MEIRE JULIANA ANTONACCI

NCL: UMA LINGUAGEM DECLARATIVA PARA ESPECIFICAÇÃO DE DOCUMENTOS HIPERMÍDIA COM SINCRONIZAÇÃO TEMPORAL E ESPACIAL

DISSERTAÇÃO DE MESTRADO

Departamento de Informática

Rio de Janeiro, 19 de Abril de 2000.

Pontifícia Universidade Católica do Rio de Janeiro

MEIRE JULIANA ANTONACCI

NCL: UMA LINGUAGEM DECLARATIVA PARA ESPECIFICAÇÃO DE DOCUMENTOS HIPERMÍDIA COM SINCRONIZAÇÃO TEMPORAL E ESPACIAL

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

Orientador: Luiz Fernando Gomes Soares.

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 19 de Abril de 2000.

Este trabalho é dedicado

aos meus amados pais Edson Antonacci e Maria Paulina Antonacci.

AGRADECIMENTOS

- Ao professor Luiz Fernando Gomes Soares, meu querido orientador, pela competência profissional, pelo exemplo de mestre e amigo, pela constante disponibilidade e atenção, e pela confiança que em mim depositou.
- Aos meus pais, irmãos e cunhado, Edson, Paulina, Paula, Júnior e Eduardo, que, apesar da distância, se fizeram fortemente presentes durante todo o desenvolvimento deste trabalho, pela paciência, compreensão, constante incentivo e apoio, e, principalmente, por todo amor e carinho.
- À equipe do Laboratório TeleMídia, pela amizade e pelo companheirismo, que muito contribuiram para a realização deste trabalho de uma forma mais amena e prazerosa. Em especial, aos amigos Débora Muchaluat e Rogério Rodrigues, pela presença constante, apoio e orientação indispensáveis à efetivação deste trabalho.
- À família Brandão, pela amizade, carinho, apoio incondicional, e força na decisão de buscar meus sonhos.
- Aos meus amigos muito queridos, Adelailson, Adriana, Christina, Suzana e Tonho, que suavizaram as dificuldades desta minha jornada, com a presença, o apoio, a solidariedade e a amizade nos diversos momentos.
- À amiga Karine Versieux, pela sua forma carinhosa de acompanhar este meu trabalho, sempre com uma palavra de apoio e incetivo.
- Aos inesquecíveis mestres, Cláudia Ferris e Pietrobon, pelo incentivo e confiança.
- A todos os professores e funcionários do Departamento de Informática da PUC-Rio que colaboraram, de uma forma ou de outra, para a conclusão deste trabalho.
- À CAPES e Embratel pelo apoio financeiro fornecido durante o curso.
- A Deus, pelo dom da vida e pela oportunidade de realizar este trabalho.

RESUMO

O principal objetivo deste trabalho foi a criação de uma linguagem declarativa para especificação de documentos hipermídia. Inicialmente foi realizada uma análise das diversas linguagens existentes e foram levantadas algumas características fundamentais que toda linguagem hipermídia deve satisfazer. Essas características de fato advêm dos modelos hipermídia em que essas linguagens se baseiam. Modelos com maior poder de expressão provêem mais recursos para autoria, isto é, permitem a criação de documentos mais completos. Após a análise das linguagens e o reconhecimento das características que seus modelos suportam, foi realizado um estudo e refinamento do modelo conceitual hipermídia NCM visando provê-lo de todas as características desejáveis encontradas nas demais linguagens. Assim, a criação de uma linguagem declarativa baseada no NCM também irá satisfazer as mesmas características. A linguagem criada, denominada NCL, foi especificada utilizando-se o padrão XML, que define uma meta-linguagem de marcação recomendada pelo W3C. Através da NCL é possível especificar documentos hipermídia genéricos contendo as características fundamentais reconhecidas anteriormente, incluindo relacionamentos de sincronização temporal e espacial entre seus componentes, sendo esta a contribuição central do trabalho. Uma outra contribuição a ser salientada foi a proposta de extensão da linguagem SMIL, linguagem hipermídia padronizada pelo W3C, visando sanar algumas de suas limitações e oferecer as mesmas vantagens da NCL.

Palavras-chave: autoria hipermídia, linguagem declarativa, NCM, XML.

ABSTRACT

The main goal of this work is the creation of a declarative language for the specification of hypermedia documents. At first, an analysis of several existing languages is made. Some fundamental characteristics of hypermedia languages are then recognized. Indeed these characteristics derive from hypermedia models the languages are based on. The more expression power models have the more authoring resources they provide allowing the development of complex documents. After analyzing these languages and describing the in models characteristics an analysis and refinement of the NCM hypermedia conceptual model is made with the purpose of providing NCM with all desirable characteristics found in the languages. The creation of a declarative language based on NCM will then provide the same characteristics. The language developed, called NCL, was specified using the XML standard, which defines a markup language recommended by W3C. Through NCL it is possible to specify generic hypermedia documents containing the previously mentioned characteristics, including time and space synchronization relations among their components – this is the main contribution of the present work. Another contribution to be pointed out is the proposal for an extended SMIL, a hypermedia language standardized by W3C, aiming at repairing some SMIL shortcomings and offering the same advantages of NCL.

Key words: hypermidia authoring, declarative language, NCM, XML.

Conteúdo

Lista de Figuras

Lista de Acrônimos

CSS ..... Cascading Style Sheets - Level 1
CSS2 ..... Cascading Style Sheets - Level 2
DSSSL..... Document Style Semantics and Specification Language
DTD ..... Document Type Definition
HTML ..... HyperText Markup Language
HTML+TIME..... Timed Interactive Multimedia Extensions for HTML
HTTP ..... HyperText Transfer Protocol
ISO..... International Organization for Standardization
NCL ..... Nested Context Language
NCM ..... Nested Context Model
SGML ..... Structured Generalized Markup Language
URI ..... Uniform Resource Identifier
URL ..... Uniform Resource Locator
URN..... Uniform Resource Name
W3C..... World Wide Web Consortium
XML..... Extensible Markup Language
XSL..... Extensible Style Language
XHTML ..... Extensible HyperText Markup Language
SMIL..... Synchronized Multimedia Integration Language

1. Introdução

1.1. Motivação

A utilização de uma linguagem de marcação simples (HTML [HTML98]) e um protocolo de comunicação de baixa complexidade (HTTP [BFFGM97]) são os principais fatores responsáveis pela grande popularidade da Web [BCGP92]. Porém, alguns anos atrás, mesmo quando a Web era utilizada basicamente para oferecer acesso a texto e imagens, e os documentos utilizados possuíam apenas os relacionamentos convencionais de navegação, denominados hiper-elos[1], a simplicidade da linguagem HTML já era um aspecto questionado.

Em geral, várias características dificultam o uso da linguagem HTML pelas aplicações, como por exemplo: a) HTML não é extensível – uma aplicação não pode definir novos elementos e tê-los reconhecidos por outras aplicações; b) pouca, ou quase nenhuma, semântica pode ser extraída de um documento HTML; c) os elos existentes em um documento HTML estão embutidos em seu conteúdo; d) não é possível definir relacionamentos temporais – os únicos elos existentes são os hiper-elos.

[1] Hiper-elos são os elos de navegação disparados pela ação do usuário.

Atualmente, com a grande difusão da Web e o aumento na utilização de documentos hipermídia, novas necessidades surgiram, como por exemplo a possibilidade de especificar relacionamentos de sincronização temporal e espacial entre componentes de documentos. Para isso, é preciso haver um modelo de documentos hipermídia com o poder de expressão desejado. Mais que isso, é importante que os documentos definidos de acordo com esse modelo possam ser representados em um formato estruturado e padronizado, com o intuito de garantir o intercâmbio de informações e dessa forma obter a interoperabilidade entre aplicações.

Visando definir linguagens para especificar documentos hipermídia de forma declarativa, a fim de conseguir essa interoperabilidade entre aplicações, várias linhas foram seguidas. Algumas propostas partiram da própria linguagem HTML e definiram extensões para a mesma, objetivando solucionar algumas de suas limitações, mas mantendo sua simplicidade. Como principal exemplo de proposta nessa direção, pode ser destacado o HTML+TIME [TIMEH98], que sugere acrescentar novos atributos e elementos ao HTML permitindo definir um comportamento temporal nos seus documentos. Outra proposta também nessa direção está descrita em [RoDu98]. No entanto, essas propostas, apesar de incluírem a capacidade de definir relacionamentos de sincronização temporal em documentos HTML, herdam diversas limitações como, por exemplo, o fato dos elos estarem embutidos no conteúdo dos documentos, impossibilitando, entre outras coisas, o reuso do conteúdo dos documentos sem a herança obrigatória das relações.

Seguindo uma outra abordagem, grupos de trabalho do W3C tentaram propor novas linguagens, independentes do HTML, baseadas em outros modelos de documento hipermídia. Como ponto de partida para criação dessas novas linguagens foi utilizado o padrão SGML [SGML86], recomendado pela ISO em 1986 e utilizado para formalizar o HTML desde a versão 2.0. O SGML é uma meta-linguagem poderosa o suficiente para suportar a criação de linguagens para especificação de documentos hipermídia contendo relações temporais ou ligações hipertexto complexas. Várias vantagens podem ser associadas ao seu uso, entre elas: a) estrutura hierárquica – o SGML permite a representação da estrutura hierárquica dos elementos em um documento; b) flexibilidade – SGML não dita quais tipos de elementos devem ser criados ou como eles se relacionam; c)

especificação formal – a estrutura e o tipo dos elementos contidos em um documento SGML são formalmente definidos por uma gramática; d) reusabilidade – tanto as definições dos documentos como os documentos propriamente ditos podem ser utilizados por diversas aplicações.

Dessa forma, diversas propostas para especificação de documentos hipermídia foram desenvolvidas baseadas no padrão SGML. Entre elas, encontra-se a linguagem HyTime [HyTime97] definida como um padrão internacional pela ISO. Essa linguagem provê facilidades para representar informações estáticas e dinâmicas que são processadas e intercambiadas por aplicações multimídia, permite a especificação de interconexões complexas dentro e entre os documentos e o escalonamento de informações multimídia no tempo e no espaço.

Observa-se que as limitações do HTML são superadas com o uso de SGML para criação de novas linguagens baseadas em modelos mais poderosos. Porém, por ser bastante completa, a utilização da meta-linguagem SGML pode tornar o processamento de documentos muito complexo e custoso, impedindo o seu uso direto em aplicações rodando sobre o ambiente distribuído da Web. Por este motivo, seu uso ficou limitado a uma classe de usuários para o qual esse custo era viável, dada a qualidade e quantidade de documentos manipulados. Assim, entre os principais usuários do SGML hoje estão as grandes editoras, os departamentos de defesa e de processamento de impostos dos EUA, empresas de aviação, etc.

Sendo assim, verifica-se a necessidade de uma proposta mais robusta que o HTML, porém especificada em alguma meta-linguagem mais fácil de ser tratada que o SGML. Com esse objetivo, surgiu o padrão XML [XML98], adotado como recomendação pelo W3C para descrição e intercâmbio de dados estruturados na Web. O XML é uma versão simplificada do SGML. Segundo seus autores, o XML deve alcançar o poder e a generalidade oferecidos pelo SGML, mantendo a simplicidade do HTML. Seu principal objetivo é representar, genérica e estruturadamente, documentos que podem ser intercambiados e processados na Web, sanando a dificuldade de se obter interoperabilidade e intercâmbio de informações entre sistemas hipermídia heterogêneos, um dos requisitos fundamentais para permitir a portabilidade e reutilização de aplicações.

Atualmente, já existem algumas linguagens baseadas no padrão XML [SMIL98, JLRST98] que possibilitam a especificação de documentos hipermídia contendo, além dos relacionamentos convencionais de navegação citados anteriormente, algum tipo de relacionamento de sincronização. Dentre elas, a mais difundida é a linguagem SMIL [SMIL98], também adotada como recomendação pelo W3C. No entanto, tanto SMIL como as propostas de extensão do HTML mencionadas, por tentarem estender o HTML buscando não perder a característica de simplicidade, acabam resolvendo vários problemas, mas permanecendo com algumas limitações, como pode ser visto em detalhes em [RRMS99].

Desenvolvido no laboratório TeleMídia, o modelo conceitual NCM (*Nested Context Model* – Modelo de Contextos Aninhados) [Soar00] é um modelo hipermídia que permite a criação de documentos contendo relacionamentos de sincronização espacial e temporal bastante poderosos. Percebeu-se que a definição de uma linguagem declarativa baseada neste modelo, e especificada em XML, irá suprir a grande necessidade da existência de uma linguagem padronizada, e sem as limitações identificadas em linguagens como o SMIL, para descrição de documentos hipermídia, que é a principal motivação deste trabalho.

A existência de uma linguagem que permita estruturar todas as características do modelo NCM irá, em adição, possibilitar a autoria declarativa de documentos hipermídia compatíveis com o modelo e o intercâmbio desses documentos de forma padronizada entre clientes e servidores NCM ou na Web. Um outro benefício será a facilidade para conversão automática de documentos NCM para outros modelos.

1.2. Objetivos

Este trabalho tem como principal objetivo a criação de uma linguagem declarativa para autoria de documentos hipermídia, denominada NCL (*Nested Context Language*), baseada em uma DTD (*Document Type Definition*) XML. Como base para definição dessa linguagem, optou-se por utilizar o modelo conceitual hipermídia NCM. Assim, é objetivo que a linguagem represente integralmente, e de forma estruturada, objetos hipermídia do modelo NCM.

Um outro objetivo deste trabalho é a implementação de módulos responsáveis pela conversão entre objetos NCM em uma representação NCL, e objetos NCM na representação Java definida no sistema HyperProp. O sistema HyperProp [RMS98] é um sistema para tratamento de documentos hipermídia, baseado no NCM, que vem sendo desenvolvido também no laboratório TeleMídia. Esse sistema permite a autoria gráfica de documentos com suporte a controle de versões, assim como a apresentação desses documentos em uma máquina também implementada em Java. Dessa forma, é objetivo que esses módulos conversores estejam totalmente integrados à atual versão do sistema HyperProp.

Neste trabalho, pretende-se também propor extensões à linguagem SMIL, visando sanar as limitações descritas em [RRMS99] e alcançar as mesmas facilidades proporcionadas pela linguagem NCL. Após estas extensões, outros módulos conversores devem ser implementados para possibilitar a integração da linguagem SMIL com o sistema HyperProp.

Como resultado deste trabalho, pretende-se definir um novo formato para troca de documentos hipermídia com sincronismo espacial e temporal na Web.

1.3. Organização da Dissertação

Esta dissertação inicia apresentando, no segundo capítulo, algumas linguagens para especificação declarativa de documentos hipermídia. Através da análise dessas linguagens, são identificadas algumas características básicas e fundamentais que toda linguagem com esse propósito deve possuir, servindo de base para a criação da linguagem NCL, proposta por esta dissertação.

No terceiro capítulo, é abordado o foco principal deste trabalho. Inicialmente, é introduzido o padrão XML que define a meta-linguagem de marcação utilizada para especificar a linguagem NCL. O modelo conceitual hipermídia NCM e alguns refinamentos sofridos por este em decorrência da criação da linguagem NCL são descritos em seguida. Por fim, são apresentadas as características principais da linguagem NCL e a sua DTD.

O quarto capítulo descreve a implementação dos módulos responsáveis pela conversão entre objetos NCM em uma representação NCL e objetos NCM na representação Java definida no sistema HyperProp. Neste capítulo, serão apresentadas sua estruturação e a descrição dos testes realizados.

No quinto capítulo, são descritos os principais pontos de extensão sofridos pela linguagem SMIL que deram origem à linguagem X-SMIL (a DTD X-SMIL é apresentada no Apêndice A). Ainda no quinto capítulo, são apresentados os módulos conversores que foram implementados para possibilitar a integração da linguagem SMIL com o sistema HyperProp.

Finalizando, no sexto capítulo, é feita uma comparação crítica entre a linguagem NCL e as linguagens apresentadas no Capítulo 2. Além disso, as principais contribuições deste trabalho são ressaltadas, bem como alguns tópicos a serem abordados como trabalhos futuros.

2. Trabalhos Relacionados

Neste capítulo, são descritas algumas linguagens para especificação declarativa de documentos hipermídia que possuem algum tipo de interatividade e sincronização temporal e espacial. As linguagens, aqui apresentadas, foram escolhidas de forma a ilustrarem: exemplos de linguagens que partiram da linguagem HTML e propuseram extensões a ela (HTML+TIME); linguagens que usaram o SGML para sua especificação (HyTime); linguagens especificadas em XML e que refletem o mesmo modelo adotado pela linguagem HTML (XHTML); e, ainda, linguagens que são também especificadas em XML, mas que se baseiam em um modelo hipermídia próprio (Madeus e SMIL).

No decorrer do capítulo, podem ser observadas algumas características comuns e outras particulares a cada uma das linguagens descritas. Uma avaliação das vantagens e desvantagens dessas linguagens, bem como uma comparação final entre suas características e as características da linguagem NCL, descrita no próximo capítulo, será apresentada na Seção 5.1.

2.1. HTML+TIME

HTML+TIME (*Timed Interactive Multimedia Extensions for* HTML) [TIMEH98] é uma proposta, submetida recentemente ao W3C, para a integração de interatividade e

temporização à linguagem HTML através da adição de novos elementos e atributos. Essa integração se baseia principalmente nos conceitos definidos pela linguagem SMIL, apresentada na Seção 2.5.

Segundo seus idealizadores, o fato dos autores HTML não precisarem aprender uma sintaxe totalmente nova para especificar estrutura e temporização em seus documentos, constitui uma grande vantagem oferecida pela proposta.

Basicamente, HTML+TIME utiliza: a linguagem HTML [HTML98] e CSS2 [CSS298], para a formatação e exibição dos elementos de um documento; a linguagem XML [XML98], para a adição de dados estruturados; e a linguagem SMIL [SMIL98], para a definição de comportamentos temporais e interativos em um documento.

O comportamento temporal em um documento, pode ser definido através de atributos que determinam o início, o fim e a duração da exibição de elementos, o número de vezes que o elemento deve ser repetido em seqüência, bem como através de atributos que permitem agrupar elementos em *containers* que possuem algum outro comportamento temporal especificado. Os elementos que suportam a especificação desses atributos são todos os elementos HTML que são definidos dentro do elemento *body* e que representam conteúdo ou estilo.

O agrupamento de elementos é obtido através da especificação dos atributos *par* ou *seq* de um *container* HTML com o valor *true*. Dessa forma, se um elemento *container* especificar o atributo *par* com valor *true*, ele passa a se comportar como um *container* temporal paralelo, no qual todos os seus elementos filhos devem ser exibidos simultaneamente. Se, ao invés do atributo *par*, um *container* especificar o atributo *seq* com valor *true*, ele passará a se comportar como um *container* temporal seqüencial e todos os seus filhos serão exibidos em série.

HTML+TIME, através dos atributos *beginEvent* e *endEvent* permite que os elementos tenham sua exibição controlada por eventos (de usuário, por exemplo). Opcionalmente, é

permitido que, com esses atributos configurados com o valor *none*, a exibição de elementos seja controlada através de *scripts* que ordenam a execução de métodos[2].

Dois métodos da linguagem são disponíveis para controlar a exibição de elementos (*containers*, inclusive): *beginElement*(), para acionar o início da exibição de um elemento, e *endElement*(), para terminar a exibição de um elemento.

Em HTML+TIME, além de controlar o início da exibição de seus elementos, é permitido controlar o início da exibição de um documento[3]. Esse controle é feito através da especificação do atributo *timeStartRule* que pode assumir os seguintes valores: *immediate*, indicando que a exibição deve ser iniciada logo que o código do documento começar a ser carregado (esse é o valor default); *onDocLoad*, indicando que a exibição deve começar quando todo o documento tiver sido carregado, sem precisar esperar a carga de outros documentos e elementos associados ao documento em questão; *onDocComplete*, indicando que a exibição só deve começar quando o documento e todos os seus elementos e documentos associados estiverem completamente carregados; *onStartTag*, indicando que a exibição do documento deve se iniciar quando um elemento identificado pela marcação *startDocumentTimeline* for reconhecido e instanciado (essa nova marcação é utilizada para definir o quanto do documento deve estar carregado para que a exibição seja iniciada).

HTML+TIME suporta a definição de hiper-elos temporais, assim como a definição de elementos específicos para representar objetos de mídia temporizáveis. Com o uso de tais elementos, a integração de objetos de diversas mídias a uma apresentação fica facilitada.

O conteúdo de uma apresentação em HTML+TIME pode ser selecionado de acordo com características da plataforma de exibição. Para implementar tal comportamento, HTML+TIME incorpora os mesmos elementos e atributos de seleção de conteúdo definidos em SMIL 1.0 (Seção 2.5).

---

[2] Encontra-se em estágio inicial a definição de um modelo de objetos de documentos (DOM) da linguagem HTML+TIME. Em [TIMEH98] estão documentados alguns métodos, propriedades e eventos disponibilizados pela linguagem atualmente.

3 Normalmente, exibe-se um documento quando ele está completamente disponível, ou seja, quando todos os seus componentes foram carregados.

2.2. HyTime

A linguagem HyTime (*Hypermedia/Timed-Based Structuring Language*) [HyTime97], definida como um padrão internacional pela ISO, foi desenvolvida e especificada de acordo com a meta-linguagem SGML [SGML86].

