(end_no) 43
27 i(HIe3) 44
27 i(end_no) 45
27 userAHI 46
27 i(HJe1<0,0>) 47
27 1 48
28 1 49
29 i(ABDe1<10,0>) 50
29 1 51
30 i(HJe2) 52
30 i(HIe3) 53
30 i(req_end) 54
30 i(end_no) 55
30 i(endAH) 56
30 5 57
35 i(BDe2) 58
35 i(BCe1<0,0>) 59
35 2 60
37 i(HJe2) 61
37 i(req_end) 62
37 i(endAH) 63
37 2 64
39 i(BDe2) 65
39 i(BCe1<0,0>) 66
39 1 67
40 i(Ae3) 68
40 i(end_no) 69
40 i(req_end) 70
40 i(endA) 71
40 i(startA<50,0>) 72
40 i(start_b) 73
40 i(AFe1<0,0>) 74
40 i(ADe1<0,0>) 75
40 i(Ae1<0,0>) 76
40 i(startAB<0,0>) 77
40 i(exit) 78
40 10 79
50 2 80
52 i(AFe2) 81
52 3 82
55 5 83
60 i(ADe3) 84
60 i(end_no) 85
60 userA 86
60 i(ABDe1<15,0>) 87
60 1 88
61 14 89
75 i(BDe2) 90
75 i(BCe1<0,0>) 91
75 10 92
85 i(BCe2) 93
85 5 94
90 i(Ae3) 95
90 i(end_no) 96
90 i(req_end) 97
90 i(endA) 98
90 i(startA<50,0>) 99
90 i(start_b) 100
90 i(ADe1<0,0>) 101
90 i(Ae1<0,0>) 102
90 i(startAB<0,0>) 103
90 i(AFe1<0,0>) 104
90 i(exit) 105
90 10 106
100 2 107
102 i(AFe2) 108
102 userAD 109
102 i(AEe1<0,0>) 110
102 i(exit) 111
102 1 112
103 4 113
107 i(AGe1<0,0>) 114
107 2 115
109 userAEe4 116
109 i(exit) 117
109 i(startAH<5,0>) 118
109 i(start_b) 119
109 i(HIe1<0,0>) 120
109 1 121
110 i(AEe2) 122
110 i(ADe3) 123
110 i(exit) 124
110 i(end_no) 125
110 i(startAH<5,0>) 126
110 i(start_b) 127
110 i(HIe1<0,0>) 128
110 1 129
111 1 130
112 i(AGe2) 131
112 i(AEe3) 132
112 userAHI 133
112 i(HJe1<0,0>) 134
112 1 135
113 3 136
116 userAHI 137
116 i(HJe1<0,0>) 138
116 1 139
117 i(AEe6) 140
117 i(end_no) 141
117 2 142
119 i(HIe3) 143
119 i(end_no) 144
119 i(ABDe1<10,0>) 145
119 1 146
120 i(HIe3) 147
120 i(end_no) 148
120 2 149
122 i(HJe2) 150
122 i(req_end) 151
122 i(endAH) 152
122 4 153
126 i(HJe2) 154
126 i(req_end) 155