Ela define alguns recursos e arquiteturas relacionados, mas totalmente independentes, que provêem, como já mencionado anteriormente, facilidades para representar informações estáticas e dinâmicas que são processadas e intercambiadas por aplicações multimídia; e que permitem a especificação de interconexões complexas dentro e entre os documentos e o escalonamento de informações multimídia no tempo e no espaço.

Desde o início, a proposta da linguagem era definir um padrão genérico e totalmente abstrato que possibilitasse a especificação de documentos hipermídia de diversos tipos. No entanto, o alto nível de abstração torna-se uma desvantagem, pois, muitas vezes, dificulta o entendimento da linguagem. Nem sempre é óbvio perceber como as suas abstrações poderiam ser aplicadas para solucionar um determinado problema.

Uma outra desvantagem da linguagem HyTime está relacionada à sua especificação. Documentos especificados em SGML são geralmente muito complexos, tornando o seu custo de processamento muito elevado. Desta forma, HyTime, apesar de possuir um modelo hipermídia bastante poderoso e completo e ser especificada em uma meta-linguagem formal e genérica, não é apropriada para o uso no ambiente distribuído da Web.

Maiores detalhes sobre as características específicas desse padrão fogem ao escopo desta dissertação e podem ser encontrados em [HyTime97].

2.3. XHTML

O XHTML é uma proposta de reformulação do HTML como uma aplicação XML. Sua especificação é baseada na versão 4.0 do HTML e define três DTDs que correspondem à DTD única do HTML.

Segundo os autores do XHTML, esta é a primeira linguagem de conteúdo que está de acordo com o padrão XML e que, ao mesmo tempo, se algumas regras de conversão forem seguidas, opera em conformidade com o HTML 4.0. Além disso, eles afirmam que os desenvolvedores que migrarem seu conteúdo para XHTML irão obter vários benefícios, como: documentos XHTML são compatíveis com XML, podendo ser lidos, visualizados, editados e validados em qualquer ferramenta XML; documentos XHTML podem ser servidos como sendo do tipo *text/html*, ou *text/xml*, ou *application/xml*.

Atualmente, os desenvolvedores de documentos estão constantemente descobrindo maneiras de expressar suas idéias através de novas marcações. Em XML, é relativamente fácil introduzir novos elementos ou atributos. Esse é um outro benefício oferecido pela proposta XHTML, uma vez que ela foi projetada visando suportar essas extensões através de novos módulos, que devem ser desenvolvidos baseando-se em algumas técnicas também descritas na proposta.

Em XHTML, os elementos e atributos possuem o mesmo significado dos elementos definidos no HTML, portanto sua semântica encontra-se definida em [HTML98]. Aqui, cabe-se apenas descrever as suas diferenças principais em relação ao HTML 4.0, ressaltando-se que essas diferenças estão relacionadas à forma como os elementos e atributos de um documento devem ser especificados.

Devido ao fato da linguagem XHTML ser uma aplicação XML, certas práticas que eram permitidas em HTML devem ser modificadas e as seguintes regras devem ser obedecidas:

- todos os elementos não vazios devem possuir *tags* de fim (*end-tags*);
- todos os elementos vazios devem ser especificados em uma *tag* de elemento vazio (*empty-element tag*);
- todos os elementos devem ser devidamente aninhados;
- todos os nomes de elementos e atributos devem ser especificados em letras minúsculas;
- valores de atributos devem estar sempre entre aspas.

Uma especificação mais detalhada da proposta e as suas DTDs podem ser encontradas em [XHTML99].

2.4. Madeus

Madeus [JLRST98, JRT98] é um sistema para autoria e apresentação de documentos multimídia interativos, desenvolvido no centro de pesquisa Rhône-Alpes do INRIA (*Institute National de Recherche en Informatique et en Automatique*), situado na França.

O sistema Madeus é uma ferramenta de autoria baseada em restrições, na qual o autor pode descrever a organização temporal e espacial dos documentos estabelecendo restrições entre os seus componentes, que podem ser simples ou compostos.

Madeus permite definir restrições espaciais clássicas, como: *align*, *center* e *shift*, tanto no eixo vertical quanto horizontal. As restrições temporais são baseadas na álgebra de Allen [Alle83] e utilizam operadores como: *equals, starts, before*, etc.

Para possibilitar, além da autoria gráfica, uma autoria declarativa de documentos baseados no modelo conceitual definido pelo sistema Madeus, foi especificada uma DTD XML para o sistema. Seguindo as regras dessa DTD, é possível especificar, em XML, relacionamentos temporais e espaciais entre os componentes do documento, através de operadores específicos.

Basicamente, em um documento Madeus, são definidos os seus componentes e os relacionamentos entre eles.

Os componentes do documento podem ser: componentes simples que representam objetos de mídia de algum tipo, ou componentes compostos, formados por outros componentes simples e/ou compostos. Madeus prevê a existência dos seguintes tipos de componentes simples: *Text, Picture, Video, Movie, Audio, Plugin, Unknown, External* e *Fictitious*. Cada um desses componentes pode especificar uma série de atributos definindo seu posicionamento na tela e seu tamanho no momento da exibição, assim como atributos que definem a localização do arquivo que contém o objeto especificado e sua duração. Além desses, todo componente simples deve especificar um atributo *Name* que define o seu nome/identificador. O componente *Text* permite, ainda, que seja definida a formatação da

fonte a ser utilizada, através de atributos específicos. E, para elementos do tipo *Applet* também podem ser definidos parâmetros a serem passados para o browser na sua execução.

Um componente composto é definido através do elemento *Composite*. Esse elemento pode possuir atributos que definem seu posicionamento na tela no momento da exibição, assim como atributos que determinam sua duração. O atributo *Name* também deve ser especificado sempre que um elemento *Composite* for definido. Esse atributo define o nome/identificador do elemento. O conteúdo de um elemento que representa um componente composto pode ser formado por todos os componentes simples definidos anteriormente e por outros componentes compostos.

Componentes compostos têm sua duração determinada em função da duração de todos os seus elementos. Essa duração é o menor intervalo possível que englobe a exibição de todos os seus componentes. De forma semelhante, a área de exibição de um componente composto é a menor área possível que englobe todas as áreas de exibição dos componentes nele contidos.

Quaisquer componentes, simples ou compostos, definidos em um documento Madeus, podem conter a definição de hiper-elos, através dos atributos *a* ou *anchor*, cuja sintaxe é semelhante à sintaxe adotada pelo HTML.

Os relacionamentos existentes entre os componentes do documento podem ser temporais ou espaciais. Esses relacionamentos são definidos através de restrições, facilitando a autoria de documentos hipermídia, uma vez que os autores não precisam ter conhecimento da real duração dos componentes, necessitando, apenas, especificar a duração aceitável de cada um deles e as restrições que devem ser respeitadas na apresentação do documento.

A definição de restrições entre componentes em um documento Madeus é feita através de elementos que representam os operadores temporais ou espaciais. Os elementos *Equals, Meets, Finishes, Starts, After, Before, Begins, Overlaps* e *During* são exemplos de elementos que representam operadores temporais. Enquanto, *Left_align, Right_align, Bottom_align, Top_align, Center_v* e *Center_h* são alguns exemplos de elementos definidos na DTD do Madeus que permitem a especificação de operadores espaciais.

2.5. SMIL

O padrão SMIL, recomendado pelo W3C em junho de 1998, especifica uma linguagem para autoria declarativa de documentos multimídia definida através de uma DTD XML. Seu principal objetivo é possibilitar a integração de um conjunto de dados de mídia independentes[4] em uma apresentação multimídia sincronizada [SMIL98].

Assim como o HTML, SMIL é uma linguagem de marcação. No entanto, por serem especificados em XML, documentos SMIL possuem um formato mais rígido. O conteúdo desses documentos é formado apenas por elementos SMIL, sendo que os dados de mídia propriamente ditos são armazenados separadamente e referenciados através de atributos dos elementos que os representam.

A estrutura lógica de um documento SMIL é definida por seus elementos e atributos. Basicamente, um documento SMIL contém um elemento mais externo denominado *smil* que, por sua vez, pode conter dois outros elementos principais: *head* e *body*.

O elemento *head* contém informações associadas ao layout da apresentação do documento. *Layout* é um elemento contido em *head* que define e descreve a área da tela onde o documento será exibido, ou seja, a janela de apresentação do documento. No elemento *layout* é possível especificar regiões da janela de apresentação onde os componentes podem ser exibidos e a ordem de sobreposição destas regiões, uma vez que as regiões podem se sobrepor, total ou parcialmente.

O elemento *body* contém informações sobre os objetos de mídia que compõem o documento, assim como a especificação dos instantes de início e fim da exibição desses objetos e a especificação de relações entre eles.

Os objetos de mídia básicos que podem compor um documento SMIL são: áudio, vídeo, texto e imagem, representados pelos elementos *audio*, *video*, *text* e *img*, respectivamente. Cada um desses elementos possuem vários atributos que especificam, por exemplo: a URI

[4] O termo independente significa que cada mídia deve ser armazenada separadamente.

do arquivo que contém os dados relativos à mídia em questão; a região de apresentação na qual o elemento deve ser exibido; a duração explícita do elemento; seu tempo de início e término explícitos; e o número de vezes que o elemento deve ser exibido seqüencialmente.

Dois outros elementos bastante importantes que fazem parte do conteúdo do elemento *body* são os elementos *par* e *seq*. Esses elementos permitem estruturar a apresentação do documento e determinam como seus componentes serão exibidos. Eles definem grupamentos que podem ser formados por elementos que representam objetos de mídia e/ou por elementos que representam grupamentos (*par* e *seq*). Os filhos do elemento *seq* são exibidos seqüencialmente, enquanto os filhos do elemento *par* são exibidos em paralelo, ou seja, simultaneamente.

A possibilidade de especificação do início, fim e duração da exibição de cada objeto, bem como a especificação de exibições paralelas e seqüenciais dos componentes do documento, consistem em características importantes da linguagem SMIL. A especificação SMIL também define um modelo temporal próprio, no qual são estabelecidas regras para cálculo dos eventos (início, duração e fim) de cada elemento de sincronização (objetos de mídia e grupamentos *par* e *seq*). Maiores detalhes sobre o modelo temporal SMIL podem ser encontrados em [SMIL98].

Os elos entre os componentes do documento SMIL são estabelecidos entre uma âncora de origem e uma âncora de destino. As âncoras podem identificar um elemento inteiro ou sub-regiões espaciais ou temporais de objetos de mídia. A navegação em um elo é sempre disparada pela ação do usuário, ou seja, na linguagem SMIL os elos são sempre hiper-elos, não sendo possível definir elos de sincronismo.

A linguagem SMIL também possibilita a definição de comportamentos alternativos para a apresentação do conteúdo de um documento. Esses comportamentos são baseados em fatores específicos da plataforma de exibição, como por exemplo: as preferências do usuário; os recursos disponíveis no computador onde o documento será exibido; o desempenho da rede. As alternativas são especificadas através do elemento *switch* e dos atributos de teste definidos em seus componentes. O *switch* pode conter qualquer elemento de sincronização, além do próprio *switch*. Os atributos de teste representam valores

booleanos que, quando avaliados como falso, fazem com que o elemento não seja exibido. O primeiro elemento, filho do elemento *switch*, que satisfizer os atributos de testes será exibido, ignorando-se todos os outros.

3. A Linguagem NCL

Após o estudo e análise de algumas linguagens declarativas para especificação de documentos hipermídia, percebeu-se que cada uma delas apresentava várias vantagens e desvantagens, porém, nenhuma oferecia todas as características desejáveis para especificação de um documento hipermídia com algum tipo de interatividade e sincronização entre seus componentes[5].

Esta foi a principal motivação deste trabalho, cujo objetivo é a criação de uma linguagem genérica e sem limitações ou, com o mínimo possível, para descrição de documentos hipermídia com sincronização espacial e temporal.

No entanto, para se criar essa nova linguagem, além do levantamento das características mínimas desejadas, dois fatores fundamentais devem ser observados:

- toda linguagem se baseia em algum modelo conceitual;
- as linguagens são definidas formalmente utilizando-se alguma meta-linguagem de marcação.

O modelo conceitual é responsável pelas características básicas da linguagem, enquanto a meta-linguagem é responsável pela sua simplicidade, legibilidade e facilidade de criação.

Assim, para se ter uma boa linguagem, é fundamental que o seu modelo base seja um modelo satisfatório e que a meta-linguagem utilizada para formalizá-la possibilite a criação de documentos legíveis com extrema facilidade.

Preocupando-se com esses fatores, para criar a nova linguagem proposta por este trabalho, optou-se por utilizar a meta-linguagem XML para formalizá-la e o modelo hipermídia NCM como sendo o seu modelo base. A nova linguagem foi denominada NCL – *Nested Context Language*.

Neste capítulo, inicialmente é introduzido o padrão XML que define a meta-linguagem de marcação utilizada para especificar a linguagem NCL. Em seguida, o modelo conceitual hipermídia NCM é apresentado, assim como alguns refinamentos no modelo decorridos da criação da linguagem NCL. Por fim, são descritas as características principais da linguagem NCL e suas etapas de criação.

3.1. O Padrão XML

Criado em 1996, o grupo de trabalho *XML Working Group*, originalmente conhecido como *SGML Editorial Review Board*, pretendia desenvolver uma versão simplificada do SGML [SGML86] que apresentasse o mesmo poder e generalidade, porém que fosse mais simples e apropriada para o uso na Web. Como resultado desse trabalho, foi publicado em fevereiro de 1998, como recomendação pelo W3C, o padrão XML, dando início a vários outros trabalhos e recomendações associados.

Basicamente, o XML especifica uma meta-linguagem de marcação que permite representar estruturadamente informações diversas que podem ser intercambiadas e processadas na Web de forma uniforme.

Segundo o *XML Working Group*, os principais objetivos do padrão são:

- ser fortemente utilizado na Internet;

---

[5] Uma avaliação das vantagens e desvantagens das linguagens descritas no capítulo anterior, bem como uma comparação entre essas e a linguagem NCL, proposta nesta dissertação, será apresentada na Seção 5.1.

- suportar uma grande variedade de aplicações;
- ser compatível com SGML;
- ser formal e conciso;
- permitir uma fácil elaboração de programas que processem documentos XML;
- conter um número mínimo de características opcionais, preferencialmente nulo;
- permitir a produção de documentos XML legíveis pelo homem e razoavelmente claros;
- possibilitar uma rápida elaboração de projetos em XML;
- possibilitar a criação de documentos XML com facilidade.

Assim como o SGML, o XML especifica somente o conteúdo e a estrutura de um documento. Dessa forma, para que sejam criadas múltiplas apresentações baseadas em um mesmo documento, basta associá-lo a folhas de estilo diferentes, conforme ilustrado na Figura 1.

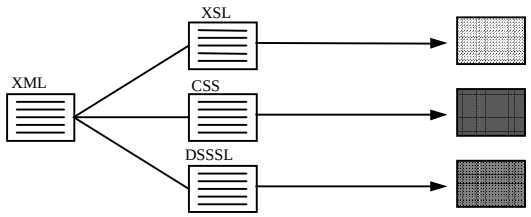


Figura 1 – Apresentações diferentes de um mesmo documento XML

Além de possibilitar o reuso das estruturas e a criação de apresentações diferentes para o mesmo documento, vários outros benefícios são apresentados pelo XML, como por exemplo, fornecer aos usuários uma visão estruturada dos dados, possibilitar a integração de dados estruturados de diversas fontes diferentes, descrever dados de uma grande variedade de aplicações e, permitir que os autores definam seus próprios conjuntos de *tags*.

A especificação XML [XML98] descreve uma classe de objetos de dados chamada documentos XML e descreve, parcialmente, o comportamento dos programas que irão processá-los.

Cada documento XML possui uma estrutura lógica e uma estrutura física. Fisicamente, o documento é composto de unidades de armazenamento, denominadas entidades, que podem ser internas ou externas. Entidades externas são arquivos que contêm marcações XML.

Utilizando essas entidades, é possível construir documentos XML em módulos. Por exemplo, suponha que se pretenda criar um documento XML representando um livro com vários capítulos. Para que esse documento não seja muito extenso, pode-se especificar cada capítulo em um arquivo físico separado. O documento XML que representa o livro deverá declarar cada um desses arquivos como uma entidade externa e referenciá-la na ordem desejada, conforme apresentado na Figura 2. Outra grande vantagem das entidades externas é o reuso, ou seja, uma mesma entidade externa pode ser utilizada em vários documentos diferentes.

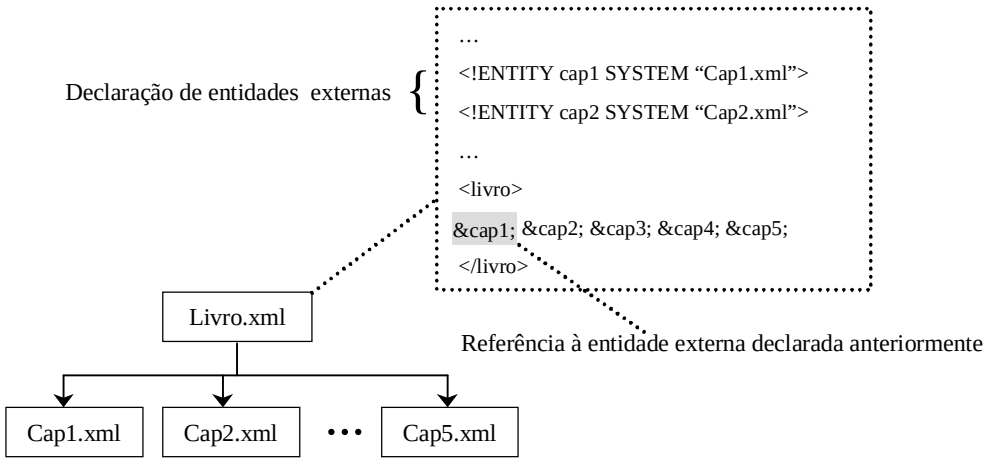


Figura 2 – Representação da estrutura física de um documento XML utilizando entidades externas.

O XML também suporta a idéia de entidades internas, que são um tipo de atalho para um pedaço de marcação XML, ou dados, contidos dentro do documento. Entidades internas podem ser usadas para frases muito comuns, expansões ou abreviações, conforme o exemplo da Figura 3, no qual a referência "*&xml;*" será substituída por "*Extensible Markup Language*".

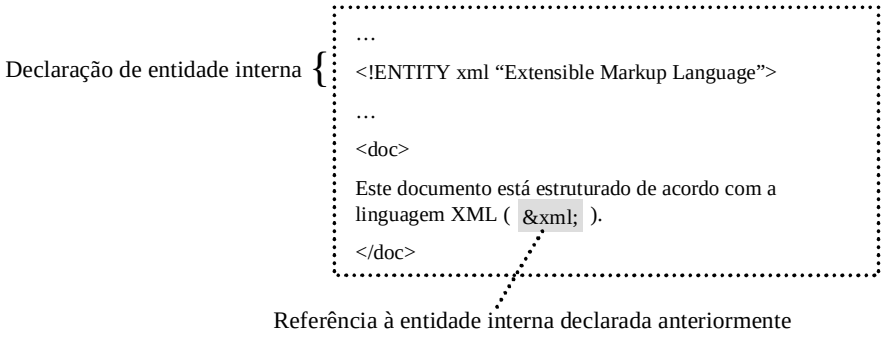


Figura 3 – Utilização de entidades internas em documentos XML.

A estrutura lógica de um documento XML é indicada pela marcação que ele contém. São seis os tipos de marcações existentes em XML. Portanto, logicamente, o documento é composto de declarações, elementos, comentários, seções CDATA, referências e instruções de processamento, conforme detalhado a seguir.

- **Elementos**

Os elementos são os principais construtores de XML, através dos quais a estrutura hierárquica de um documento é criada. Um documento XML contém um ou mais elementos. Cada elemento possui um tipo, identificado por um nome e, pode ou não, conter um conjunto de especificações de atributos.

Um elemento não vazio tem seu conteúdo delimitado por sua *start-tag*, por sua vez, delimitada por "<"e ">", e sua *end-tag*, delimitada por "</"e ">", como em "*<nome-do-elemento> conteúdo do elemento </nome-do-elemento>*". Um elemento vazio contém apenas atributos e precisa apenas da *empty-element tag* que começa com "<" e termina com "/>", como em "*<nome-elemento-vazio especificação-de-atibutos/>*". Todo o texto entre a *start-tag* e a *end-tag* de um elemento faz parte do seu conteúdo que pode ser composto de outros elementos e/ou caracteres. No exemplo da Figura 4, o elemento "*Depto*" é um elemento vazio, o elemento "*Professor*" contém o elemento "*Nome*" que, por sua vez, contém somente texto na forma de caracteres.

- **Atributos**

Os atributos qualificam um elemento. Cada atributo é um par (*nome="valor"*) presente na *start-tag* do elemento, logo após o seu nome.

Em XML, todos os valores de atributos devem estar entre aspas, podendo ser simples ou duplas. Além disso, na especificação do atributo, não é permitido nenhum espaço em branco antes nem depois do sinal "=".

Um elemento pode especificar mais de um atributo, porém, um determinado atributo não pode ser especificado mais de uma vez no mesmo elemento. Na Figura 4, o elemento

"*Curso*" especifica os atributos "*cod*" e "*nome*" com os valores "*mm-0100"* e *"Multimidia"*, respectivamente.

```
<?xml version="1.0"?>
<!DOCTYPE curso SYSTEM "curso.dtd">
<Curso cod="mm-0100" nome="Multimidia">
    <Depto nome="Informatica"/>
    <Professor>
        <Nome> Jose Diniz </Nome>
    </Professor>
    <Aluno>
        <Nome> Daniela </Nome>
        <Nome> Márcio </Nome>
        <Nome> Paulo </Nome>
    </Aluno>
</Curso>
```

Figura 4 – Exemplo de um documento XML.

- **Entidades**

Como mencionado anteriormente, um documento XML, fisicamente, consiste em uma ou mais unidades de armazenamento denominadas entidades. Todas as entidades possuem um conteúdo e são identificadas por um nome.

As entidades podem ser internas ou externas. Entidades internas possuem um valor que é atribuído diretamente na sua declaração. A declaração de entidade da Figura 3 mapeia o nome "*xml*" para o conteúdo "*Extensible Markup Language*". Neste caso, nenhuma unidade de armazenamento separada está envolvida.

A especificação XML predefine cinco entidades internas. São elas: "*lt*", "*gt*", "*amp*", "*apos*" e "*quot*". Quando referenciadas, essas entidades internas são substituídas, respectivamente, pelos seguintes caracteres[6]: <, >, &, ' e ".

Entidades externas referenciam, na sua declaração, unidades de armazenamento separadas (Figura 2), que podem conter texto ou dados binários (como por exemplo, arquivos

[6] Os caracteres <, > e & só devem aparecer em um documento, em sua forma literal, se fizerem parte da marcação. Nos outros casos, devem ser substituídos pelas entidades correspondentes.

bitmap). Quando uma entidade externa, referenciada em um documento XML, contém texto, o seu conteúdo também será passado para o parser que irá analisá-lo como parte integrante do documento. Porém, se a entidade contiver somente dados binários, seu conteúdo não será passado para o parser e, consequentemente, não será analisado.

Uma referência a uma entidade interna ou externa, em um documento XML, é da forma "*&nome-da-entidade;*", assim, a expressão "*x < y*" deve ser escrita como "*x < y*". Com exceção das entidades internas predefinidas, antes de referenciar qualquer entidade é preciso declará-la, sendo que o nome utilizado para referenciar a entidade deve ser exatamente igual ao nome utilizado na sua declaração.

O XML não permite recursão, ou seja, uma entidade não pode referenciar direta ou indiretamente a si própria.

- **Comentários**

Comentários podem ser usados para fazer alguma anotação nos documentos XML. Eles são delimitados por "<!--" e "-->", como em "<!--*Isto é um comentário*-->" e podem ser de qualquer tamanho, ocorrer em qualquer lugar do documento e conter qualquer conjunto de dados[7], exceto a seqüência "--".

Os comentários devem ser usados com um certo cuidado, pois, eles não fazem parte do conteúdo do documento, consequentemente, os processadores XML podem ou não torná-los disponíveis para as aplicações.

- **CDATA**

As seções CDATA são usadas para escrever textos que contêm caracteres reservados que poderiam ser reconhecidos como marcações, mas que, neste caso, não são. Elas podem ocorrer em qualquer lugar no documento onde é permitida a ocorrência de caracteres de dados. Seu formato é "*<![CDATA[…]]>*", como ilustrado em: "*<![CDATA[*p = &q; b = (I <= 3);]]>*".

Qualquer conjunto de dados pode estar contido nas seções CDATA, exceto a seqüência "*]]>*". Portanto, não é permitido o aninhamento de seções CDATA.

- **Instruções de Processamento**

As instruções de processamento são da forma "*<?nome-ip ...?>*". Elas não fazem parte do conteúdo do documento XML devendo ser passadas pelo processador para a aplicação. As aplicações devem processar, apenas, as instruções que reconhecerem, ignorando as demais. Todas as instruções que começam com "*xml*' são reservadas para a linguagem.

Documentos XML podem e devem começar com uma instrução de processamento denominada declaração XML, como em: "*<?xml version="1.0"?>*" (Figura 4). Nesta declaração, o número da versão é provido para futuras extensões da linguagem.

- **Prólogo**

Os documentos XML possuem um prólogo opcional seguido por um elemento obrigatório, conhecido como elemento raiz[8] e, depois, por uma miscelânea de marcações[9] que também são opcionais. O texto "*<saudacao> Alo mundo!!! </saudacao>*" representa um documento XML que não contém um prólogo, ele é formado somente pelo elemento raiz "*saudacao*".

O prólogo é formado por dois componentes principais: a declaração XML e a declaração de tipo de documento.

Conforme visto anteriormente, a declaração XML é um tipo especial de instrução de processamento que indica à aplicação que este é um documento XML e qual a versão utilizada. Ela pode ser usada pelo processador, por exemplo, para sinalizar um erro caso o

---

[7] Os caracteres <, > e & podem ser usados normalmente dentro dos comentários, pois, nestes elementos, nenhuma marcação é interpretada.
[8] Elemento raiz é o elemento que engloba todos os outros elementos, dados ou comentários do documento XML.

documento recebido tenha sido escrito em uma versão que ele não suporta. A declaração XML, se presente, é sempre a primeira marcação existente no documento.

O segundo componente do prólogo, a declaração de tipo de documento, contém, ou referencia, ou ambos, um conjunto de marcações declarativas que definem a gramática do documento, isto é, sua definição de tipo de documento (DTD). Esta declaração deve aparecer entre a declaração XML e o começo do elemento raiz.

```
<?xml version="1.0"?>
<!DOCTYPE saudacao SYSTEM "alo.dtd">
<saudacao> Alo mundo !!! </saudacao>
```

Figura 5 – Documento XML contendo uma declaração de tipo de documento que referencia um conjunto externo de regras.

No exemplo da Figura 5, as regras para o documento podem ser encontradas no arquivo referenciado pela URL "*alo.dtd*". Esta entidade é chamada de subconjunto DTD externo.

Conforme ilustrado no exemplo na Figura 6, a declaração de tipo de documento também pode incluir algumas ou todas as regras dentro dela mesmo. Neste caso, a única regra para este documento (a declaração de um elemento "*saudacao*" cujo conteúdo deve ser formado por caracteres) está dentro da declaração de tipo de documento e antes do começo do elemento raiz, denominando-se subconjunto DTD interno.

```
<?xml version="1.0"?>
<!DOCTYPE saudacao [
<!ELEMENT saudacao (#PCDATA)>
]>
<saudacao> Alo mundo !!! </saudacao>
```

Figura 6 – Documento XML contendo uma declaração de tipo de documento que referencia um conjunto interno de regras.

Geralmente, a declaração de tipo de documento irá conter ambos os subconjuntos DTD externo e interno. A idéia é que seja usado um subconjunto DTD externo para referenciar uma DTD padrão e, no subconjunto DTD interno, sejam declaradas as características

---

[9] Estas marcações podem ser comentários, instruções de processamento ou caracteres de tabulação e espaços em branco.

específicas para o documento em questão. Uma observação importante é que as regras especificadas no subconjunto DTD interno serão lidas primeiro e terão precedência sobre as declarações do subconjunto DTD externo.

- **Definição de Tipo de Documento - DTD**

Corresponde às marcações declarativas que definem a gramática do documento. A DTD (Figura 7) especifica a seqüência e aninhamento de elementos exigidos ou permitidos, os valores e tipos dos atributos exigidos ou permitidos e os nomes de entidades internas e externas necessárias.

```
<!ELEMENT Curso (Depto, Professor, Aluno)>
<!ATTLIST Curso cod ID #REQUIRED>
<!ATTLIST Curso nome CDATA #IMPLIED>
<!ELEMENT Depto EMPTY>
<!ATTLIST Depto nome CDATA #IMPLIED>
<!ELEMENT Professor (Nome+)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT Aluno (Nome*)>
```

Figura 7 – Exemplo de definição de tipo de documento utilizada para validar o exemplo da Figura 4.

Declarações de elementos identificam os nomes dos elementos e a natureza de seu conteúdo. São da seguinte forma: "*<!ELEMENT nome-elemento conteúdo>*". O conteúdo do elemento pode ser especificado pelas palavras "*EMPTY*", "*ANY*", ou "*#PCDATA*" que indicam, respectivamente, que o elemento é vazio, que o elemento pode conter qualquer conteúdo e que o conteúdo do elemento é formado por uma seqüência de caracteres.

Caso o conteúdo do elemento seja composto de outros elementos, deve-se especificar, entre parênteses, uma lista dos elementos permitidos seguidos ou não por algum símbolo especial. Estes símbolos podem ser: o sinal de adição ("+") indicando que o elemento deve ocorrer uma ou mais vezes, o asterisco ("*") indicando que o elemento deve ocorrer zero ou mais vezes e a interrogação ("?") indicando que o elemento é opcional. Esta lista deve ser separada por vírgulas (",") indicando que os elementos devem ocorrer na ordem especificada, e/ou por barras verticais ("|") indicando que, apenas um dos elementos listados, deve ocorrer. A vírgula tem o significado de 'e' e a ordem dos elementos na lista deve ser obedecida. Por exemplo, na Figura 7 o elemento "*Curso*" deve conter como filhos um elemento "*Depto*", seguido por um elemento "*Professor*" e depois por um elemento

"*Aluno*", nesta ordem. A barra vertical significa 'ou'. Por exemplo, se na mesma Figura 7, em vez de vírgula, fosse usada uma barra vertical para separar os elementos, o conteúdo do elemento "*Curso*" seria composto por um elemento "*Depto*", ou por um elemento "*Professor"*, ou por um elemento "*Aluno*". Nesse mesmo exemplo, o elemento "*Curso*" também poderia ter sido declarado como "*<ELEMENT Curso ((Depto)*, (Professor)?, ((Aluno)+ | (Turma)))*". Neste caso, o conteúdo do elemento deveria ser formado por zero ou mais elementos do tipo "*Depto*" e, zero ou um elemento do tipo "*Professor*" e, um ou mais elementos do tipo "*Aluno*" ou um elemento do tipo "*Turma*", seguindo essa ordem.

Após a especificação do elemento, pode-se especificar seus atributos. Uma especificação de atributo é da seguinte forma: "*<!ATTLIST nome-elemento atributo>*".

Cada atributo é composto por três partes: nome, tipo e valor padrão. O tipo pode ser "*CDATA*", que define seqüências de caracteres, "*ID*", que indica um identificador único para o documento, "*IDREF*" ou "*IDREFS*", que fazem referência, respectivamente, a um valor, ou a uma lista de valores de atributos do tipo "*ID*" declarados no mesmo documento, "*ENTITY*", que faz referência a uma entidade, "*NMTOKEN*" ou "*NMTOKENS*" que é uma forma restrita do "*CDATA*", representam um nome ou uma lista de nomes. Um atributo também pode ser do tipo enumeração. Neste caso, o campo tipo na especificação do atributo deve ser formado por seus possíveis valores, entre parênteses e separados, um a um, por uma barra vertical.

O valor padrão de um atributo pode ser: "*#REQUIRED*", indicando que o atributo deve obrigatoriamente ser especificado, "*#IMPLIED*", indicando que o atributo é opcional, "valor", indicando o valor a ser atribuído ao atributo quando outro não for especificado, "*#FIXED 'valor' "*, indicando o valor obrigatório do atributo sempre que ele for especificado.

No exemplo da Figura 7, o elemento "*Curso*" possui dois atributos. O atributo "*cod*" é do tipo "*ID*" e possui valor padrão "*#REQUIRED*", dessa forma, esse atributo deve ser especificado obrigatoriamente todas as vezes que for declarado um elemento "*Curso*" e seu valor deve ser único no documento inteiro. O segundo atributo do elemento "*Curso*" é o atributo "*nome*". Esse atributo é do tipo "*CDATA*" e possui valor padrão "*#IMPLIED*",

indicando que a sua especificação é opcional e que seu valor, quando especificado, deve ser formado por uma seqüência de caracteres.

De acordo com a especificação XML, um documento pode ser classificado como válido (Figura 5 e Figura 6) ou bem definido (Figura 8). Documentos válidos possuem, além da declaração XML, a declaração de tipo de documento que, conforme visto anteriormente, contém ou referencia as regras estruturais que o documento deve seguir (DTD). Esse conjunto de regras, ou gramática, informa ao software como processar o documento XML, permitindo que o documento seja validado. Assim, pode-se dizer que um documento XML é válido se ele está associado a uma DTD e satisfaz todas as suas regras.

Para ser bem definido, o documento não precisa estar associado a nenhuma gramática ou conjunto de regras, basta simplesmente que ele seja estruturado e satisfaça as seguintes restrições:

- o documento deve começar com a declaração XML (“<?xml version=“1.0”?>”);
- todos os elementos devem estar contidos dentro de um elemento raiz;
- todos os elementos devem estar devidamente aninhados;
- todos os elementos não vazios devem ter *start-tag* e *end-tag*;
- cada uma das entidades analisadas que são referenciadas direta ou indiretamente dentro do documento devem ser entidades bem definidas, ou seja, devem satisfazer estas mesmas regras.

```
<?xml version=“1.0”?>
<saudacao> Alo mundo !!! </saudacao>
```

Figura 8 – Documento XML bem definido

Uma aplicação que utilize documentos XML deve processar os documentos e verificar se seu conteúdo está de acordo com as regras de formação de um documento XML em geral (documento bem definido) e, se for o caso, validar sua estrutura e conteúdo frente à gramática correspondente, definida na DTD (documento válido).

Como algumas vezes não se deseja validar o documento, mas apenas ler seu conteúdo, na declaração XML, apresentada anteriormente, também pode ser especificado o atributo “*rmd*” (*required markup declaration*). Esse atributo serve de guia para o processador XML

e pode assumir três valores: "*internal*", indicando que deve-se validar apenas as declarações internas do documento, "*all*", indicando que devem-se validar tanto as declarações internas quanto as declarações externas ao documento, e "*none*", indicando que nenhuma das declarações do documento deve ser validada.

3.2. O Modelo de Contextos Aninhados (NCM)

O NCM é um modelo conceitual hipermídia baseado no conceito de nós, representando os componentes de um documento hipermídia, e elos, representando os relacionamentos entres esses nós.

Os nós do NCM podem ser de dois tipos básicos: nós de conteúdo (ou nós terminais) e nós de contexto. Esses últimos permitem definir a estrutura do documento e são o ponto central do modelo.

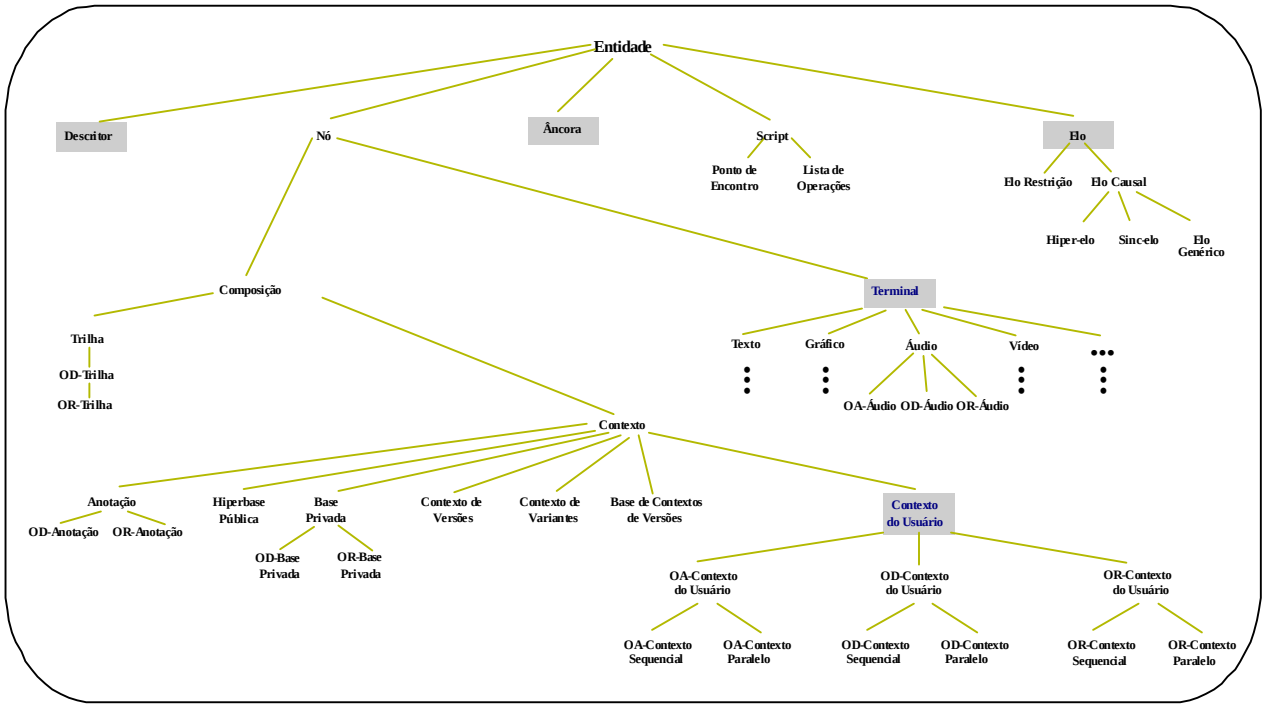


Figura 9 – Hierarquia de classes do NCM

A estrutura lógica do documento é definida utilizando o conceito de contexto (*nó de contexto*) como um grupo de componentes de documento (*nós*) e opcionalmente seus relacionamentos (*elos*).

Na Figura 9 está ilustrada a hierarquia de classes do NCM. Apesar de cada uma dessas classes apresentar a sua importância no modelo, nesta seção serão definidas, além das classes básicas, apenas as classes destacadas com fundo cinza, por serem as únicas necessárias na definição da linguagem NCL. Uma especificação completa do modelo e de todas as suas classes pode ser encontrada em [Soar00].

No modelo NCM, uma *entidade* é um objeto que possui um identificador único, hora e data de criação, autor e uma lista de controle de acesso. Para cada atributo da entidade, a lista de controle de acesso associa um usuário, ou grupo de usuários, aos seus direitos de acesso ao atributo.

Um *nó* é uma entidade que tem como atributos adicionais:

- um *tipo do conteúdo* – indicando o tipo do conteúdo do nó, esse tipo está de acordo com a especificação MIME;
- uma *especificação do conteúdo* – contendo uma descrição textual do conteúdo do nó;
- um *conteúdo* – conjunto de unidades de informação específicas para cada tipo de nó;
- uma *lista ordenada de âncoras* – cada elemento dessa lista é uma *âncora*[10] que define um conjunto de unidades de informação marcadas no conteúdo do nó; e
- um *conjunto de descritores alternativos* – contendo um conjunto de descritores que podem ser selecionados para definir a apresentação do nó. Um *descritor* contém informações que determinam como o nó deve ser apresentado, no tempo e no espaço, similar ao modelo Dexter [HaSc90]. O objeto descritor será apresentado posteriormente.

[10] Existe uma âncora *default*, denominada âncora lambda (λ), que representa o conteúdo inteiro do nó.

Um *nó de conteúdo* ou *nó terminal* contém informações e âncoras dependentes da aplicação. Eles podem ser especializados em outras classes, como por exemplo texto, vídeo, áudio, etc.

Um *nó de contexto de usuário C* possui, como conteúdo, uma lista *L* de nós que podem ser de conteúdo e de contexto do usuário, recursivamente. Nós de contexto de usuário diferentes podem conter um mesmo nó e 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 contexto de usuário 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 contexto de usuário, N_i está contido em N_{i+1}, para $i \in [1,m)$ e N_m não está contido em qualquer nó. Note-se que podem haver várias perspectivas diferentes para um mesmo nó *N*, se este nó estiver contido em mais de um contexto de usuário.

Os nós de contexto de usuário *C* também possuem dois atributos adicionais, denominados coleção de apresentação e relações.

Uma coleção de apresentação contém, para cada nó contido em *C*, um grupo de conjuntos de descritores alternativos (definidos posteriormente) do qual apenas um descritor pode ser selecionado em cada conjunto (um descritor é escolhido dependendo da melhor QoS possível para uma dada plataforma). Assim, tem-se, para cada nó de *C*, um grupo de descritores selecionados ou o valor nulo. O grupo de descritores selecionados deve necessariamente formar um conjunto, ou seja, não pode haver repetição de descritores no grupo[11].

O conteúdo do atributo relações é formado por um conjunto R de elos, tais que todo nó cabeça de cada elo em R ou é o próprio contexto de usuário *C*, ou um nó filho de *C* (a

[11] A semântica por trás da definição do grupo de descritores selecionados para cada nó N contido em um nó de contexto de usuário é permitir uma navegação em profundidade para N especificando várias exibições diferentes, da mesma forma como será permitido pela navegação por elos com diferentes pontos terminais de destino para um mesmo nó.

definição de elo e nó cabeça de um elo será dada a seguir). Um elo representa uma relação entre os nós a que se refere.

Nós de contexto de usuário vão servir, entre outras coisas, para definir uma estrutura hierárquica, para documentos, permitindo a definição de diferentes visões de um mesmo documento e melhorando a orientação do usuário na navegação em um documento. Quando especializados em nós de contexto seqüencial ou paralelo, nós de contexto de usuário vão ser úteis na definição da sincronização temporal de seus componentes.