126 i(endAH) 156
126 3 157
129 i(BDe2) 158
129 i(BCe1<0,0>) 159
129 10 160
139 i(BCe2) 161
139 1 162
140 i(Ae3) 163
140 i(req_end) 164
140 i(endA) 165
140 i(startA<50,0>) 166
140 i(start_b) 167
140 i(AFe1<0,0>) 168
140 i(ADe1<0,0>) 169
140 i(startAB<0,0>) 170
140 i(Ae1<0,0>) 171
140 i(exit) 172
140 10 173
150 userAD 174
150 i(AEe1<0,0>) 175
150 1 176
151 1 177
152 i(AFe2) 178
152 i(exit) 179
152 3 180
155 2 181
157 i(AGe1<0,0>) 182
157 userAEe4 183
157 i(exit) 184
157 i(startAH<5,0>) 185
157 i(start_b) 186
157 i(HIe1<0,0>) 187
157 1 188
158 i(AEe2) 189
158 i(exit) 190
158 i(startAH<5,0>) 191
158 i(start_b) 192
158 i(HIe1<0,0>) 193
158 2 194
160 i(ADe3) 195
160 i(end_no) 196
160 i(AEe3) 197
160 1 198
161 userAHI 199
161 i(HJe1<0,0>) 200
161 userA 201
161 i(ABDe1<15,0>) 202
161 1 203
162 i(AGe2) 204
162 2 205
164 userAHI 206
164 i(HJe1<0,0>) 207
164 1 208
165 i(AEe6) 209
165 i(end_no) 210
165 2 211
167 i(HIe3) 212
167 i(end_no) 213
167 1 214
168 i(HIe3) 215
168 i(end_no) 216
168 1 217
169 i(ABDe1<10,0>) 218
169 2 219
171 i(HJe2) 220
171 i(req_end) 221
171 i(endAH) 222
171 3 223
174 i(HJe2) 224
174 i(req_end) 225
174 i(endAH) 226
174 2 227
176 i(BDe2) 228
176 i(BCe1<0,0>) 229
176 3 230
179 i(BDe2) 231
179 i(BCe1<0,0>) 232
179 7 233
186 i(BCe2) 234
186 3 235
189 i(BCe2) 236
189 1 237
190 i(Ae3) 238
190 i(end_no) 239
190 i(req_end) 240
190 i(endA) 241
190 i(startA<0,0>) 1
190 i(start_b) 2
190 i(startAB<0,0>) 3
190 i(Ae1<0,0>) 4
190 i(ADe1<0,0>) 5
190 i(AFe1<0,0>) 6
190 i(exit) 7
190 10 8
200 2 9
202 i(AFe2) 10
202 userAD 11
202 i(exit) 12
202 i(AEe1<0,0>) 13
202 1 14
203 4 15
207 i(AGe1<0,0>) 16
207 userAEe4 17
207 i(exit) 18
207 i(startAH<5,0>) 19
207 i(start_b) 20
207 i(HIe1<0,0>) 21
207 1 22
208 2 23
210 i(AEe2) 24
210 i(ADe3) 25
210 i(exit) 26
210 i(end_no) 27
210 i(startAH<5,0>) 28
210 i(start_b) 29
210 i(HIe1<0,0>) 30
210 userA 31
210 i(ABDe1<15,0>) 32
210 userAHI 33
210 i(HJe1<0,0>) 34
210 1 35
211 1 36
212 i(AGe2) 37
212 i(AEe3) 38
212 1 39
213 3 40
216 1 41
217 i(AEe6) 42
217 i(end_no) 43
217 i(HIe3) 44
217 i(end_no) 45
217 userAHI 46
217 i(HJe1<0,0>) 47
217 1 48
218 1 49
219 i(ABDe1<10,0>) 50
219 1 51
220 i(HJe2) 52
220 i(HIe3) 53
220 i(req_end) 54
220 i(end_no) 55
220 i(endAH) 56
220 5 57
225 i(BDe2) 58
225 i(BCe1<0,0>) 59
225 2 60
227 i(HJe2) 61
227 i(req_end) 62
227 i(endAH) 63
227 2 64
229 i(BDe2) 65
229 i(BCe1<0,0>) 66
229 1 67
235 i(BCe2) 234
235 3 235
239 i(BCe2) 236
239 1 237
240 i(Ae3) 68
240 i(end_no) 69
240 i(req_end) 70
240 i(endA) 71

Referências

[Alle83] Allen, J.F. “Maintaining Knowledge about Temporal Intervals”. Communications of the ACM, November 1983.

[BoBr87] Bolognesi, T.; Brinksma, E. “Introduction to the ISO Specification Language LOTOS”. *Computer Networks and ISDN Systems*, vol 14, n. 1, 1987.

[BHLM92] Blakowski, G.; Hubel, J. Langrehr, U.; Mühlhäuser, M. “Tool Support for the Synchronization and Presentation of Distributed Multimedia”. *Computer Communications*. December 1992.

[BlSt96] Blakowiski, G.; Steinmetz, R. “A Media Synchronization Survey: Reference Model, Specification and Case Studies”. *IEEE Journal on Selected Areas in Communication*. Vol 14, N° 1, Janeiro 1996.

[BoBr87] Bolognesi, T.; Brinksma, E. “Introduction to the ISO Specification Language LOTOS”. *Computer Networks and ISDN Systems*, vol 14, n. 1, 1987.

[BuRL91] Bulterman, D.; van Rossum, G. and van Liere, R. “A Structure for Transportable, Dynamic Multimedia Documents”. *In Proceedings 1991 Summer USENIX Conference*. June 1991.

[Bush45] Bush, V. “As We May Think”. *The Atlantic Monthly*. Julho, 1945.

[BuZe92] Buchanan, M.C.; Zellweger, P.T. "Specifying Temporal Behavior in Hypermedia Documents", *Proceedings of European Conference on Hypertext, ECHT'92*. Milano. December 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. 1993. pp. 341-350

[Casa91] Casanova, M.A.; Tucherman, L.; Lima, M.J.; Rangel Netto, J.L.; Rodriguez, N.R.; Soares, L.F.G. “The Nested Context Model for Hyperdocuments”. *Proceedings of Hypertext ' 91*. Texas. December 1991.

[CGRM92] International Organization for Standardization. “Information Processing Systems — Computer Graphics — Computer Graphics Reference Model (CGRM)”. *ISO/IEC IS 11072*. 1992.

[CMSS96] Costa, F.; Muchaluat, D.; Soares, L.F.G.; Souza, G.L. “Editor Gráfico para Estrutura e Sincronismo de Documentos Multimídia”, *Anais do IX SIBGRAPI*, Caxambu, Outubro 1996.

[CoOl94] Courtiat, J.-P.; De Oliveira, R.C. “About Time non-determinism and Exception Handling in a Temporal Extension of LOTOS”. *In Protocol Specification, Testing and Verification XIV*, Vancouver, Canada, June 1994. Chapman & Hall.

[CoOl95] Courtiat, J.-P.; De Oliveira, R.C. “A Reachability Analisys of RT-LOTOS Specifications”. *In Proceedings of the 8th Intern. Conference on Formal Description Techniques.* Montreal, Canada, Outubro 1995. Chapman & Hall.

[CoOl96] Courtiat, J.-P.; De Oliveira, R.C. “Proving Temporal Consistency in a New Multimedia Synchronization Model”. *ACM Multimedia’96*. 1996.

[CoSS96] Costa, F.; Souza, G.; Soares, L.F.; “Editor Gráfico para Sincronização Temporal e Espacial de Objetos Multimídia/Hipermídia”, *Anais do II Workshop de Sist. Hipermídia Distribuídos*, Fortaleza, Maio, 1996.

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

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

[CSOS97] Courtiat, J.P.; Souza, G.L.; Oliveira, R.C. e Soares, L.F.G. “Usando RT-LOTOS para Verificar a Consistência de Documentos Multimídia em Plataformas Configuráveis”. *Anais do 2° Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos*. Fortaleza - CE. Novembro, 1997. Aceito para publicação.

[DeSc85] Delisle, N.; Schwartz, M. “Neptune: A Hypertext System for CAD Applications”. *Proceedings of ACM SIGMOD '85*. Washington, D.C. May 1985.

[DiSé94] Diaz, M.; Sénac, P. “Time Stream Petri Nets, a Model for Timed Multimedia Information”. *Proc. Of the 15th Int. Conf. On Application and Theory of Petri Nets*, Zaragoza. 1994.

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

[FHHD90] Fountain, A.; Hall, W.; Heath, I.; Davis, H. “Microcosm: An Open Model for Hypermedia wiyh Dynamic Linking”. *Proceedings of ECHT'90*. Cambridge University Press. 1990, pp. 298 -311.

[Fluc95] Fluckiger, F. “Understanding Networked Multimedia: Applications and Technology”. *Prentice Hall*. 1995.

[Furn86] Furnas, G.; Generalized Fisheye Views, *Proceedings of CHI'86 Human Factors in Computing Systems*, Boston, Abril, pp. 16-23, 1986

[FuSt90] Furuta, R. e Stotts, P.D. “Generalizing Hypertext: Domains of the Trellis Model”. *T.S.I.: Technique et Science Informatique*, Vol.9, N° 6. 1990.

[Gloo91] Gloor, P.; CYBERMAP Yet Another Way of Navigating in Hyperspace, *Proceedings of the Third ACM Conference on Hypertext*, pp. 107-121, San Antonio, Texas, Dezembro, 1991

[Hala88] Halasz, F.G. “Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems”. *Communications of the ACM*. Vol.31, N° 7. Julho 1988.