A unidade básica para sincronização no NCM é o evento. Seguindo a definição encontrada em [PeLi96], um *evento* é uma ocorrência no tempo que pode ser instantânea ou durar um período de tempo. Para o NCM, um *evento* é a apresentação, *evento de apresentação*, ou a seleção, *evento de seleção*, de um conjunto não vazio de unidades de informação em uma dada perspectiva, seguindo as diretrizes de um dado descritor ou, ainda, a mudança de um atributo de um nó ou de um descritor (detalhado a seguir), chamado *evento de atribuição*.

O conceito de perspectiva, apresentado anteriormente, é de suma importância para a definição de eventos. Diz-se que um evento é definido não para um nó, mas para um nó em uma dada perspectiva.

Um evento pode estar em um dos seguintes estados: *dormindo*, *preparando*, *preparado*, *ocorrendo*, *suspenso* e *abortado*. A máquina de estados genérica dos eventos NCM é apresentada na Figura 10.

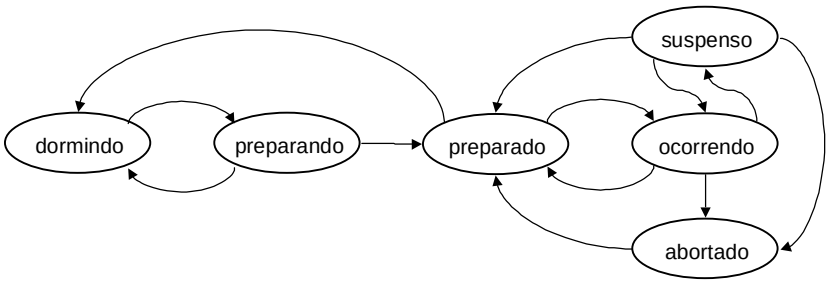


Figura 10 – Máquina de estados de eventos

Todo evento possui um atributo denominado *ocorrências — Oc*, que conta o número de vezes que o evento muda do estado *ocorrendo* para o estado *preparado* durante a apresentação de um documento. Eventos de exibição também possuem um atributo

denominado *repetições — Rpt*, que determina o número de vezes seguidas que o mesmo deve ser exibido. Esse atributo pode conter um valor finito ou o valor *indefinido*, que levará a uma execução em loop do evento, até que a mesma seja interrompida. Além disso, eventos de exibição possuem outros quatro atributos: custo de elasticidade, duração mínima, duração máxima e duração esperada. *Custo de elasticidade* descreve uma função que informa as durações mínima, ótima e máxima para apresentação do evento. Mais que isso, a função indica a relação "duração *versus* custo", descrevendo o preço que se paga por assumir uma duração diferente da ótima, ou seja, o custo por encolher ou esticar o tempo de exibição de um objeto. A duração ótima de apresentação é dada pelo ponto mínimo da função, podendo, inclusive, existirem múltiplos valores. As *durações mínima, esperada* e *máxima* serão calculadas pela máquina responsável pelo controle de apresentação do documento, denominado formatador temporal e espacial [Rodr97]. A duração esperada, idealmente, assumirá um valor ótimo, dado pela função de custo de elasticidade, mas poderá assumir novos valores, em função de relacionamentos especificados no documento que tenham que ser satisfeitos e de parâmetros que definam a plataforma de exibição. As durações mínima e máxima servirão para definir o intervalo de tempo válido para duração da exibição, levando em consideração não apenas o custo de elasticidade do evento de forma isolada, mas também todos os relacionamentos especificados no documento em que esse evento participe, direta ou indiretamente.

O *elo* é a entidade do modelo utilizada para representar relacionamentos entre os nós recursivamente contidos em um contexto. Um elo NCM é um relacionamento *n:m* composto por um conjunto de pontos terminais de origem, um conjunto de pontos terminais de destino e um ponto de encontro. O conjunto de pontos terminais define eventos e o ponto de encontro do elo, relações entre eventos.

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: $<(N_k,..., N_2, N_1), \alpha, tipo, \mathcal{D}>$, onde:

- N_{i+1} é um nó de contexto e N_i está contido em N_{i+1}, para todo $i \in (0,k)$, com k>0.
- α é uma âncora ou a identificação de um atributo de N_1.

- *tipo* especifica o evento (apresentação, seleção ou atribuição) associado a α. No caso de evento de atribuição, o valor de α deve identificar o atributo ou uma âncora de N_1. No caso de evento de apresentação ou seleção, α sempre especifica uma âncora de N_1.
- $\mathcal{D}$ é um conjunto de objetos descritores alternativos.

O nó N_k é chamado *nó cabeça* do elo, ele é o nó que contém o elo. N_1 é chamado de *nó âncora* ou *nó base* do elo.

A classe elo é especializada nas classes *elo causal* e *elo restrição*.

Conforme mencionado, o atributo ponto de encontro do elo define o relacionamento entre seus eventos (pontos terminais de origem e destino). O valor do atributo *ponto de encontro* de um *elo* causal é uma 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 (por exemplo, a apresentação de uma âncora) dos elos causais. Condições dizem respeito aos pontos terminais de origem (por exemplo, a seleção de uma âncora pelo usuário).

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 em que a condição é avaliada, e uma condição corrente, a ser satisfeita no instante de tempo em que 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: $=,\neq,<,\leq,>$ e $\geq$. As comparações podem ser realizadas em relação ao estado de um evento, ou em relação à variável *ocorrências (Oc)* ou à variável *repetições (Rpt)* associada a um evento, ou em relação aos atributos de um nó, no caso de um evento de atribuição. Qualquer expressão de condições baseada nos operadores lógicos $\wedge$ (e), $\vee$ (ou) e $\neg$ (negação) define uma condição composta. Além disso, a qualquer condição, simples ou composta, pode ser aplicado o operador modal (denominado

retardo). O operador retardo aplicado a uma condição *C* é definido da forma *C* [t_1, t_2], onde t_1, $t_2 \in \Re$ e $0 \leq t_1 \leq t_2$. Dado que uma condição *C* é verdade num instante *t*, uma condição *C'*, definida como *C* [t_1, t_2], é verdade no intervalo de tempo [$t+t_1$, $t+t_2$].

Similar às condições, as ações[12] de um ponto de encontro do elo causal podem ser simples ou compostas. Uma ação simples é executada sobre um evento pertencente ao conjunto de pontos terminais de destino. Uma ação composta é formada por uma expressão de ações baseada nos operadores | (paralelo) e $\rightarrow$ (seqüencial), definindo a ordem de execução de cada elemento da ação. Além disso, toda ação, simples ou composta, possui um atributo opcional denominado *tempo de espera*. Esse atributo define um tempo que deve ser aguardado antes que a ação seja executada. Ele é especificado através de valores mínimo, ótimo, máximo e custo de elasticidade.

A classe elo causal 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.

No caso de um hiper-elo, uma ação composta também pode ser formada baseada no operador binário $\oplus$. A $\oplus$ B implica que uma e, apenas uma, das ações A e B será executada. A escolha pode, por exemplo, ser passada ao leitor, logo após a seleção. Este operador pode dar uma flexibilidade grande na navegação de um documento. Por exemplo, através de seu uso podemos determinar se um nó N_1 deve iniciar sua exibição em paralelo ao fechamento da exibição do nó N_2, a partir de onde foi feita a seleção do hiper-elo, ou se a exibição de N_2 continua junto com a exibição de N_1 .

Relações de causa e efeito (condições/ações) não são suficientes para determinar todas as relações existentes entre componentes do documento. Existem relações que, semanticamente, especificam restrições (*constraints*) entre os pontos terminais do elo, tais como: dois nós devem terminar sua exibição ao mesmo tempo, *se e somente se* os dois nós estiverem sendo exibidos.

Um *elo restrição* é uma entidade cujo conjunto de pontos terminais de destino é nulo. O conjunto de pontos terminais de origem define eventos e o ponto de encontro da restrição vai definir restrições entre esses eventos. Um ponto de encontro de um elo restrição contém uma única operação denominada *simultânea*, composta por um conjunto de condições binárias simples. A semântica da operação é que todas as condições prévias que forem *VERDADE* em um dado instante de tempo, devem ter nas condições correntes correspondentes o mesmo valor (*VERDADE* ou *FALSO*), neste mesmo instante de tempo. Condições binárias simples dizem respeito aos pontos terminais de origem e têm as mesmas definições que nos elos causais.

No NCM, a especificação de layouts e de mudanças de comportamento durante a apresentação dos nós são definidas no objeto descritor. O descritor é especificado de forma separada do nó, permitindo um melhor reuso dos objetos. A apresentação de um nó do documento para um usuário final (leitor do documento) é feita a partir da união dos dados do nó propriamente dito a um descritor.

Um *descritor* é uma entidade que tem como atributos adicionais uma especificação de método para apresentação/edição do nó, uma especificação de método para finalizar a apresentação do nó e uma coleção de descrições de eventos. Assim como o *método de exibição/edição do nó*, o *método para finalizar uma apresentação* pode identificar qualquer programa ou pode ser um método implementado pelo próprio descritor. Cada *descrição de evento* consiste em uma tupla da forma <α, *tipo*, *dur*, *rep*, *oper*, *hab*>. O parâmetro α é um identificador de uma âncora pertencente à coleção de âncoras do nó ao qual o descritor será associado. *Tipo* especifica o tipo de evento associado à âncora (apresentação, seleção) ou tem o valor especial ξ. *Dur* é uma função de custo para iniciar o atributo custo de elasticidade do evento, sendo válido apenas para eventos de apresentação. No caso de um evento de seleção a duração recebe o valor <nulo>, indicando que a duração é instantânea. *Rep* também só é válido para eventos de apresentação e especifica o número de repetições do evento. Esse parâmetro inicia o atributo *repetições* do evento, caso o mesmo não seja iniciado por uma ação de um elo. *Oper* especifica um objeto lista de operações. *Hab* inicia o

---

[12] Uma descrição mais detalhada sobre as ações definidas pelo modelo pode ser encontrada em [Soar00].

atributo *habilita* do evento, que indica se a lista de operações pode ser executada ou não. Se o parâmetro *tipo* especificar um evento de apresentação ou seleção e o atributo *habilita* permitir, a lista de operações é avaliada sempre que houver uma mudança no estado do evento. Caso o parâmetro *tipo* receba o valor ξ, e o atributo *habilita* permitir, a lista de operações deve ser avaliada se o nó associado ao descritor receber uma mensagem de um elo especificando a âncora, resultado da ação *ativa* de um ponto de encontro.

3.3. Refinamentos do Modelo NCM

O modelo de contextos aninhados, descrito na seção anterior, surgiu aproximadamente em 1991 [Casa91]. Desde então, inúmeras pesquisas relacionadas vêm sendo realizadas, ocasionando em refinamentos e consolidações de características básicas do mesmo. Nesta seção, são descritos alguns desses refinamentos decorrentes da criação da linguagem declarativa NCL.

Durante a fase de especificação dessa linguagem, nos deparamos com vários pontos do modelo que ainda estavam em aberto, ou que precisavam sofrer algum tipo de refinamento, ou até mesmo que precisavam ser repensados. Entre eles, encontra-se a definição do descritor.

O conceito de descritor surgiu a partir da necessidade de especificação de mudanças de comportamento durante a apresentação de um documento, assim como da necessidade de se permitirem várias apresentações diferentes para um mesmo componente do documento (nó).

De acordo com o modelo, um descritor contém informações determinando como deve ser feita a apresentação do nó para o usuário, baseada no tempo e no espaço. Ele especifica, por exemplo, como o nó deve ser iniciado e qual dispositivo de E/S deve ser utilizado para sua apresentação.

Como visto na seção anterior, um descritor define uma especificação de iniciação, uma especificação de término e uma coleção de descrições de eventos. A especificação de iniciação contém as informações necessárias para iniciar a apresentação de um nó, assim

como um conjunto de operações que devem ser executadas para preparar esta apresentação. A especificação de término contém as informações necessárias para finalizar a apresentação do nó, ou seja, contém basicamente operações que devem ser executadas ao final da apresentação. As descrições de eventos especificam atributos relacionados à apresentação do nó, tais como, sua duração e número de repetições.

A especificação de iniciação e de término de um nó dependem do seu tipo. Por exemplo, para inicializar um áudio, um parâmetro importante é o volume. Enquanto para um texto, o volume não interessa, mas o tamanho da letra poderia ser importante.

Considerando-se um caso mais simples, a especificação de iniciação e de término poderia identificar um controlador específico para apresentação do nó, passando a este parâmetros para seu correto funcionamento. Uma especificação mais detalhada dos parâmetros do descritor, bem como a adição de novos parâmetros, constituem um dos pontos do modelo que sofreram alterações a partir da criação da linguagem NCL.

Entre os parâmetros de um descritor, podem ser salientados:

- **duração**

Este parâmetro corresponde ao parâmetro *dur* especificado na tupla que forma a descrição de evento do descritor, definido na seção anterior. Ele especifica a função de custo de elasticidade relativa à apresentação de um evento[13] do nó associado ao descritor. Esta função informa as durações mínima, ótima e máxima para apresentação do evento e descreve o preço que se paga por assumir uma duração diferente da ótima, ou seja, o custo por encolher ou esticar o seu tempo de exibição.

- **repetições**

Repetições é um parâmetro correspondente ao parâmetro *rep* especificado na tupla que forma a descrição de evento do descritor, definido na seção anterior. Seu valor especifica o

[13] O evento em questão é especificado no elo que possui este nó e descritor definidos no seu ponto terminal de origem ou destino.

número de vezes que um evento[14] do nó associado ao descritor deverá ter sua exibição repetida, em seqüência. Assim como o parâmetro duração, repetições é um parâmetro útil somente quando o evento associado ao nó for o evento de apresentação.

- **início**

Especifica uma função de custo relativa ao início explícito de um evento do nó associado ao descritor. Esta função informa o tempo mínimo, ótimo e máximo para o início do evento e descreve o preço que se paga por assumir um tempo de início diferente do ótimo, ou seja, o custo por antecipar ou retardar o seu início.

- **fim**

Especifica uma função de custo relativa ao término explícito de um evento do nó associado ao descritor. Esta função informa o tempo mínimo, ótimo e máximo para o término do evento e descreve o preço que se paga por assumir um tempo de término diferente do ótimo, ou seja, o custo por antecipar ou retardar o seu fim.

- **janela**

Este é um parâmetro que especifica o tamanho da área de visualização, ou seja, a janela onde é realizada a apresentação do documento. Se o descritor não especificá-lo, a janela de apresentação se torna dependente da implementação, devendo ser definida pela aplicação.

Os valores especificados por esse parâmetro correspondem às seguintes informações:

→ **altura**

Define a altura da janela de apresentação.

→ **largura**

Define a largura da janela de apresentação.

→ **dispositivo**

Considerando-se que um único documento pode ser apresentado em mais de um dispositivo de saída, por exemplo, o vídeo sendo apresentado em uma TV e o texto

[14] O evento em questão é especificado no elo que possui este nó e descritor definidos no seu ponto terminal

em um monitor de computador, podem-se agora definir janelas de apresentação em dispositivos diferentes. Por isso, além das informações de altura e largura da janela, deve-se também especificar um dispositivo de apresentação ao qual esta janela se refere. Quando a aplicação suportar somente um dispositivo de saída, esta informação não precisa ser especificada. Os valores correspondentes aos dispositivos deverão ser padronizados pelos sistemas de aplicação.

→ **cor-de-fundo**

Define a cor de fundo da janela de apresentação. Se esse valor não for especificado, o fundo da janela será transparente.

- **região**

Várias informações fazem parte do conteúdo desse parâmetro que identifica, basicamente, o espaço de apresentação do nó dentro da janela onde o documento será exibido. São elas:

→ **posição-x**

Corresponde à coordenada x do ponto superior esquerdo da região que está sendo definida. Seu valor é definido em relação à janela de apresentação especificada pelo descritor (parâmetro janela) ou pela aplicação.

→ **posição-y**

Corresponde à coordenada y do ponto superior esquerdo da região que está sendo definida. Seu valor é definido em relação à janela de apresentação especificada pelo descritor (parâmetro janela) ou pela aplicação.

→ **altura**

Define a altura da região de apresentação.

→ **largura**

---

de origem ou destino.

Define a largura da região de apresentação.

→ **cor-de-fundo**

Define a cor de fundo da região. Se esse valor não for especificado, o fundo da região será transparente.

→ **ajuste**

Este parâmetro especifica o comportamento no caso da altura e largura intrínsecas de um nó visual serem diferentes dos valores da altura e largura da região na qual o nó será exibido. Ele pode assumir os seguintes valores:

- *preencher*

  Dimensiona a altura e largura do nó independentemente, de forma que o seu conteúdo preencha todas as extremidades da região.

- *esconder*

  Se a altura ou largura intrínseca do nó for inferior à altura ou largura definida pela região, o nó é exibido a partir da extremidade superior esquerda e a altura ou largura restante é preenchida com a cor de fundo.

  Se a altura ou largura intrínseca do nó for superior à altura ou largura definida pela região, o nó é exibido a partir da extremidade superior esquerda até se alcançar a altura ou largura total da região, sendo omitidas as partes do nó que ficarem abaixo ou à direita.

- *adequar*

  Dimensiona o nó mantendo proporcionais as respectivas dimensões, até que a altura ou largura seja igual ao valor da altura ou largura especificado pela região, sem omitir qualquer parte do conteúdo do nó. O canto superior esquerdo do nó é posicionado nas coordenadas superiores esquerdas da região, enquanto que o espaço vazio à direita ou abaixo é preenchido com a cor de fundo.

- *adequar-e-cortar*

Dimensiona o nó mantendo proporcionais as respectivas dimensões, de forma que a altura e largura sejam iguais aos valores da altura e largura, especificados pela região, embora parte do conteúdo do nó possa ser omitida. Neste caso, é apresentada uma fatia horizontal ou vertical do nó. A largura excedente é omitida a partir do lado direito do nó e a altura omitida a partir da base.

- *rolar*

  Deve ser criado na região um mecanismo de deslocamento sempre que o conteúdo do nó exibido exceder os respectivos limites.

Se esse parâmetro não for especificado, o comportamento default da aplicação deve ser o comportamento definido pelo valor *esconder*.

→ **prioridade**

Define a prioridade da região, caso partes de duas ou mais regiões se sobreponham na janela de apresentação.

Uma outra parte importante do modelo NCM que sofreu alterações foi a parte relacionada às entidades virtuais.

Percebeu-se, no modelo, a necessidade da existência de entidades que possibilitassem a definição de um tipo de "contexto de consulta alternativo", no qual apenas um dos seus elementos constituintes seria selecionado, em tempo de execução, para ser apresentado. Essa seleção poderia ser dependente de algum parâmetro especificado pelos componentes do documento e pelo sistema, permitindo, por exemplo, que em uma determinada plataforma, fossem apresentados somente os componentes do documento cujo idioma fosse compatível com o idioma especificado pelo sistema na plataforma. Essa funcionalidade pode ser alcançada com o uso de entidades virtuais definidas a seguir.

Segundo o modelo, uma entidade virtual X é uma entidade que tem como valor de, pelo menos um de seus atributos, uma expressão escrita em uma linguagem de consulta formalmente definida. O atributo, cujo valor é formado por uma expressão, é denominado virtual. Quando alguma operação requer o seu valor, o valor resultante da avaliação da

expressão é retornado. Como exemplo, pode-se criar um nó virtual cujo conteúdo e regiões de suas âncoras são definidos por expressões.

No caso particular das âncoras, uma âncora virtual, em um conjunto de âncoras de um nó, é uma âncora cuja região é definida por uma expressão computada quando um elo que a tem em seu ponto terminal é percorrido. A expressão da âncora deve, nesse caso, resultar em um conjunto de unidades de informação dentro do nó no qual ela está contida, incluindo o símbolo especial λ, representando todo o conteúdo do nó.

Um elo virtual é um elo que tem, pelo menos um dos seus pontos terminais, definidos por uma expressão que será computada quando o elo for selecionado. Se esse elo estiver contido em um nó de contexto, então a sua expressão deve retornar um nó cabeça que seja o próprio contexto ou um nó contido nele.

Entre as consultas que definem entidades virtuais no NCM, uma diz respeito às âncoras virtuais em nós de contextos. Para um nó de contexto, uma âncora pode especificar uma expressão que determina um conjunto de nós contidos nele.

Uma outra consulta diz respeito ao elo virtual cujo ponto terminal é dado pela expressão: $<(N_k,..., N_2, (Select\ from\ N, exp)), \alpha, tipo, \mathcal{D}>$, tal que:

1. N_{i+1} é um nó de contexto e N_i está contido em N_{i+1}, para todo $i \in (1,k)$, com k>1
2. *N* é um contexto que limita o espaço de consulta aos nós que contém.
3. *tipo* e $\mathcal{D}$ têm as definições usuais do elo.
4. *exp* é uma expressão, ou a identificação de uma âncora de *N*, que conterá a expressão de consulta, ou ainda o valor ϕ, que identifica a expressão de consulta default. Em qualquer caso, a expressão determina um nó N_1 contido em *N*, ou um nó de contexto de usuário N_1 que contém um conjunto de nós contidos em *N*; neste último caso, *exp* é válida apenas se *tipo* for de apresentação ou seleção.
5. α é uma âncora (identificada pela posição na lista ordenada de âncoras) ou a identificação de um atributo do nó N_1, no caso da consulta resultar em um único nó. Caso contrário, o valor de α deve ser substituído pelo valor λ, representando uma âncora para todo o conteúdo de N_1.

Diz-se que N_2 contém, não N, mas sim o nó N_1 especificado pela consulta, isto é, o ponto terminal resolvido é dado pela tupla $<(N_k,..., N_2, N_1), \alpha, tipo, \mathcal{D}>$.

De forma simétrica para a relação de inclusão, é definido um nó de contexto de usuário virtual *C*, cujo conteúdo, do conjunto de nós, contém um nó dado pela expressão $<(Select\ from\ N, exp), g\{\mathcal{D}\}>$, tal que:

1. *N* é um nó de contexto que limita o espaço de consulta aos nós que contém.
2. *exp* é uma expressão, ou a identificação de uma âncora de *N*, que conterá a expressão de consulta, ou ainda o valor ϕ, que identifica a expressão de consulta default. Em qualquer caso, a expressão determina um nó N_1 contido em *N*, ou um nó de contexto de usuário N_1 que contém um conjunto de nós contidos em *N*.
3. $g\{\mathcal{D}\}$ é um grupo de conjuntos de descritores, ou o valor nulo. Como sempre, o *conjunto de descritores* contém um conjunto de descritores alternativos do qual apenas um pode ser selecionado em cada conjunto (um descritor é escolhido dependendo da melhor QoS possível para uma dada plataforma). Assim, tem-se, para o nó de N_1, um grupo de descritores selecionados, ou o valor nulo. O grupo de descritores selecionados deve necessariamente formar um conjunto, ou seja, não pode haver repetição de descritores no grupo. No caso de N_1 ser um nó de contexto, o grupo de descritores selecionados contém no máximo um objeto descritor.

Diz-se que *C* contém N_1.

Note que para um elo virtual $<(N_k,..., N_2, (Select\ from\ N, exp_1)), \alpha, tipo, \mathcal{D}>$ se referir a um nó N_1 resolvido pela consulta, N_1 deve estar contido diretamente em N_2, ou indiretamente, através de uma consulta $<(Select\ from\ N, exp_2), g\{\mathcal{D}\}>$. Esta é uma restrição do modelo. Mais ainda, se a consulta do contexto de usuário virtual ainda não estiver resolvida quando da navegação pelo elo, ela deve ser primeiramente resolvida, para então ser feita a navegação.

A especificação de alguns atributos para teste da plataforma, que podem ser avaliados pela expressão definida no elo virtual ou, no conteúdo de um nó de contexto de usuário virtual, foi outro ponto discutido e adicionado ao modelo recentemente. Estes atributos são

necessários para possibilitar a adaptação do documento à plataforma com o intuito de alcançar a melhor QoS. Um exemplo da utilização destes atributos seria a seleção de um nó para apresentação de acordo com a largura de banda disponível no sistema.

A definição dos valores desses atributos pode ser feita nos nós ou no descritor[15] associado ao nó. No caso destes atributos serem definidos em mais de um lugar, prevalece o valor especificado no descritor.

Basicamente, esses atributos podem ser utilizados para testar as capacidades e definições do sistema. São eles:

- **system-bitrate**

Este atributo especifica a largura de banda mínima, em bits por segundo, necessária para a exibição do nó. Em uma expressão, ele é avaliado como “true” se a taxa de bits disponível no sistema for igual ou superior ao valor especificado. Caso contrário, ele é avaliado como “false”. Seu valor pode receber qualquer inteiro superior a 0. Porém, se esse valor exceder qualquer valor máximo de largura de banda definido pela implementação, esse atributo é sempre avaliado como “false”.

- **system-language**

O valor do atributo é uma lista, separada por vírgulas, de nomes de idiomas, conforme definidos em [RFC1766]. Sua avaliação é verdadeira em dois casos: se pelo menos um dos idiomas especificados no sistema estiver presente na lista; se pelo menos um do idiomas especificados no sistema for igual ao prefixo de um dos idiomas presentes na lista (o prefixo é sucedido pelo “-”). Nos casos restantes é avaliado como “false”.

- **system-screen-size**

Os valores do atributo possuem a seguinte sintaxe:

```
valor-tamanho-da-tela ::= altura-da-tela"X"largura-da-tela
```

Cada um destes valores é expresso em pixels, devendo ser valores inteiros positivos. Esse atributo é avaliado como “true” em uma expressão se o sistema de exibição do documento

for capaz de exibir um nó com o tamanho indicado. Caso contrário, é avaliado como "false".

- **system-screen-depth**

Este atributo especifica, em número de bits, a precisão da paleta de cores da tela necessária para apresentar o nó. Seu valor deve ser positivo. Os valores mais característicos são 1, 8, 24, etc. É avaliado como "true", se o sistema de exibição do documento for capaz de apresentar o nó com a precisão de cor indicada. É avaliado como "false", se o sistema só for capaz de apresentar nós com precisões de cor inferiores.

- **system-captions**

Este atributo permite ao autor disponibilizar legendas relativas a um objeto de áudio, para pessoas com problemas de audição ou aprendendo uma nova língua, por exemplo. Assume o valor "on" (ativo) se o autor disponibilizar a apresentação das legendas, e "off" caso contrário. É avaliado como "true" se o valor for igual ao especificado no sistema.

- **system-overdub-or-caption**

Este atributo permite ao autor distinguir entre uma versão dublada ou legendada da apresentação do nó. O atributo pode tomar os valores "caption" (legendas) ou "overdub" (dublagem). É avaliado como "true" se o valor do atributo for igual ao especificado no sistema. Caso contrário, é avaliado como "false".

3.4. A DTD NCL

Como já mencionado anteriormente, a NCL é uma linguagem baseada no modelo conceitual NCM e especificada em XML. Suas características principais são as mesmas características do modelo NCM (Seção 3.2) e sua gramática está especificada de acordo com as normas do padrão XML (Seção 3.1).

---

[15] Apesar de não terem sido mencionados anteriormente, estes atributos também são parâmetros que foram recentemente acrescentados ao descritor.

Nesta seção, inicialmente é feita uma breve descrição dos passos envolvidos no desenvolvimento da linguagem. Logo após, são apresentadas algumas considerações importantes que devem ser observadas em um projeto XML e, em seguida, são salientadas algumas das características mais importantes da linguagem criada. A DTD NCL se encontra listada no final da seção e em [AnSo00a] pode ser encontrada uma especificação formal e bem detalhada de todos os elementos e atributos dessa DTD.

3.4.1. Etapas do Desenvolvimento da NCL

O desenvolvimento da NCL foi feito em quatro etapas. Na primeira etapa, foi feito um estudo das linguagens similares que estavam formalizadas em SGML e XML. Nessa etapa, foram analisadas as diversas características de cada uma das linguagens estudadas (Capítulo 2) e, ainda, foi feita uma lista de requisitos fundamentais que toda linguagem para descrição de documentos hipermídia deve satisfazer. Essa etapa foi seguida por um profundo estudo e refinamento do modelo NCM, modelo escolhido para ser a base da nova linguagem, diante da lista de requisitos levantada anteriormente. Após o estudo e refinamento do modelo, foi desenvolvido o primeiro projeto da linguagem NCL e implementados módulos para importar e exportar documentos NCL para outro formato; no caso, de acordo com o sistema HyperProp [RMS98]. Seguinte à criação dessas ferramentas para importar e exportar documentos, e em uma última etapa do desenvolvimento, efetuaram-se algumas modificações na linguagem, gerando finalmente a sua última versão.

3.4.2. Considerações Importantes em um Projeto XML

Na maioria das listas de discussões sobre XML, um dos principais tópicos abordados refere-se ao projeto em XML. Uma das questões relacionadas a este tópico que constituiu um dos pontos de muita alteração durante o desenvolvimento da linguagem NCL foi a questão da escolha entre representar uma informação como atributo de um elemento já definido ou como um novo elemento.

Segundo o *XML Working Group*, a escolha entre usar atributos ou elementos é principalmente uma questão de estética, porém, existem algumas regras práticas que devem ser observadas quando está se projetando uma linguagem de marcação.

Dentre elas, destacam-se:

- use um elemento embutido quando a informação que se quer codificar faz parte do elemento pai;

```
<autor>
    <primeiro-nome> Jorge </primeiro-nome>
    <ultimo-nome> Amado </ultimo-nome>
</autor>
```

- use um atributo quando a informação é inerente ao pai, mas não é uma parte constituinte;

```
<pessoa altura="1.70">
    <cabeca/>
    <corpo/>
</pessoa>
```

- use atributos para validação de tipos de dados simples;
- use elementos para validação de estruturas complexas;
- use elementos quando a informação possui uma estrutura interna própria;
- use elementos quando a informação pode estar contida em mais de um elemento.

Além dessas regras, o *XML Working Group* também cita algumas vantagens e desvantagens relacionadas ao uso de atributos.

Vantagens do uso de atributos:

- podem ter valores default;
- podem ser de um tipo de dados;
- ocupam menos espaço uma vez que eles não precisam de *start-tags* e *end-tags*.

Desvantagens do uso de atributos:

- atributos não são convenientes para grandes valores;

- espaços em branco não podem ser ignorados nos atributos;
- valores de atributos nem sempre aparecem na tela em ferramentas de edição;
- atributos não possuem ordem.

Agora, baseando-se nas regras práticas definidas e, considerando as vantagens e desvantagens em se usarem atributos, os projetistas criarão suas linguagens definindo quando as informações devem ou não ser mapeadas em novos elementos ou atributos. No entanto, é importante salientar que não existe uma norma rígida quanto ao uso de atributos ou elementos. De fato, algumas vezes, devem-se usar atributos para representar uma determinada informação e, em outro contexto, devem-se usar elementos para representar essa mesma informação.

```
<retangulo x="0" y="0" largura="10" altura="30"/>
```

```
<retangulo>
    <origem><x>0</x><y>0</y></origem>
    <tamanho><dx>10</dx><dy>30</dy></tamanho>
    <nome>Meu retangulo!</nome>
    <imagem>floral.jpg</imagem>
    <cor-de-fundo>cinza</cor-de-fundo>
</retangulo>
```

Figura 11 – Exemplo de uso de atributos e elementos.

A Figura 11 ilustra como a codificação de um objeto pode mudar unicamente porque o objeto está sendo descrito em um contexto diferente. No primeiro caso, foi modelado um objeto retângulo de uma maneira bem minimalista. Esse elemento poderia ser útil, por exemplo, quando se deseja simplesmente criar uma borda retangular ao redor de algum texto. Já no segundo caso, o objeto retângulo mais detalhado poderia ser usado para definir uma área de apresentação, sendo que, para essa área podem ser especificados, além da posição e tamanho, uma cor de fundo, uma imagem e um nome.

3.4.3. Principais Elementos e Características da NCL

Um documento NCL é sempre formado por um elemento raiz, denominado *ncl*, que contém dois outros elementos, o *head* e o *doc-body*. O *head* representa o cabeçalho do documento,

contendo informações relativas à exibição dos componentes, como por exemplo, o layout espacial da apresentação. O elemento *doc-body* representa o corpo do documento, contendo a definição dos componentes e seus relacionamentos. A Figura 12 ilustra a estrutura básica de um documento especificado em NCL.

```
<ncl>
  <head>
     <!--   ...definição do layout espacial
            ...definição dos descritores -->
  </head>
  <doc-body>
     <!--   ...especificação dos objetos de mídia
            ...especificação das composições
            ...especificação dos relacionamentos -->
  </doc-body>
</ncl>
```

Figura 12 – Estrutura de um documento NCL

O layout espacial da apresentação (elemento *layout*) contém a definição de janelas e regiões onde os componentes do documento serão exibidos. Ao contrário de outras linguagens, para cada dispositivo de exibição, pode ser definida uma janela, representada pelo elemento *root-layout*. Também podem ser definidas regiões (elemento *region*), associadas a essas janelas, que definem a área de exibição de algum componente. A linguagem permite que essas regiões sejam definidas independente das janelas, possibilitando o reuso de regiões em janelas diferentes.

Cada região, além dos atributos de identificação (*id*) e título (*title*), pode especificar os atributos *width*, *height*, *left* e *top*. Esses atributos identificam a largura e altura da região e a coordenada (x,y) onde seu vértice superior esquerdo será posicionado na janela de exibição associada, identificada por um outro atributo, chamado *root-layout*. Os valores dos atributos que especificam a área espacial podem ser valores absolutos, definidos em número de pixels, ou valores percentuais relativos ao tamanho da janela de exibição.

A Figura 13 ilustra a definição de dois elementos *root-layout*. O primeiro deles define uma janela de apresentação com largura igual a 640 e altura igual a 400 pixels (atributos *width* e *height*). O atributo *device* associa uma janela a um dispositivo de exibição, no caso o monitor do computador. O nome desse dispositivo deve ser reconhecido pelo formatador do documento. O segundo elemento *root-layout* define uma janela para um outro dispositivo, identificado por “tv”.

Na mesma figura, são definidas três regiões espaciais. A região “norte”, por exemplo, define uma área, posicionada na coordenada (0,0), com largura igual a 100% e altura igual a 20% relativas à janela de exibição, que corresponde à janela definida pelo *root-layout* “monitor”.

```
<ncl id=“/.telemidia:apresentacao”>
  <head>
    <layout>
      <root-layout id=“monitor” width=“640” height=“400” device=“monitor”/>
      <root-layout id=“tv” width=“860” height=“600” device=“tv”/>
      <region id=“norte” left=“0” top=“0” width=“100%” height=“20%” root-layout=“monitor”/>
      <region id=“leste” left=“20%” top=“20%” width=“80%” height=“60%”/>
      <region id=“oeste” left=“0” top=“20%” width=“20%” height=“60%”/>
    </layout>
    <descriptors-set>
      <descriptor id=“d1” region=“oeste” root-layout=“monitor”/>
      <descriptor id=“d2” region=“norte”/>
      <descriptor id=“d3” region=“leste” root-layout=“monitor”/>
      <descriptor id=“d4” repetitions=“5” dur=“9 10 11 3 3” root-layout=“monitor”/>
      <descriptor id=“d5” repetitions=“10” dur=“10” root-layout=“tv”/>
      <descriptor id=“d6” region=“norte” root-layout=“tv”/>
      <descriptor id=“d7” left=“0” top=“20%” width=“100%” height=“80%” root-layout=“tv”/>
    </descriptors-set>
  </head>
  <doc-body>
    <!-- ... -->
  </doc-body>
</ncl>
```

Figura 13 – Exemplo de definição do layout espacial da apresentação e das características de exibição dos componentes de um documento NCL

Nas linguagens hipermídia, usualmente, parte das informações relativas à apresentação de um componente são definidas no mesmo elemento que representa esse componente. No entanto, na NCL essas informações são inteiramente definidas em um elemento separado, denominado descritor (*descriptor*). Cada componente do documento pode estar associado a um ou mais descritores, o que permite especificar apresentações diferentes para um mesmo componente.

Dentre os atributos dos descritores, podem ser salientados:

- *dur, begin, end* – diferente das outras linguagens nas quais os valores desses atributos geralmente especificam tempos fixos, NCL, oferece flexibilidade para especificação desses tempos. Esses atributos são especificados através de funções de custo de elasticidade relativas à duração, início e término da apresentação do componente. Por exemplo, suponha um vídeo de duração igual a 20s, porém com uma tolerância de mais ou menos 10%, implicando que o mesmo pudesse ser acelerado ou atrasado em até 2s sem que isso comprometesse sua inteligibilidade. Supondo ainda que a perda de qualidade da apresentação fosse proporcional ao desvio da duração ideal, poderia ser

definida uma função de custo linear para especificação da duração do vídeo, determinando um valor mínimo (18s), ideal (20s) e máximo (22s) para exibição, e dois coeficientes (c_1 e c_2) que indicassem, respectivamente, a proporção de perda de qualidade (aumento do custo) caso o vídeo fosse encurtado ou prolongado. Dessa forma, o custo de exibir o vídeo com a duração ideal seria 0, com a duração mínima seria $2c_1$ e com a duração máxima seria $2c_2$.

- *repetitions* – especifica o número de vezes que o componente deverá ter sua exibição repetida, em seqüência.
- *root-layout* – referencia um elemento *root-layout*, já definido no elemento *layout*.
- *region* – especifica uma área para exibição do componente ou faz referência a uma região já definida no elemento *layout*.

Voltando ao exemplo da Figura 13, dentro do elemento *descriptors-set*, foram definidos sete descritores que serão associados aos componentes do documento para definir características das suas apresentações. Alguns deles ("d1", "d3" e "d6") definem, através dos atributos *root-layout* e *region*, uma janela e uma área para exibição do componente dentro dessa janela. Os valores desses atributos correspondem aos valores dos atributos *id* dos elementos *root-layout* e *region* associados. Um outro descritor ("d2") define apenas uma região, pois nela já foi especificada uma janela a ser associada. Os descritores "d4" e "d5" definem uma janela de apresentação, o número de vezes que o componente deve ser apresentado (*repetitions*) e a função de custo de elasticidade relativa à duração do componente (*dur*). Para o descritor "d4", foi especificada uma função de custo linear com os valores mínimo (9s), ideal (10s) e máximo (11s) para a duração da exibição, e os coeficientes que indicam o aumento do custo quando a duração afasta-se do valor ideal. Para o descritor "d5", foi especificada uma função de custo com o valor ótimo em 10s. O descritor "d7" também referencia uma janela de apresentação e especifica uma área para exibição do componente dentro dessa janela. Porém, nesse descritor, essa área foi definida através dos atributos *left*, *top*, *width* e *height*, que possuem a mesma semântica dos atributos homônimos especificados no elemento *region*. Se no elemento descritor, o autor referenciar uma região e, além disso, definir os atributos *left*, *top*, *width*, *height* ou *root-layout*, os valores dos atributos homônimos do elemento *region* serão sobrepostos pelos

valores definidos no descritor. Esse é o caso do atributo *root-layout* definido no descritor "d6" do exemplo ilustrado na Figura 13.

Em NCL, os componentes de um documento podem ser simples (objetos de mídia) ou compostos (composições).

Existem elementos específicos que possibilitam a definição de objetos de mídia de tipos diferentes, tais como texto (*text*), imagem (*img*), vídeo (*video*) e áudio (*audio*). Esses elementos não contêm o conteúdo dos dados propriamente dito, e sim uma referência (URI) para esse conteúdo. Essa referência é especificada através do atributo *src* dos elementos, conforme ilustrado na Figura 14, onde foram especificados três elementos do tipo texto, uma imagem e um áudio. Em cada um desses elementos, foram especificados, além do atributo de identificação (*id*), que será definido posteriormente, o atributo *uid*, contendo a URI do componente, o atributo *src*, contendo a URI do conteúdo da mídia e o atributo *descriptor-list*. Esse último possui a especificação dos descritores que poderão ser associados ao elemento para definir sua apresentação. Seu valor pode ser formado por uma lista de valores correspondentes aos atributos *id* de descritores definidos no cabeçalho do documento. Essa lista possibilita que um mesmo elemento possa ser apresentado de maneiras diferentes, de acordo com parâmetros de qualidade de serviço. No exemplo da Figura 14, o elemento *audio* pode ser associado aos descritores "d4" ou "d5" definidos na Figura 13.

A linguagem permite ainda definir âncoras temporais e espaciais específicas para cada tipo de objeto de mídia. Por exemplo, uma âncora de um texto (*text-anchor*) determina uma seqüência de caracteres (atributo *text*) que se encontra em uma posição especificada pelo número de caracteres a partir do início do texto (atributo *position*). Na Figura 14, o texto "introducao_menu" possui duas âncoras. Também pode ser delimitada uma âncora em uma mídia do tipo áudio, a partir de duas amostras (início e fim) especificadas em um arquivo de áudio. Em uma mídia do tipo imagem, pode ser criada uma âncora espacial (*img-anchor*) delimitada por um retângulo, especificado a partir de seu vértice superior esquerdo, altura e largura. Na Figura 14, a imagem "introducao_imagem" possui duas âncoras. As âncoras são bastante úteis na especificação de relacionamentos entre os componentes, descritos a seguir.

```
<ncl id="/.telemidia:apresentacao">
  <head>
     <!-- ... -->
  </head>
  <doc-body>
     <!-- ... -->
      <text id="titulo_monitor" uid="./telemidia:introducao.titulo_mn"
       src="./telemidia:introducao.titulo_mn.txt" descriptor-list="d2"/>
      <text id="titulo_tv" uid="./telemidia:introducao.titulo_tv"
       src="./telemidia:introducao.titulo_tv.txt" descriptor-list="d6"/>
      <text id="introducao_menu" uid="./telemidia:introducao.menu"
       src="./telemidia:introducao.menu.txt" descriptor-list="d1">
        <text-anchor id="menu_projetos" text="projetos" position="10"/>
        <text-anchor id="menu_equipe" text="equipe" position="22"/>
      </text>
      <img id="introducao_imagem" uid="./telemidia:introducao.imagem"
       src="./telemidia:introducao.imagem.gif" descriptor-list="d3">
        <img-anchor id="img_projetos" left="0" top="10" width="30" height="25"/>
        <img-anchor id="img_equipe" left="0" top="35" width="30" height="25"/>
      </img>
      <audio id="introducao_audio" uid="./telemidia:introducao.audio"
       src="./telemidia:introducao.audio.aiff" descriptor-list="d4 d5"/>
     <!-- ... -->
  </doc-body>
</ncl>
```

Figura 14 – Exemplo de definição de objetos de mídia e âncoras específicas em NCL

Em NCL, uma composição pode conter objetos de mídia e composições[16], assim como os relacionamentos entre eles. Elas podem ser de quatro tipos: composição de estruturação (*context*), composição paralela (*par*), composição seqüencial (*seq*) ou composição alternativa (*switch*). Composições de estruturação permitem a definição da estrutura lógica do documento, sem nenhuma semântica de apresentação embutida. Composições paralelas e seqüenciais definem a forma de apresentação dos seus componentes. Numa composição paralela, os componentes são exibidos simultaneamente, enquanto em uma composição seqüencial, os componentes são apresentados em série. A composição alternativa provê meios para definir comportamentos alternativos para a apresentação de um documento. Nela podem ser definidos componentes dentre os quais apenas um será selecionado para ser exibido. Essa seleção é feita em tempo de execução, baseada em testes sobre parâmetros da plataforma, especificados nos atributos dos componentes e no formatador.