[Hamb72] Hamblin, C. “Instants and Intervals”. *Proceedings of the 1st Conference of the International Society for the Study of Time*, 1972.

[HaBR93] Hardman, L.; Bulterman, D. C. A.; van Rossum, G. “The Amsterdam Hypermedia Model: Extending Hypertext to Support Real Multimedia”, *Hypermedia*, May 1993.

[HaRB93] Hardman, L.; van Rossum, G.; Bulterman, D.C.A. “Structured Multimedia Authoring”. *Proceedings of the First Internation Conference on Multimedia*. Anahein, California, August 1993.

[HaBR94] Hardman, L.; Bulterman, D. C. A.; van Rossum, G. “The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Reference Model”. *Communications of the ACM*, Fevereiro 1994.

[HaSc90] Halasz, F.G.; Schwartz, M. "The Dexter Hypertext Reference Model". *NIST Hypertext Standardization Workshop*. Gaithersburg. January 1990.

[HaSR92] Hamakawa R.; Sakagami H.; Rekimoto J. “Audio and Video Extensions to Graphical User Interface Toolkits”. *Proceedings Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, California. Novembro 1992.

[Hoep94] Hoepner, P. “Synchronizing the Presentation of Multimedia Objects”. *Computer Communication*. Novembro 1994.

[HoSA89] Hodges, M. E., Sasnett, R. M. e Ackerman, M. S. “Athena Muse: a Construction Set for Multimedia Applications”. *IEEE Software*. Janeiro 1989.

[IBM90] IBM Corporation; *Audio Visual Connection User’s Guide and Authoring Language Reference*. Agosto 1990.

[ISO89] ISO IEC JTC1, SC 18, WG 3, AFNOR Expert Group. “Multimedia Synchronization: Definitions and Model, Input Contribution on Time Variant Aspects and Synchronization in ODA-Extensions”. Fevereiro 1989.

[ISO89a] “LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behavior”. ISO 8807, 1989.

[ISO92] ISO IEC. “Hypermedia/Time-based Document Structuring Language — HyTime”. International Standard ISO/IEC IS10744, 1992.

[Karm93] Karmouch, A. “Multimedia Distributed Cooperative System”. *Computer Communication*. Vol 16, N° 9. 1993.

[KiSo95] Kim, M.Y; Song, J. “Multimedia Documents with Elastic Time”. *Proceedings of the ACM Multimedia’95*, San Francisco, CA, Novembro 1995.

[LiGh90] Little, T. D. C.; Ghafoor, A. “Synchronization and Storage Models for Multimedia Objects”. *IEEE Journal on Selected Areas in Communications*, vol.8, no. 3. Abril 1990.

[LiGh91a] Little, T. D. C.; Ghafoor, A. “Scheduling of bandwidth-constrained multimedia traffic”. *Proc. 2nd Int. Workshop On Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany, Novembro 1991.

[LiGh91b] Little, T. D. C.; Ghafoor, A. “Spatio-temporal composition of distributed multimedia objects for value added networks”. *IEEE Computer*, vol 24, no 10. Outubro 1991.

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

[MeRT91] Meghini, C.; Rabitti, F. e Thanos, C. “Conceptual Modeling of Multimedia Documents”. *Computer*. Outubro 1991.

[MHEG95] Multimedia and Hypermedia Information Coding Expert Group — MHEG. “Information Technology — Coded Representation of Multimedia and Hypermedia Information Objects — Part1: Base Notation”. *ISO/IEC DIS 13522-1*. Setembro 1995.

[MiSu91] Misue, K.; Sugiyama, K.; “Multi-viewpoint Display Methods: Formulation and Application to Compound Graphs”. *Proceedings of the Fourth Conference on Human-Computer Interaction*, Stuttgart, F.R. Germany, Setembro 1-6, 1991

[Misu94] Misue K.; D-ABDUCTOR 2.30 User Manual, Institute for Social Information Science. *FUJITSU LABORATORIES LTD.*, Japan, 1994

[MSCS97] Muchaluat, D.C.; Soares, L.F.; Costa, F.R..; Souza; G.L.; “Graphical Structured-Editing of Multimedia Documents with Temporal and Spatial Constraints”. *Proceedings of the Multimedia Modeling Conference — MMM’97*, Singapura. Novembro 1997. Aceito para publicação.

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

[Nels65] Nelson, T. “A File Structure for the Complex, The Changing and The Indeterminate”. *ACM 20th National Conference*. 1965.

[OgHK90] Ogawa, R.; Harada, H. and Kaneko, A. “Scenario-based Hypermedia: A Model and a System”. *First European Conference on Hypertext — ECHT’90*. INRIA France. November 1990.

[PéLi96] Pérez-Luque, M.J., Little, T. “A Temporal Reference Framework for Multimedia Synchronization”. *IEEE Journal on Selected Areas in Communication*. Vol 14, N° 1, Janeiro 1996.

[PREM95a] International Organization for Standardization. “Information Processing Systems — Computer Graphics and Image Processing — Presentation Environments for Multimedia Objects (PREMO) — Part 1: Foundamentals of PREMO”. *Committee Draft ISO/IEC CD 14478-1*. August 1995.

[PREM95b] International Organization for Standardization. “Information Processing Systems — Computer Graphics and Image Processing — Presentation Environments for Multimedia Objects (PREMO) — Part 2: Foundation Component”. *Committee Draft ISO/IEC CD 14478-2*. August 1995.

[PREM95c] International Organization for Standardization. “Information Processing Systems — Computer Graphics and Image Processing — Presentation Environments for Multimedia Objects (PREMO) — Part 3: Modelling, Rendering, and Interaction Component”. *Working Draft ISO/IEC 14478-3*. May 1995.

[PREM94] International Organization for Standardization. “Information Processing Systems — Computer Graphics and Image Processing — Presentation Environments for Multimedia Objects (PREMO) — Part 4: PREMO Multimedia Systems Services”. *Working Draft ISO/IEC 14478-4*. September 1994.

[Pogg85] Pogio, A. et al. “CCWS: A Computer-based Multimedia Information System”. *IEEE Computer*. Outubro 1985.

[Pool91] Poole, L. “QuickTime in motion”. *MACWORLD*. September 1991.

[PuGu90] Puttress, J.J.; Guimarães, N.M. “The Toolkit Approach to Hypermedia”. *Proceedings of European Conference on Hypertex, ECHT'90*. 1990.

[RiSa92] Rizk, A.; Sauter, L. “MultiCard: An Open Hypermedia System”. *Proceedings of European Conference on Hypertex, ECHT'92*. Milano. December 1992.

[RJMB93] van Rossum, G.; Jansen, J.; Mullender, K.S.; Bulterman, D.C.A. “CMIFed: a Presentation Environment for Portable Hypermedia Documents”. *Proceedings of the First International Conference on Multimedia*. Anahein, California, August 1993.

[Rodr97] Rodrigues, R.F. “Formatação Temporal e Espacial no Sistema Hyperprop”. *Dissertação de Mestrado, Departamento de Informática, PUC - Rio, Brasil*. Maio de 1997.

[RoSS97a] Rodrigues, R.F.; Soares, L.F.G.; Souza, G.L. “O Ambiente de Execução do Sistema HyperProp para Apresentação de Documentos Multimídia/Hipermídia”. *III Workshop sobre Sistemas Multimídia e Hipermídia*, São Carlos. Maio 1997.

[RoSS97b] Rodrigues, R.F.; Soares, L.F.G.; Souza, G.L. “Authoring and Formatting of Documents Based on Event-Driven Hypermedia Models”. *Proceedings of the IEEE Conference on Protocols for Multimedia Systems - Multimedia Networking — PROMSMmNet’97*, Santigo - Chile. Novembro 1997.

[SaSh90] Salmony, M.; Shepherd, D. “Extending OSI to Support Synchronization Required by Multimedia Application”. *Computer Communication*; Setembro 1990.

[SCSS97] Souza, G.L.; Courtiat, J.P.; Saibel, C.A.S.; Soares, L.F.G. “Usando RT-LOTOS para Especificar Documentos NCM”. *Anais do 2° Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos*. Fortaleza - CE. Novembro, 1997.

[ScSt90] Schütt, H.A.; Streitz, N.A. “HyperBase: A Hypermedia Engine Based on a Relational Database Management System”. *Proceedings of European Conference on Hypertex, ECHT'90*. 1990.

[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.

[Soar95] Soares, L.F.G.; et al “HyperProp: uma Visão Geral”. *I Workshop em Sistemas Hipermídia Distribuídos*, São Carlos - SP, Julho de 1995.