A Figura 15 ilustra uma composição de estruturação "telemidia" que contém duas composições paralelas e uma outra composição de estruturação. A composição paralela "equipe", por sua vez, contém uma composição seqüencial aninhada.

[16] A única restrição é que uma composição não pode estar recursivamente contida nela mesma.

Todos os elementos que representam objetos de mídia ou composições podem ser reusados em um mesmo documento ou em documentos diferentes. Para isso, eles devem especificar dois atributos obrigatórios:

- *id* – esse atributo é útil para identificar o componente dentro de um documento.
- *uid* – esse atributo deve identificar a URI correspondente ao componente.

Quando o componente estiver sendo reusado, o autor deve especificar somente seus atributos *id* e *uid*. Além disso, o nome do elemento utilizado para indicar reuso é *nomeElemento-ref*, como por exemplo, para reutilizar um componente tipo texto, deve-se utilizar o elemento *text-ref*. Uma observação importante é que nenhum atributo do componente reusado pode ser alterado em outro lugar diferente de onde ele foi especificado.

O reuso proporciona inúmeras vantagens, entre elas a facilidade de manutenção e a possibilidade de reusar, além dos dados, as estruturas de composições e os relacionamentos.

```
<ncl id="/.telemidia:apresentacao">
  <head>
    <!-- ... -->
  </head>
  <doc-body>
    <context id="telemidia" uid="./telemidia:apresentacao">
      <par id="introducao" uid="./telemidia:introducao">
        <!-- ... -->
      </par>
      <context id="projetos" uid="./telemidia:projetos">
        <text id="projetos_descricao" uid="./telemidia:projetos.descricao"
          src="./telemidia:projetos.descricao.txt" descriptor-list="d1 d3">
        <audio id="projetos_audio" uid="./telemidia:projetos.audio"
          src="./telemidia:projetos.audio.aiff" descriptor-list="d4 d5"/>
      </context>
      <par id="equipe" uid="./telemidia:equipe" descriptor-list="d3">
        <text id="equipe_descricao" uid="./telemidia:equipe.descricao"
          src="./telemidia:equipe.descricao.txt" descriptor-list="d3">
        <img id="equipe_imagem" uid="./telemidia:equipe.imagem"
          src="./telemidia:equipe.imagem.gif" descriptor-list=d1">
        <seq id="equipe_seq" uid="./telemidia:equipe.seq">
          <video id="equipe_video1" uid="./telemidia:equipe.video1"
            src="./telemidia:equipe.video1.mov" descriptor-list="d7">
          <video id="equipe_video2" uid="./telemidia:equipe.video2"
            src="./telemidia:equipe.video2.mov" descriptor-list="d7">
        </seq>
      </par>
      <!-- ... -->
    </context>
  </doc-body>
</ncl>
```

Figura 15 – Exemplo de definição de composições em um documento NCL

A linguagem NCL permite a definição de elos que representam relacionamentos complexos entre dois ou mais componentes do documento. Um elo é composto por um conjunto de pontos terminais de origem, um conjunto de pontos terminais de destino e um ponto de encontro. O conjunto de pontos terminais de origem e destino definem eventos em componentes que constituem as origens e os destinos do elo. Cada ponto terminal pode especificar os atributos *id*, *node-list*, *anchor*, *event*, *attr-name* e *descriptor-list*. O atributo *id* tem o significado usual. O atributo *node-list* especifica uma lista de valores de atributos *id* correspondentes à seqüência de componentes aninhados que deve ser percorrida até encontrar o componente do documento que atua como origem ou destino do elo (último valor de *id* especificado na lista). O atributo *anchor* especifica o valor do atributo *id* de uma âncora contida no último componente identificado por *node-list*. O atributo *event* define um evento associado à âncora. Seus possíveis valores são `presentation`, `selection`, ou `attribution`, representando, os eventos definidos pelo modelo NCM [Soar00]. Segundo o modelo, um evento pode ser a apresentação ou a seleção de uma determinada âncora ou ainda a atribuição de um valor a um dos atributos de um componente. Se o evento for de atribuição, *attr-name* deve identificar o nome do atributo correspondente. O atributo *descriptor-list* especifica uma lista de descritores que poderão ser associados ao último componente identificado por *node-list*, permitindo especificar como será a apresentação desse componente quando for feita uma navegação através desse elo. Para ilustrar a definição de elos em NCL, a Figura 16 especifica um elo (*link*) denominado “elo1”, definido na composição “telemidia”, relacionando componentes recursivamente contidos nessa composição. Esse elo possui dois pontos terminais de origem (*source-ep*), um relativo à seleção da âncora “menu_projetos” do componente “introducao_menu” e outro relativo à seleção da âncora “img_projetos” do componente “introducao_imagem”, e dois pontos terminais de destino (*target-ep*) relativos à apresentação de todo o conteúdo dos componentes “projetos_descricao” e “projetos_audio”.

O ponto de encontro de um elo (*meeting-point*) define um conjunto de condições (*condition*) e um conjunto de ações (*action*). Condições dizem respeito aos pontos terminais de origem e ações são operações que devem ser executadas nos pontos terminais de destino. Essa definição de ponto de encontro representa um relacionamento causal entre

os pontos terminais do elo. Caso o elo represente um relacionamento de restrição, apenas as condições devem ser definidas.

O elemento *condition* representa condições que avaliam valores booleanos, podendo ser simples (*simple-condition*) ou compostas (*compound-condition*). Toda condição composta (*compound-condition*) é formada por uma ou mais condições simples e uma expressão lógica relacionando as condições (*expression_condition*), através dos operadores AND, OR e NOT. Uma condição composta é satisfeita se sua expressão for verdadeira.

No "elo1", ilustrado na Figura 16, a expressão da condição composta (*expression-condition*) foi definida usando o operador OR, e será verdadeira se um dos dois eventos de seleção especificados pelos pontos terminais de origem do elo acontecer.

```
<ncl id="/.telemidia:apresentacao">
  <head>
    <!-- ... -->
  </head>
  <doc-body>
   <context id="telemidia" uid="./telemidia:apresentacao">
     <!-- ... -->
     <link id="elo1">
       <source-ep id="sep_menu_projetos" node-list="introducao introducao_menu"
        anchor="menu_projetos" event="selection" descriptor-list="d1"/>
       <source-ep id="sep_img_projetos" node-list="introducao introducao_imagem"
        anchor="img_projetos" event="selection" descriptor-list="d3"/>
       <target-ep id="tep_projetos_descricao" node-list="projetos projetos_descricao"
        event="presentation" descriptor-list="d3"/>
       <target-ep id="tep_projetos_audio" node-list="projetos projetos_audio"
         event="presentation" descriptor-list="d5"/>
       <meeting-point>
         <condition>
           <compound-condition>
             <simple-condition id="sc_menu_projetos" previous-SEP="sep_menu_projetos"/>
             <simple-condition id="sc_img_projetos" previous-SEP="sep_img_projetos"/>
             <expression-condition exp="sc_menu_projetos OR sc_img_projetos"/>
           </compound-condition>
         </condition>
         <action>
           <compound-action>
             <simple-action id="sa_projetos_descricao" TEP="tep_projetos_descricao" action-name="start"/>
             <simple-action id="sa_projetos_audio" TEP="tep_projetos_audio" action-name="start"/>
             <expression-action exp="PAR(sa_projetos_descricao, sa_projetos_audio)"/>
           </compound-action>
         </action>
       </meeting-point>
     </link>
    <!-- ... -->
   </context>
  </doc-body>
</ncl>
```

Figura 16 – Exemplo de definição de um elo multiponto em NCL

O elemento *action* representa as ações que devem ser executadas nos pontos terminais de destino de um elo causal. Elas podem ser simples (*simple-action*) ou compostas (*compound-action*).

As ações simples podem ser aplicadas a pontos terminais que definem eventos de apresentação (ações `prepare, start, stop, pause, resume, abort`) ou a pontos terminais que definem eventos de atribuição (ações `relative-assign, absolute-assign, enable, disable, activate`). Esses valores são definidos através do atributo *action-name* do elemento *simple-action*. A semântica de cada um desses valores pode ser encontrada em [Soar00].

Toda ação composta (*compound-action*) é formada por uma ou mais ações simples e uma expressão relacionando as ações (*expression_action*) através dos operadores PAR e SEQ. Esses operadores definem a ordem de execução de cada ação simples, indicando se elas serão iniciadas em paralelo ou respeitando alguma ordem predefinida.

No "elo1", ilustrado na Figura 16, a expressão da ação composta (*expression-action*) foi definida usando o operador PAR, indicando que a apresentação do texto e do áudio definidos como destinos do elo devem ser iniciados em paralelo, usando os descritores especificados nos respectivos pontos terminais.

Nesta seção, foram apresentadas algumas características da linguagem NCL, assim como as vantagens da sua utilização. Uma visão mais bem detalhada da linguagem e de todas as suas características pode ser encontrada em [AnSo00a].

3.4.4. Listagem da DTD NCL

```
<!-- Begin of DTD -->
<!-- ================== General entities ================== -->

<!-- System Attributes -->
<!ENTITY % system-attribute "
      system-bitrate               CDATA             #IMPLIED
      system-language              CDATA             #IMPLIED
      system-required              NMTOKEN           #IMPLIED
      system-screen-size           CDATA             #IMPLIED
      system-screen-depth          CDATA             #IMPLIED
      system-captions              (on|off)          #IMPLIED
      system-overdub-or-caption    (caption|overdub) #IMPLIED">


<!-- ==================== NCL Document ==================== -->

<!ELEMENT ncl     (head?, doc-body) >
<!ATTLIST ncl
      id               ID          #IMPLIED>
```

```
<!--================= Head Element ======================-->

<!ELEMENT head    (descriptors-set?,layout?,descriptors-set?)?>
<!ATTLIST head
      id                ID          #IMPLIED>


<!--================== Layout Element ======================-->

<!ELEMENT layout ANY>
<!ATTLIST layout
      id                ID          #IMPLIED
      type              CDATA       "text/ncl-basic-layout">


<!--================= Root-layout Element ====================-->

<!ELEMENT root-layout EMPTY>
<!ATTLIST root-layout
      id                ID          #REQUIRED
      title             CDATA       #IMPLIED
      height            CDATA       #IMPLIED
      width             CDATA       #IMPLIED
      background-color  CDATA       #IMPLIED
      device            CDATA       #IMPLIED>


<!--================ Region Element ======================-->

<!ELEMENT region EMPTY>
<!ATTLIST region
      id                ID          #REQUIRED
      title             CDATA       #IMPLIED
      height            CDATA       #IMPLIED
      width             CDATA       #IMPLIED
      background-color  CDATA       #IMPLIED
      left              CDATA       "0"
      top               CDATA       "0"
      z-index           CDATA       "0"
      fit               (hidden|fill|meet|scroll|slice)     "hidden"
      root-layout       IDREF       #IMPLIED>


<!-- ================= Descriptors-Set Element ================= -->

<!ELEMENT descriptors-set (descriptor)+ >
<!ATTLIST descriptors-set
      id                ID          #IMPLIED>


<!-- =========== Descriptor Element =============== -->

<!ELEMENT descriptor EMPTY >
<!ATTLIST descriptor
      id                ID          #REQUIRED
      title             CDATA       #IMPLIED
      player            CDATA       #IMPLIED
      dur               CDATA       #IMPLIED
      repetitions       CDATA       #IMPLIED
      left              CDATA       #IMPLIED
      top               CDATA       #IMPLIED
```

```
      width             CDATA       #IMPLIED
      height            CDATA       #IMPLIED
      bk-color          CDATA       #IMPLIED
      z-index           CDATA       #IMPLIED
      fit               (fill|hidden|meet|scroll|slice) #IMPLIED
      fill              (freeze|remove) #IMPLIED
      region            IDREF       #IMPLIED
      root-layout       IDREF       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      %system-attribute;>


<!-- ================ Doc-body Element ================ -->

<!ELEMENT doc-body (context|par|seq|audio|video|text|img|switch)>
<!ATTLIST doc-body
      id                ID          #IMPLIED>




<!-- ================ Context Element ================ -->

<!ELEMENT context ( (context|par|seq|audio|video|text|img|switch|
                     composition-anchor|link)*, (presentation)? )>
<!ATTLIST context
      id                CDATA       #REQUIRED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED >


<!-- ================ Context-Ref Element ================ -->

<!ELEMENT context-ref (composition-anchor)*>
<!ATTLIST context-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--================ Parallel Element ==================-->

<!ELEMENT par  ( (context|par|seq|audio|video|text|img|switch|
                    composition-anchor|link)*, (presentation)? )>
<!ATTLIST par
      id                CDATA       #REQUIRED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      endsync           CDATA       "last"
      descriptor-list   IDREFS      #IMPLIED>


<!--================ Parallel-Ref Element ==================-->
```

```
<!ELEMENT par-ref  (composition-anchor)*>
<!ATTLIST par-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--============== The Sequential Element ==================-->

<!ELEMENT seq  ( (context|par|seq|audio|video|text|img|switch|
                  composition-anchor|link)*, (presentation)? )>
<!ATTLIST seq
      id                CDATA       #REQUIRED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>




<!--============== The Sequential-Ref Element ==================-->

<!ELEMENT seq-ref  (composition-anchor)*>
<!ATTLIST seq-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--=================== The Switch Element ================-->

<!ELEMENT switch (context|par|seq|audio|video|text|img|switch)*>
<!ATTLIST switch
      id                CDATA       #REQUIRED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>


<!--================= The Switch-Ref Element ================-->

<!ELEMENT switch-ref EMPTY>
<!ATTLIST switch-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--================ Presentation Element ==================-->

<!ELEMENT presentation  (object)*>
<!ATTLIST presentation
      id                CDATA       #IMPLIED>


<!--================ Object Element ==================-->
```

```
<!ELEMENT object    EMPTY>
<!ATTLIST object
      id                CDATA       #IMPLIED
      object-id         CDATA       #REQUIRED
      descriptor-list   IDREFS      #IMPLIED>


<!--=============== Media Object Elements ====================-->

<!ENTITY % mo-attributes "
      id                CDATA       #REQUIRED
      uid               CDATA       #IMPLIED
      src               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      alt               CDATA       #IMPLIED
      longdesc          CDATA       #IMPLIED
      type              CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED">



<!ELEMENT audio         (audio-anchor)*>
<!ELEMENT img           (img-anchor)*>
<!ELEMENT video         (video-anchor)*>
<!ELEMENT text          (text-anchor)*>

<!ATTLIST audio         %mo-attributes;>
<!ATTLIST img           %mo-attributes;>
<!ATTLIST video         %mo-attributes;>
<!ATTLIST text          %mo-attributes;>


<!--=============== Media Object Ref Elements ==============-->

<!ENTITY % mo-ref-attributes "
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED">


<!ELEMENT audio-ref           (audio-anchor)*>
<!ELEMENT img-ref             (img-anchor)*>
<!ELEMENT video-ref           (video-anchor)*>
<!ELEMENT text-ref            (text-anchor)*>

<!ATTLIST audio-ref           %mo-ref-attributes;>
<!ATTLIST img-ref             %mo-ref-attributes;>
<!ATTLIST video-ref           %mo-ref-attributes;>
<!ATTLIST text-ref            %mo-ref-attributes;>


<!-- ============== Text-Anchor Element ============== -->

<!ELEMENT text-anchor EMPTY >
<!ATTLIST text-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      text              CDATA       #REQUIRED
      position          CDATA       #IMPLIED
```

```
      case-sensitive    CDATA       'false'>


<!-- ============== Audio-Anchor Element ============== -->

<!ELEMENT audio-anchor EMPTY >
<!ATTLIST audio-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      first-sample      CDATA       #IMPLIED
      last-sample       CDATA       #IMPLIED>


<!-- ============== Video-Anchor Element ============== -->

<!ELEMENT video-anchor EMPTY >
<!ATTLIST video-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      first-frame       CDATA       #IMPLIED
      last-frame        CDATA       #IMPLIED
      left              CDATA       '0'
      top               CDATA       '0'
      width             CDATA       'right'
      height            CDATA       'bottom'
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED>


<!-- ============== Img-Anchor Element ============== -->

<!ELEMENT img-anchor EMPTY >
<!ATTLIST img-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      left              CDATA       '0'
      top               CDATA       '0'
      width             CDATA       'right'
      height            CDATA       'bottom'>


<!-- ============== Composition-Anchor Element ============== -->

<!ELEMENT composition-anchor EMPTY >
<!ATTLIST composition-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      node-list         CDATA       #REQUIRED>


<!-- ================== Link Element ================== -->

<!ELEMENT link (source-ep+, target-ep*, meeting-point)  >
<!ATTLIST link
      id                CDATA       #IMPLIED
      title             CDATA       #IMPLIED >


<!-- ========== Source-EP Element ========== -->
```

```
<!ELEMENT source-ep EMPTY >
<!ATTLIST source-ep
      id                CDATA       #REQUIRED
      node-list         CDATA       #IMPLIED
      anchor            CDATA       #IMPLIED
      event       (presentation|selection|attribution)  'selection'
      attr-name         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>


<!-- ========== Target-EP Element =========== -->

<!ELEMENT target-ep EMPTY >
<!ATTLIST target-ep
      id                CDATA       #REQUIRED
      node-list         CDATA       #IMPLIED
      anchor            CDATA       #IMPLIED
      event             (presentation|attribution)    'presentation'
      attr-name         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>







<!-- =========== Meeting-Point Element ================ -->

<!ELEMENT meeting-point (condition, action?) >


<!-- =============== Condition Element ================ -->

<!ELEMENT condition ((simple-condition, delay-condition?) |
                     compound-condition) >


<!-- ============= Simple-Condition Element ============== -->

<!ELEMENT simple-condition EMPTY >
<!ATTLIST simple-condition
      id                CDATA       #REQUIRED
      previous          (true)      #IMPLIED
      previous-SEP      CDATA       #IMPLIED
      previous-function  (state|occurrence|repeat|attribute)  'state'
      previous-operator  (eq|dif|gt|get|lt|let)  'eq'
      previous-value    CDATA       'prepared'
      previous-SEP2     CDATA       #IMPLIED
      previous-type-attr (long | short | string) 'long'
      current           (true)      #IMPLIED
      current-SEP       CDATA       #IMPLIED
      current-function  (state|occurrence|repeat|attribute)  'state'
      current-operator  (eq|dif|gt|get|lt|let)  'eq'
      current-value     CDATA       'occurring'
      current-SEP2      CDATA       #IMPLIED
      current-type-attr (long|short|string)  'long' >


<!-- ========== Compound-Condition Element =========== -->

<!ELEMENT compound-condition (simple-condition+, delay-condition*,
```

```
                              expression-condition)>

<!-- ========= Expression-Condition Element ========== -->

<!ELEMENT expression-condition EMPTY>
<!ATTLIST expression-condition
      exp               CDATA       #REQUIRED>


<!-- ================ Delay-condition Element ================== -->

<!ELEMENT delay-condition EMPTY >
<!ATTLIST delay-condition
      id                CDATA       #REQUIRED
      t-begin           CDATA       #REQUIRED
      t-end             CDATA       #IMPLIED>


<!-- ================ Action Element ================ -->

<!ELEMENT action ((simple-action,delay-action?) | compound-action) >




<!-- ============= Simple-Action Element ============= -->

<!ELEMENT simple-action EMPTY >
<!ATTLIST simple-action
      id                CDATA       #REQUIRED
      TEP               CDATA       #REQUIRED
      action-name       (prepare|start|stop|pause|resume|
                        abort|relative-assign|absolute-assign|
                        enable|disable|activate)  'start'
      repetitions       CDATA       #IMPLIED
      attr-value        CDATA       #IMPLIED
      type-attr         (long|short|string)  #IMPLIED>


<!-- ============= Compound-Action Element ============ -->

<!ELEMENT compound-action (simple-action+, delay-action*,
   expression-action) >


<!-- =========== Expression-Action Element =========== -->

<!ELEMENT expression-action EMPTY>
<!ATTLIST expression-action
      exp               CDATA       #REQUIRED>


<!-- ================ Delay-action Element ================== -->

<!ELEMENT delay-action EMPTY >
<!ATTLIST delay-action
      id                CDATA       #REQUIRED
      t-min             CDATA       #IMPLIED
      t-opt             CDATA       #REQUIRED
```

```
      t-max             CDATA       #IMPLIED
      cost-function     CDATA       #IMPLIED>

<!-- End of DTD -->
```

3.5. Conversores NCL

Os conversores NCL foram implementados com o intuito de possibilitar a conversão automática no sistema HyperProp de objetos NCM em uma representação Java, de acordo com a atual implementação do sistema, para objetos NCM em uma representação NCL, e vice-versa. Esse capítulo irá descrever a implementação desses conversores, apresentar sua organização modular e seus principais métodos[17]. Ao final do capítulo também será feita uma breve descrição sobre os testes realizados.