[SoCC93] Soares, L.F.G.; Casanova, M.A.; Colcher, S. “An Architecture for Hypermedia Systems Using MHEG Standard 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”. *International Journal on Information Systems; Special issue on Multimedia Information Systems.* September 1995.

[SoCR96] Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.R. “Nested Composite Nodes and Version Control in an Open Hypermedia System Revisited”. *I Prêmio Compaq de Estímulo à Pesquisa e Desenvovimento em Informática*. Instituto UNIEMP. São Paulo-SP. Novembro de 1996; pp.99-116.

[SoCS96a] Soares, L.F.; Casanova, M.A.; Souza; G.L.; “Âncoras e Elos em Nós de Composição”. *Anais do XXIII Seminário Integrado de Software e Hardware — SEMISH’96*, Recife - PE, Agosto 1996.

[SoCS96b] Soares, L.F.; Casanova, M.A.; Souza; G.L.; “Anchors and Links for Nested Composite Nodes”. *Proceedings of the Multimedia Modeling Conference — MMM’96*, Toulouse, France. November 1996.

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

[SoSC95] Souza; G.L.; Soares, L.F.; Casanova, M.A. “Synchronization Aspects of a Hypermedia Presentation Model with Composite Nodes” *Electronic Proceedings of the ACM Workshop on Effective Abstractions in Multimedia, in connection with ACM Multimedia’95*, San Francisco, EUA. Novembro 1995.

[SoSo95] Souza; G.L.; Soares, L.F.G. “O Modelo de Apresentação de Documentos do HyperProp”. *III Workshop sobre Sistemas Multimídia e Hipermídia*, São Carlos - SP. Julho 1995.

[SoSo96] Souza; G.L.; Soares, L.F.G. “Edição e Execução de Apresentações de Documentos Hipermídia no HyperProp”. *II Workshop em Sistemas Hipermídia e Multimídia*, Fortaleza - CE, Maio de 1996.

[Souz93] Souza; G.L. “Um Ambiente para Edição e Execução de Apresentações Multimídia”. *Relatório de Exame de Qualificação do Programa de Doutorado*, Departamento de Informática, PUC-Rio, Novembro de 1993.

[STCN92] Soares, L.F.G.;Tucherman, L.; Casanova, M.A.; Nunes, P.R. "Fundamentos de Sistemas Multimídia". *Porto Alegre: Instituto de Informática da UFRGS*. 1992.

[StFu90] Stotts, D.; Furuta, R. “Temporal hyperprograming”. *J. Visual Languages and Computing*. Vol. 1. N. 3. Setembro 1990.

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

[TsGD91] Tsichritzis, D.; Gibbs, S.; Dami, L. “Active Media”. *Object Composition, D. Tsichritzis ed.* Universite de Geneve, Centre Universitaire d’Informatique, Geneve, June 1991.

[WaRo94] Wahl, T. and Rothermel, K. “Representing Time in Multimedia Systems”. *Proceedings of International Conference on Multimedia Computing and Systems*. Boston, MA. IEEE Computer Society Press, May 1994.

[WiLe92] Wiil, U.K.; Leggett, J.J. “Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems”. *Proceedings of European Conference on Hypertext, ECHT'92*. Milano. December 1992.

[Will96] Willrich, R. “Conception Formelle de Documents Hypermedias Portables”. *Thèse Présentée au Laboratoire d’Analyse et D’Architecture de Sustémes du CNRS — Rapport LAAS n° 96376*. Setembro 1996.

[WSSD96] Willrich, R.; Sénac, P.; Saqui-Sannes, P. and Diaz, M. “Hypermedia Documents Design Using the HTSPN Model”. *Third International Conference on MultiMedia Modeling — MMM’96*, Tolouse, France, Novembro 1996.

[Zell89] P. Zellweger “Scripted Documents: A Hypermedia Path Mecanism”. *Proceedings of Hypertext'89*, Pittsburg, Pennsylvania, pp. 1-14, November, 1989.

[Zell95] P. Zellweger “Introductory Slides for Session 4: Presentation Temporal Layout”. *Electronic Proceedings of the ACM Workshop on Effective Abstractions in Multimedia, in connection with ACM Multimedia’95*, San Francisco, EUA. Novembro 1995.