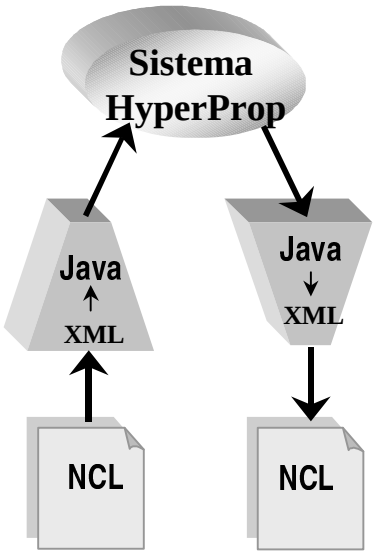


Figura 17 – Conversores NCL

Um objeto NCM em uma representação NCL, ou seja, um documento NCL, é sempre formado por um elemento raiz, denominado *ncl*, que contém dois elementos principais: o *head* e o *doc-body*. O elemento *head* contém informações associadas à apresentação dos elementos contidos no documento e suas disposições espaciais. O *doc-body* representa um repositório que contém todos os outros elementos do documento, tais como elementos que representam objetos de mídia, ou nós terminais, e elementos que representam composições.

[17] A especificação completa sobre do projeto desses conversores pode ser encontrada em [Anto99].

Quando o usuário desejar gerar um documento NCL correspondente a um nó terminal do sistema, por exemplo um vídeo, ele deverá, primeiramente, selecionar esse nó terminal no sistema e, em seguida, solicitar a geração do documento NCL correspondente. Um novo documento será criado e, nesse caso, será inserido no seu elemento *doc-body* um elemento *video* representando o nó terminal selecionado pelo usuário.

Da mesma forma, o usuário também pode solicitar a geração de um documento NCL correspondente a uma composição, como por exemplo, um nó de contexto. No entanto, caso o nó de contexto que o usuário selecionou possua outros nós de composição aninhados em qualquer profundidade, nós terminais e elos, o elemento *doc-body* do documento gerado será formado pelo nó de contexto selecionado e por todos os outros elementos recursivamente nele contidos, ou seja, dado um nó de contexto, no documento NCL serão criados elementos para representar cada um dos seus filhos, os filhos dos seus filhos, e assim por diante.

Na conversão de um documento NCL em um objeto Java também serão criados no sistema elementos correspondentes a todos os elementos especificados no documento. Esses elementos serão inseridos na perspectiva atual do sistema, selecionada pelo usuário.

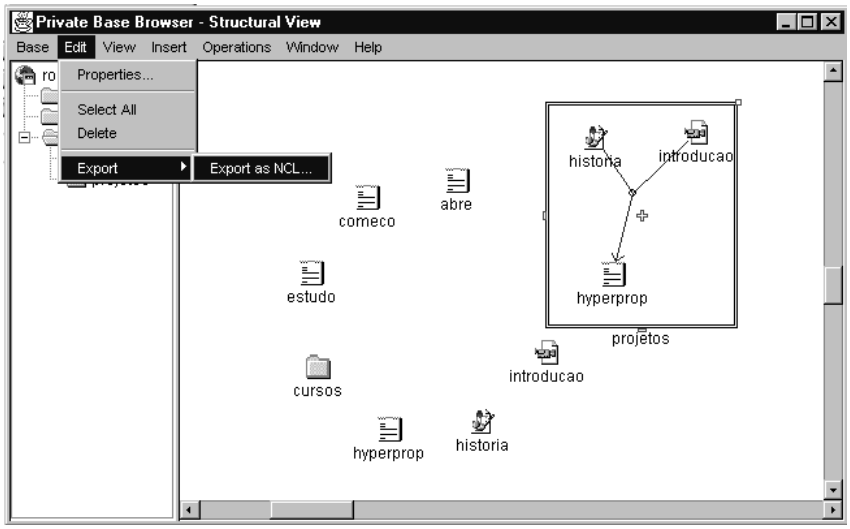
Ambos os conversores foram implementados em Java, seguindo o paradigma de orientação a objetos. Suas hierarquias de classes são bastante simples. Cada conversor é formado por uma classe principal, contendo os métodos responsáveis pela função de conversão, e por algumas outras classes auxiliares que são usadas, por exemplo, para armazenar dados intermediários.

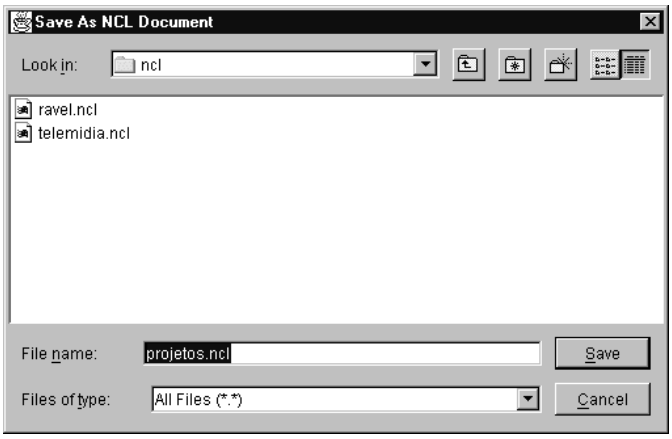
A organização modular dos conversores também é similar. Basicamente, para cada elemento da DTD foi desenvolvido um método correspondente. Esse método é responsável pela leitura ou escrita de todos os atributos do elemento e pela verificação da ordem e número de seus elementos filhos. Além desses métodos que representam os elementos da DTD, cada conversor possui um método principal, responsável pela inicialização da leitura ou escrita do documento. Esses são os únicos métodos públicos das classes, que devem ser chamados pelas janelas do sistema responsáveis pela importação ou exportação de documentos NCL.

O primeiro conversor implementado é responsável pela exportação de documentos NCL, ou seja, ele gera um documento NCL a partir de um objeto Java selecionado no browser de base privada do sistema HyperProp (Figura 18). O documento gerado é compatível com a DTD criada para a linguagem e representa integralmente o objeto selecionado, isto é, contém elementos representando todos os filhos recursivos do objeto.

A importação de documentos NCL para o sistema HyperProp, na forma de objetos Java, é feita pelo segundo conversor (Figura 19). Esse conversor lê um documento NCL e verifica se está de acordo com a DTD da linguagem. Em caso positivo, gera o objeto Java equivalente e o insere na base privada do sistema.

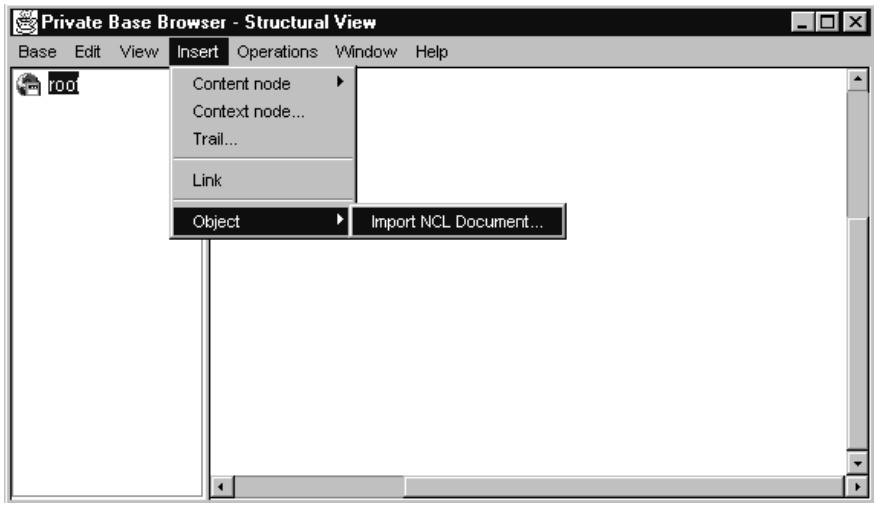
Em ambos os processos de conversão, existe uma etapa na qual é realizada a validação de documentos NCL lidos (importação) e gerados (exportação) pelos conversores. Nessa etapa, um documento NCL é submetido a um parser XML genérico que realiza a sua validação diante da DTD da linguagem, também passada para o parser. Para essa validação foi utilizado um parser da ORACLE, desenvolvido em Java e distribuído gratuitamente na rede [XMLPv2].






```
<?xml version="1.0"?>
<!DOCTYPE ncl SYSTEM "ncl.dtd">
<ncl>
   <head>
      <descriptor-set>
         <descriptor id="d1" player="tm" top="2" left="10"/>
         <descriptor id="d2" player="hyp" top="17" left="30"/>
      <descriptor-set>
   </head>
   <doc-body>
     <context id="ic1" uid="telemidia:projetos" title="projetos" descriptor-list="d1 d2">
        <text id="c1 t1" uid="telemidia:projetos/hyperprop" title="hyperprop" descriptor-list="d1"/>
        <video id="c1 v1" uid="telemidia:projetos/intro" title="introducao" descriptor-list="d1 d2"/>
        <img id="c1 i1" uid="telemidia:projetos/historia" title="historia" descriptor-list="d2"/>
        <link>
          <!-- … --!>
        </link>
     </context>
   </doc-body>
</ncl>
```

Figura 18 – Exportando o contexto "projetos" como um documento NCL




```
<?xml version="1.0"?>
<!DOCTYPE ncl SYSTEM "ncl.dtd">
<ncl>
   <head>
      <descriptor-set>
         <descriptor id="d1" player="tm" top="2" left="10"/>
         <descriptor id="d2" player="hyp" top="17" left="30"/>
      <descriptor-set>
   </head>
   <doc-body>
      <text id="id_t1" uid="telemidia:tm" title="telemidia" descriptor-list="d1 d2"/>
      <!-- … --!>
   </doc-body>
</ncl>
```

Figura 19 – Importando o documento "telemidia.ncl" para o sistema HyperProp

3.5.1. Testes

Esta seção tem como principal finalidade descrever os testes realizados nos programas implementados. Foram feitos exercícios sistemáticos e inspeções nos métodos dos conversores, especialmente naqueles em que a probabilidade de ocorrência de erros era maior, visando garantir a qualidade do software.

Alguns critérios foram seguidos na execução dos testes, com o intuito de otimizar todo o processo. Primeiramente, testou-se cada método isoladamente, verificando se todas as situações internas atendiam à funcionalidade especificada no projeto. Para isso, foram utilizados casos de teste valorados, cujos valores eram relacionados aos atributos e ao conteúdo do elemento referente àquele método. A escolha destes valores não foi livre; foram selecionadas entradas válidas e inválidas baseadas nos requisitos especificados, com o propósito de observar os efeitos produzidos. Pôde-se dessa maneira, analisar e corrigir mais rapidamente eventuais erros de lógica existentes.

A utilização de depuradores de código também foi uma técnica válida para testar alguns métodos do sistema.

Em seguida, foram testadas as relações entre os métodos. Cada teste especificado envolvia no mínimo dois métodos, sendo que o principal objetivo era verificar o comportamento das interfaces. Foram analisados os pedidos de serviços de um método para outro, monitorando seu retorno.

Para o conversor responsável pela exportação de documentos NCL, o último teste realizado foi a submissão do documento gerado a um parser XML genérico que validava o documento diante da DTD especificada para a linguagem.

Como o produto final do conversor responsável pela importação de documentos NCL era um objeto Java do sistema HyperProp, seu último teste foi uma análise visual deste objeto. Depois disso, o primeiro conversor foi utilizado para gerar um documento NCL deste objeto (nessa etapa o primeiro conversor já estava concluído e testado), sendo feita, em seguida, uma comparação entre o documento gerado e o documento original.

4. A Linguagem X-SMIL

Outro objetivo deste trabalho é a extensão da linguagem SMIL, apresentada na Seção 2.5, visando sanar as limitações descritas em [RRMS99] e alcançar as mesmas facilidades proporcionadas pela linguagem NCL.

Neste capítulo, inicialmente, são descritos os principais pontos de extensão sofridos pela linguagem SMIL e, logo após, é feita uma apresentação dos módulos conversores que foram implementados para possibilitar a integração da linguagem SMIL com o sistema HyperProp.

A DTD da nova linguagem SMIL estendida, denominada X-SMIL, está listada no Apêndice A e a especificação completa de todos os seus elementos e atributos está descrita em [AnSo00b].

4.1. Extensões da Linguagem SMIL

Para criar a nova linguagem SMIL estendida, mantendo compatibilidade com a versão inicial, partimos da sua DTD original e acrescentamos a ela novos elementos e atributos de forma a permitir uma maior liberdade e flexibilidade para a autoria de documentos hipermídia.

Pode-se perceber que os novos elementos e atributos acrescentados são similares aos elementos e atributos da NCL. A NCL é uma linguagem que abrange o SMIL, ou seja, tudo que é especificado no SMIL pode ser traduzido para a linguagem NCL. A recíproca não é verdadeira, assim, o que se pretende com a extensão do SMIL é suportar as características adicionais da NCL. Portanto, é necessário adicionar ao SMIL alguns elementos e atributos da NCL que ele não suporta.

É importante frisar que tanto um documento SMIL, quanto um documento NCL, estarão compatíveis com a nova linguagem X-SMIL.

A seguir, será feita uma descrição sucinta dos principais elementos e atributos acrescentados à linguagem SMIL, dando origem à X-SMIL. Note que alguns desses elementos e atributos já foram brevemente explicados na Seção 3.4.3. Portanto, aqui iremos apenas destacar algumas de suas características e funções básicas. Novamente, deve-se lembrar que em [SMIL98], [AnSo00a], [AnSo00b] pode-se encontrar uma especificação completa e detalhada das linguagens SMIL, NCL e X-SMIL.

Entre os elementos acrescentados, encontram-se:

- ***descriptor***
- ***context***
- ***audio-anchor, video-anchor, img-anchor, text-anchor, composition-anchor***
- ***link***

O elemento *descriptor* é útil para especificação de atributos relacionados à apresentação de outros elementos. Para que esse elemento seja associado a outros elementos, também foi necessário adicionar à lista de atributos dos elementos, que poderão ser associados a um descritor, um novo atributo, denominado *descriptor-list*, que deverá conter uma referência para o descritor a ser utilizado.

Através dos atributos do elemento *descriptor*, podem-se, por exemplo, definir mudanças de comportamento em um objeto durante a sua exibição, o que antes era considerado uma limitação do SMIL.

O elemento *context* serve para definir uma estrutura lógica para o documento, independente da sua apresentação, ou seja, nem sempre, essa estrutura será a sua estrutura de exibição do documento. Por exemplo, para se alterar a estrutura temporal do documento, basta acrescentar novos elos de sincronismo, sem precisar alterar a sua organização.

Os elementos *audio-anchor*, *video-anchor, img-anchor, text-anchor* e *composition-anchor* possibilitam a definição de âncoras específicas para cada tipo de elemento. Essas âncoras irão servir como ponto de origem e destino dos elos.

O elemento *link* também foi acrescentado à linguagem X-SMIL. Ele possibilita a definição de todos os elos suportados pelo modelo NCM, tais como os elos n:m de sincronismo e os hiper-elos. Considera-se esse como sendo um dos mais importantes pontos de extensão da linguagem SMIL.

Os principais atributos acrescentados à linguagem foram:

- ***descriptor-list***
- ***id e uid***
- ***device***
- ***root-layout***

O atributo *descriptor-list* foi acrescentado aos elementos que representam objetos de mídia (*audio, video, text, img*) e aos elementos que representam composições de estruturação e sincronização (*context, par, seq*) para possibilitar a associação desses com algum descritor. O valor desse atributo pode ser formado por uma lista de descritores alternativos, dentre os quais, apenas um será escolhido para definir a apresentação do elemento, de acordo com a melhor QoS, conforme definido pelo modelo conceitual NCM.

Os atributos *id* e *uid* são úteis para possibilitar o reuso de elementos, tanto a nível de dados quanto de estruturas, em um mesmo documento ou em documentos diferentes, sem precisar fazer cópias.

Na linguagem SMIL, a maioria dos elementos já possui o atributo *id* especificado na sua lista de atributos. Porém, também para a maioria, esse atributo foi definido como sendo opcional. De acordo com a finalidade que queremos atribuir a esse atributo, sua

especificação passa a ser obrigatória para todos os elementos que podem ser reusados, que são os elementos que representam objetos de mídia e os elementos que representam composições de estruturação e sincronização. O valor do atributo *id* irá identificar uma determinada ocorrência do elemento em um documento. Além do atributo *id*, esses elementos também devem especificar o atributo *uid* que irá referenciar a URN do elemento em questão.

Também é possível especificar, em um documento SMIL, uma janela de exibição onde o documento será apresentado. Essa especificação é feita através do elemento *root-layout*, que deve ser único por documento, ou seja, para o documento SMIL estar correto, ele deve possuir somente um elemento *root-layout* especificado. Portanto, considerando-se que um único documento pode ser apresentado em mais de um dispositivo de saída, por exemplo, o vídeo sendo apresentado em uma TV e o texto em um monitor de computador, seria necessário definir mais de um elemento *root-layout* por documento, sendo um para cada dispositivo. Para solucionar essa limitação, o atributo *device* foi acrescentado ao elemento *root-layout*. Assim, cada elemento *root-layout* deve também especificar um dispositivo de saída que não pode ter sido associado a nenhum outro elemento *root-layout* do mesmo documento. Portanto, o número de elementos *root-layout*, por documento, passa a ser limitado de acordo com o número de dispositivos de saída que serão utilizados para a apresentação do mesmo.

Essa possibilidade de definição de mais de um elemento *root-layout* por documento vai influenciar na definição da apresentação dos elementos, realizada nos atributos do descritor. O descritor deverá possuir, além dos atributos já existentes, um novo atributo contendo a informação associada ao *root-layout* onde deve ser apresentado o elemento. Esse atributo foi denominado *root-layout* e seu valor deve referenciar um elemento *root-layout* especificado no documento .

Além da adição dos atributos mencionados, os atributos *begin*, *end* e *dur* foram modificados para permitir uma maior flexibilidade na definição de início, fim e duração do elemento. Agora esses atributos definem uma função de custo para esticar ou encolher, respectivamente, os tempos de início, fim e duração ótimos do elemento.

Devido às alterações e acréscimos de funções à uma linguagem já existente, algumas definições de atributos tornaram-se redundantes. Por isso, alguns atributos podem ser especificados em mais de um lugar. Este é o caso dos atributos de definição de início e fim. Eles podem ser especificados no próprio elemento ou em um descritor associado a este elemento. No caso de um atributo ser especificado em mais de um lugar, serão aplicadas as regras de precedência definidas na especificação da linguagem.

Resumidamente, foram apresentados nesta seção, alguns elementos e atributos alterados e/ou adicionados à linguagem SMIL para a formação da nova linguagem estendida. Mais uma vez, deve-se lembrar que a definição formal da linguagem X-SMIL pode ser encontrada em [AnSo00b].

4.2. Conversores X-SMIL

Conforme mencionado no início da seção anterior, o principal objetivo da linguagem X-SMIL é estender o padrão SMIL, incorporando a ele novas funcionalidades existentes na linguagem NCL. Esse processo foi realizado partindo-se da DTD SMIL e mantendo-se todos os seus elementos e atributos originais. Com isso, assegura-se que qualquer documento compatível com a DTD SMIL será sempre compatível com a DTD da nova linguagem.

Ao acrescentar novos elementos e atributos, herdados da NCL, na DTD SMIL percebeu-se que os documentos NCL também tornaram-se um subconjunto da X-SMIL. Todas as funcionalidades da NCL estão representadas na X-SMIL, no entanto, deve-se ressaltar que na X-SMIL existem informações que podem ser representadas de maneiras diferentes, uma herdada da linguagem SMIL e outra herdada da NCL. Essa é a principal diferença entre a NCL e a X-SMIL, ou seja, a NCL é um tipo de linguagem X-SMIL “enxuta”, nela não existem redundâncias, havendo somente uma maneira de representar uma informação.

Sendo assim, a X-SMIL é uma nova linguagem que representa tanto um documento SMIL quanto um documento NCL, isto é, SMIL e NCL constituem subconjuntos da X-SMIL.

Tendo em vista as interseções entre as linguagens, um conversor responsável pela tradução de documentos X-SMIL para objetos Java do sistema HyperProp pode receber como entrada documentos X-SMIL, SMIL ou NCL. Por outro lado, um conversor responsável pela tradução de objetos Java para uma forma declarativa, poderá gerar documentos X-SMIL, SMIL ou NCL. Deve-se notar que não faz sentido gerar um documento X-SMIL contendo redundâncias, ou seja, contendo dois elementos ou atributos que representam a mesma informação. Então, os únicos formatos desejáveis na exportação de um objeto Java para a forma declarativa são NCL e SMIL. Conforme apresentado na Seção 3.5, já existe no sistema um conversor que gera documentos NCL, restando somente implementar um conversor que gera documentos SMIL. Porém, deve-se ressaltar que algumas informações podem ser perdidas durante a conversão de objetos Java para documentos SMIL devido ao fato do modelo NCM, base da linguagem NCL e do sistema HyperProp, ser mais abrangente que o modelo SMIL.

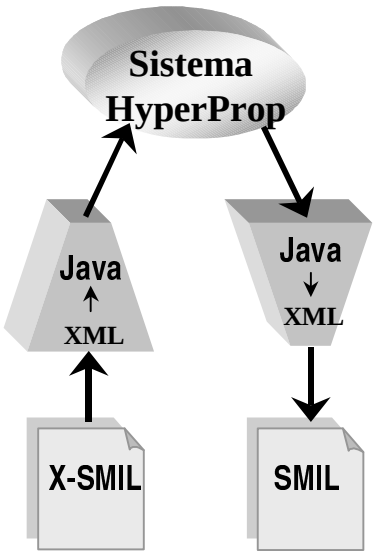


Figura 20 – Conversores X-SMIL

Resumindo, os novos conversores implementados a partir da criação da linguagem X-SMIL foram:

- X-SMIL → Objeto Java HyperProp: responsável pela conversão de documentos X-SMIL em objetos Java do sistema.

- Objeto Java HyperProp → SMIL: responsável pela conversão de objetos Java em documentos SMIL.

Esses conversores tornam possível a autoria e edição de documentos SMIL no ambiente gráfico do sistema HyperProp, possibilitando a integração SMIL-HyperProp.

Devido à grande similaridade da linguagem X-SMIL com a NCL, os conversores implementados também são similares aos conversores NCL. Eles foram implementados em Java e seguem o paradigma de orientação a objetos. Tanto a concepção dessa implementação, quanto os testes realizados seguem o mesmo princípio apresentado na Seção 3.5, onde os conversores NCL foram descritos.

5. Conclusões

Neste capítulo, inicialmente é realizada uma análise comparativa entre as linguagens apresentadas no Capítulo 2 e a linguagem NCL. Em seguida, são apresentadas as contribuições da dissertação e, logo após, são propostos alguns tópicos a serem abordados em trabalhos futuros.

5.1. Comparação com Trabalhos Relacionados

A comparação entre a NCL e as linguagens hipermídia apresentadas no Capítulo 2 será feita baseando-se em algumas características consideradas fundamentais para a especificação declarativa de documentos hipermídia.

Porém, antes de se iniciar essa comparação, são feitos alguns comentários sobre as linguagens HyTime e XHTML.

Apesar de não ter sido detalhada nesta dissertação, HyTime é uma linguagem bastante poderosa que permite escalonar objetos multimídia no tempo e no espaço, de forma bem genérica, provendo facilidades para representar documentos contendo informações estáticas e dinâmicas. Entretanto, como ressaltado na Seção 2.2, seu grande poder de expressão compromete sua usabilidade na Web. Essa linguagem foi abordada na dissertação somente com o intuito de apresentar um exemplo de linguagem formalizada em SGML. Sua

especificação mais detalhada foge ao escopo deste trabalho, podendo ser encontrada em [HyTime97]. Por isso, nesta seção, durante a comparação entre as diversas linguagens, HyTime não será mencionada.

Em relação à linguagem XHTML, deve-se lembrar que ela é simplesmente uma proposta de reformulação do HTML como uma aplicação XML. Nessa proposta, nenhuma modificação ou extensão é sugerida, sendo assim, é normal que ela apresente a mesma simplicidade e possua as mesmas limitações que o padrão HTML. Da mesma forma que HyTime, esta linguagem também não será citada durante a comparação entre as linguagens hipermídia, apresentada a seguir. Informações mais detalhadas sobre XHTML estão disponíveis em [XHTML99].

Por tudo mencionado, a comparação feita a seguir considera as linguagens HTML+TIME, SMIL e, Madeus, além da própria NCL. Objetivando uma melhor comparação, a análise é realizada por características de documentos hipermídia e não entre linguagens como um todo.

- ***reuso***

Em todas as linguagens em questão, o conteúdo dos objetos de mídia não está embutido no documento hipermídia, sendo identificado através de referências descritas na forma de URIs/URNs. Tal separação possibilita o reuso tanto em um mesmo documento, como em documentos diferentes, sendo essa a forma básica de reuso provida por todas as linguagens.

De fato, as quatro linguagens oferecem maneiras satisfatórias, e muito semelhantes, para prover o reuso de conteúdo. No entanto, ao se definir um documento hipermídia, deseja-se mais que isso. Deseja-se poder reutilizar pequenas partes do documento, com todas as definições de elementos e relacionamentos, ou até mesmo reutilizar um objeto de mídia simples, com um conteúdo e com algumas definições básicas, por exemplo relacionadas ao nome, tipo e autor do objeto. As linguagens HTML+TIME e SMIL são bastante limitadas em relação ao reuso em seus documentos, possibilitando somente o reuso de conteúdo. Como descrito em [RRMS99], na linguagem SMIL, as estruturas são reutilizadas somente

através de cópias, o que dificulta a manutenção. O mesmo acontece na linguagem HTML+TIME.

As linguagens Madeus e NCL provêem, além do reuso de conteúdo, a facilidade de reutilização de estruturas. Em Madeus, essa facilidade está relacionada à definição de componentes compostos, que são os elementos que definem uma estruturação do documento. Esses elementos compostos podem ser reutilizados na definição de novos documentos ou dentro de um mesmo documento. Dessa forma, é possível reutilizar partes estruturais do documento, porém, para se reusar um objeto de mídia simples, este deve ser definido em um elemento composto.

A NCL é mais abrangente que as demais linguagens analisadas. Nos seus documentos, é possível reusar tanto elementos simples, como grupamentos e outros tipos de elementos que definem a apresentação do documento. Na NCL, o reuso está associado ao atributo de identificação do elemento que possui um valor único (URN). Assim, podem ser reusados tanto objetos de mídia independentes quanto grupamentos de estruturação, paralelos, seqüenciais e alternativos. Além desses elementos, em um documento NCL podem ser reusados os elementos descritores, que possuem informações relacionadas à apresentação dos componentes do documento (como, por exemplo, a região espacial onde o componente deve ser exibido), e as regiões espaciais para apresentação do documento, que são também definidas em um elemento a parte.

- ***disposição espacial***

Todas as linguagens apresentadas neste trabalho permitem especificar, de alguma forma, a disposição espacial dos componentes dos documentos.

A linguagem HTML+TIME, por derivar do HTML, herda sua forma de disposição espacial dos componentes, possibilitando também o uso de folhas de estilo (*style sheets*). No entanto, a forma de disposição espacial permitida pelo HTML é bastante restrita [HTML98].

SMIL e NCL possuem uma linguagem para disposição espacial de elementos em seus documentos bastante similar. Podem ser especificadas regiões de apresentação onde os

objetos visuais do documento serão exibidos. Também é possível especificar uma janela de apresentação do sistema, na qual todas as regiões deverão estar contidas. Ao contrário do SMIL, na NCL essa janela de apresentação não precisa ser única. Pode-se especificar uma janela de apresentação diferente para cada dispositivo de saída utilizado pelo sistema para a exibição do documento.

Outra diferença entre SMIL e NCL está relacionada à forma como é determinada a disposição espacial de um determinado componente do documento, ou seja, como um objeto visual é associado a uma região de apresentação. De acordo com o padrão SMIL, os elementos que representam objetos de mídia possuem um atributo *region* cujo valor é uma referência para um elemento *region* definido no cabeçalho do documento. Na linguagem NCL, os dados relativos à apresentação estão separados da definição do componente. Com isso, é possível, por exemplo, definir apresentações diferentes para um mesmo elemento, favorecendo o reuso desses elementos. Toda informação relacionada à apresentação de um componente deve estar especificada no elemento *descriptor*. Assim, para associar uma região de apresentação a um componente, deve-se primeiro especificar o elemento *region*, em seguida, deve-se referenciar esse elemento através de um elemento *descriptor* e depois deve-se associar esse elemento *descriptor* ao componente ou objeto desejado. É importante lembrar que, por ser possível especificar uma janela de apresentação para cada dispositivo de exibição utilizado pelo sistema, ao se especificar uma região na NCL também é necessário indicar em qual janela de apresentação essa região deve ser definida.

Comparando com o SMIL, a linguagem NCL é bastante flexível em relação à definição de regiões e janelas de apresentação. Existem formas de sobrepor os valores referenciados, permitindo, por exemplo, o reuso de regiões em diferentes janelas de apresentação. A possibilidade de reusar as informações associadas à disposição espacial de componentes, sem reusar o conteúdo dos componentes propriamente ditos, constitui uma vantagem apresentada pela NCL e não encontrada nas outras linguagens estudadas. Informações mais detalhadas a respeito da especificação da disposição espacial dos componentes em um documento NCL e seu reuso, podem ser obtidas em [AnSo00a].

Em Madeus, a organização espacial dos componentes dos documentos pode ser determinada através de restrições entre eles. Essas restrições são baseadas em operadores

que posicionam, relativamente, os componentes nos documentos. Conforme mencionado na Seção 2.4, a especificação de relacionamentos temporais ou espaciais entre componentes através de restrições facilita a autoria de documentos hipermídia, uma vez que os autores não precisam saber, por exemplo, a posição de um determinado componente no eixo y, necessitando apenas especificar que o componente *A* está alinhado ao topo em relação ao componente *B*. A possibilidade de definição de disposição espacial relativa dos componentes do documento é uma grande vantagem apresentada pela linguagem Madeus, inexistente em todas as outras linguagens estudadas, que posicionam espacialmente seus componentes de forma absoluta e não relativa.

- ***sincronização temporal***

Todas as linguagens analisadas nesta seção apresentam suporte para definição de relacionamentos de sincronização temporal entre componentes dos documentos. No entanto, variam os tipos e a forma como os relacionamentos podem ser especificados. De um modo geral, os relacionamentos temporais podem ser classificados em duas categorias: relacionamentos causais e relacionamentos de restrição. Quanto ao formato de especificação, os relacionamentos são normalmente descritos através de estruturas de composições ou relações (elos).

Em documentos Madeus, a sincronização temporal pode ser especificada através de relações que determinam restrições ou relacionamentos causais entre componentes do documento. A autoria é bastante facilitada pois a linguagem oferece um conjunto de operadores de alto nível predefinidos representando esses relacionamentos. No entanto, apesar da simplicidade, nem todos os tipos de relacionamentos desejáveis num documento hipermídia podem ser especificados. Por exemplo, em um documento Madeus não é possível dizer que: se o usuário selecionar o componente A enquanto o componente B estiver ocorrendo, então deve ser exibido o componente C. Além disso, em Madeus os relacionamentos devem ser definidos somente entre elementos contidos em um mesmo componente composto (composição). Na linguagem, relações de causalidade também podem ser expressas através de elos simples, semelhante aos hiper-elos do HTML.

Madeus, não define nenhum tipo de composição para definição de sincronismo. As únicas composições existentes são as composições de estruturação.

Apesar de ser possível especificar todos os tipos de relacionamento entre componentes utilizando relações de restrição e causalidade mais genéricas, algumas vezes a especificação desses relacionamentos pode ser bastante trabalhosa do ponto de vista da autoria declarativa de documentos. Visando facilitar essa autoria, algumas linguagens definiram composições que apresentam semânticas de sincronização entre seus filhos, como é o caso das linguagens SMIL, HTML+TIME e NCL.

Em HTML+TIME e SMIL, a especificação de comportamentos temporais dos componentes de seus documentos é bastante semelhante devido ao fato da linguagem HTML+TIME ter sido definida baseando-se nos conceitos temporais da linguagem SMIL. Essas linguagens não permitem a especificação de restrições entre seus componentes e os relacionamentos de causalidade podem ser expressos através de grupamentos paralelos e seqüenciais, com semânticas de paralelização e seqüenciação da exibição dos seus elementos filhos; ou através de elos, que são estabelecidos entre âncoras de origem e âncoras de destino dos elementos. A navegação por um elo é sempre disparada pela ação do usuário e eles são sempre 1:1, ou seja, associam uma única âncora de origem a uma única âncora de destino. As âncoras de origem ou destino podem representar um elemento inteiro ou identificar sub-regiões espaciais ou temporais de elementos objetos de mídia.

Além da especificação de elos causais e de grupamentos *par* e *seq*, em SMIL e HTML+TIME podem ser definidos alguns atributos contendo valores para início, duração e fim da exibição de um elemento.

Similar às linguagens HTML+TIME e SMIL, a NCL também permite que a especificação de comportamentos temporais dos componentes seja definida através de elos causais, atributos e composições (ou grupamentos). A NCL apresenta uma flexibilidade adicional possibilitando especificar uma função de custo que informa os valores mínimo, máximo e ótimo para seus atributos de tempo, bem como o custo envolvido no ajuste deste tempo quando ele for diferente do valor ótimo.

Os elos suportados pela NCL, são os mesmos definidos pelo modelo NCM (Seção 3.2). Eles são bastante genéricos. Podem ser: hiper-elos, disparados pela ação do usuário, ou elos de sincronismo, elos que são disparados no tempo sem uma ação explícita do usuário. Permitem associar *n* âncoras de origem a *m* âncoras de destino, ou seja, são *n:m*. Além disso, essas âncoras são específicas para cada tipo de mídia, conforme ressaltado na Seção 3.4.3, e possibilitam identificar elementos inteiros ou sub-regiões espaciais e temporais em cada mídia. Por exemplo, em uma mídia do tipo texto pode-se definir uma âncora na seqüência de caracteres *"clique aqui"* que se encontra na posição 34[18] do arquivo. Pode-se também especificar uma âncora em uma mídia áudio que começa na quinta amostra e termina na décima amostra da mídia. Em uma imagem, pode-se criar uma âncora delimitada por um retângulo cujo ponto superior esquerdo é (2, 2), altura igual a 10 e largura igual a 30.

Além disso, na NCL também podem ser definidos elos que representam restrições entre os componentes do documento. Sendo que esses elos podem ser estabelecidos entre quaisquer componentes do documento.

- ***representação de objetos de mídia***

A NCL e as demais linguagens apresentadas nesta dissertação possuem elementos específicos para representação de objetos de mídia diferentes. Em algumas dessas linguagens os elementos são genéricos, enquanto em outras, pode-se especificar elementos que representam mídias mais específicas. Apesar de haver diferenças nas formas de representação desses objetos, todas as linguagens oferecem um suporte mínimo desejado, não tendo sido identificadas vantagens significativas em qualquer uma delas.

---

[18] Este número representa quantos caracteres existem entre o início do arquivo texto e a seqüência de caracteres especificada. Se o conteúdo do arquivo texto for alterado e a seqüência de caracteres que representa a âncora não estiver na posição indicada, um algoritmo de ajuste deve ser executado para obter a ocorrência da seqüência especificada, mais próxima da posição indicada, como sendo a âncora a ser utilizada. A especificação da posição da sequência de caracteres no arquivo é opcional, porém, caso ela não seja especificada, a âncora será associada à primeira ocorrência da sequência encontrada no texto.

- ***grupamento de elementos***

Madeus permite a definição de componentes compostos, que podem ser formados por outros componentes simples e compostos. Esta é considerada uma das principais vantagens oferecidas pela sua linguagem. Esses componentes, além de propiciarem uma organização hierárquica dos documentos, possibilitam o reuso de elementos e a fácil alteração de documentos. Componentes compostos podem ser reutilizados em outras partes de um mesmo documento, bem como em documentos diferentes, sendo que, quando reutilizados, levam consigo seus componentes e toda a sua estrutura espacial e temporal. Além disso, devido ao fato da estrutura de um documento Madeus ser baseada em restrições, quaisquer componentes podem ser inseridos ou retirados de um componente composto sem a necessidade de maiores modificações no restante do documento.

Em HTML+TIME e SMIL, apesar de não existirem grupamentos de estruturação, podem ser definidos grupamentos seqüenciais e paralelos, nos quais todos os elementos filhos são exibidos, respectivamente, em série ou em paralelo. Também podem ser definidos grupamentos alternativos, nos quais apenas um elemento filho é selecionado para ser exibido, de acordo com características da plataforma de apresentação. Esses grupamentos alternativos permitem que um mesmo documento seja exibido de maneiras diferentes.

A linguagem NCL, além de permitir a especificação de grupamentos seqüenciais, paralelos e alternativos, permite a especificação de grupamentos de estruturação do documento, semelhante aos componentes compostos existentes na linguagem Madeus. Percebe-se, mais uma vez, que a linguagem NCL engloba vantagens apresentadas por Madeus e por SMIL e HTML+TIME.

Como pôde ser visto, diversas características de documentos hipermídia são suportadas pelas linguagens apresentadas durante o trabalho. No entanto, essas características não são totalmente implementadas em todas as linguagens. Cada linguagem implementa apenas um subconjunto delas e, algumas vezes, de forma restrita.

A linguagem NCL, ao contrário das outras, abrange de forma satisfatória todas as características apresentadas. Isto se deve à generalidade do NCM, modelo conceitual hipermídia que serviu de base para a especificação da linguagem.

Segue abaixo um resumo de algumas vantagens oferecidas pela NCL, salientadas nesta seção:

- permite especificar áreas para apresentação dos documentos em dispositivos de saída diferentes;
- possibilita o reuso de informações associadas à disposição espacial de elementos;
- apresenta flexibilidade na especificação do comportamento temporal dos componentes;
- permite definir grupamentos seqüenciais, paralelos, alternativos e de estruturação;
- permite a especificação de elos *n:m*, que podem ser estabelecidos entre elementos inteiros ou sub-regiões temporais ou espaciais específicas para cada elemento, de acordo com seu tipo de mídia;
- possibilita o reuso de elementos simples (objetos de mídia) ou de elementos compostos (grupamentos);
- possibilita a definição de sincronização temporal através de grupamentos paralelos e seqüenciais ou de elos de sincronismo.

Deve-se ressaltar que a NCL apresenta algumas características adicionais, não encontrada nas demais linguagens estudadas. Entre elas, salienta-se a possibilidade de haver múltiplas formas de apresentação de um mesmo elemento utilizando o conceito de descritores. Alterações no comportamento dos elementos do documento durante sua exibição também se tornam viáveis a partir desse conceito.

Uma desvantagem apresentada pela NCL é a impossibilidade de especificação de disposições espaciais relativas entre os componentes do documento, como é permitido por Madeus.

5.2. Contribuições da Dissertação

A principal contribuição desta dissertação foi a definição de uma linguagem declarativa para especificação de documentos hipermídia com sincronização temporal e espacial.

A definição dessa linguagem apresenta várias vantagens. Dentre elas, podem ser salientados a autoria estruturada de documentos hipermídia compatíveis com o modelo NCM, a obtenção de um formato bem definido para intercâmbio de objetos NCM em uma representação NCL, e o reuso destes documentos NCL gerados a partir de objetos NCM em qualquer representação por outras aplicações que também obedeçam ao padrão. Outra vantagem a ser salientada, e talvez a mais importante, é a possibilidade de incorporar os elementos e atributos definidos na NCL a qualquer outra linguagem hipermídia baseada em XML, permitindo que as facilidades do modelo NCM sejam também adicionadas a essas linguagens.

A criação da linguagem declarativa NCL também ocasionou alguns refinamentos importantes no modelo conceitual NCM, dentre eles, a especificação mais detalhada do descritor.

A implementação dos módulos de conversão constitui outra contribuição desta dissertação. O sistema HyperProp, além de trabalhar normalmente com objetos NCM em uma representação Java, oferecerá módulos responsáveis por converter automaticamente esses objetos para uma representação em NCL e vice-versa.

Na atual versão do sistema HyperProp, o servidor se comunica com os clientes de duas formas. A primeira delas é via RMI, trocando objetos NCM em Java. A outra forma é via HTTP. Com a existência desses conversores bidirecionais, servidor e clientes do sistema HyperProp também poderão se comunicar via HTTP intercambiando objetos NCM em uma representação NCL.

Esses módulos conversores tornam possível a integração completa do sistema HyperProp com quaisquer outros sistemas compatíveis com o modelo NCM. Para isso, basta que os sistemas heterogêneos compatíveis com o NCM implementem módulos que façam a conversão entre objetos NCM em sua representação particular e objetos NCM em uma representação NCL, de acordo com a DTD criada para a linguagem.

Outras contribuições alcançadas foram a proposta de extensão da linguagem SMIL com as facilidades oferecidas pela linguagem NCL e a implementação de conversores que possibilitam a integração da linguagem SMIL com o sistema HyperProp. Esses conversores também estão integrados ao sistema HyperProp.

5.3. Trabalhos Futuros

Após o estudo e comparação de algumas linguagens hipermídia, percebeu-se que a NCL é bastante abrangente e oferece suporte adequado para a especificação de documentos hipermídia com sincronização temporal e espacial. No entanto, a exemplo da linguagem HyTime, não basta ter poder de expressão; é necessário poder de expressão com simplicidade de autoria.

Apesar de não ter sido uma questão de principal importância, durante o desenvolvimento da NCL a questão da facilidade de autoria também foi considerada. Um exemplo disso foi a introdução de elementos *par* e *seq* na linguagem. Como descrito durante a dissertação, toda a semântica envolvida na especificação desses elementos pode ser obtida na NCL utilizando elos para definir os relacionamentos. Porém, visando facilitar a autoria declarativa desses relacionamentos, os elementos *par* e *seq* foram herdados da linguagem SMIL e incorporados à NCL.

Dessa forma, uma sugestão para trabalho futuro, seria fazer uma análise bem detalhada das linguagens hipermídia existentes com o intuito de reconhecer facilidades de autoria expressas por elas, para possível melhoria em próximas versões da NCL. Na linguagem Madeus, por exemplo, são definidas relações de restrição espacial, que não estão presentes na NCL e que poderiam ser úteis na especificação da disposição espacial dos componentes do documento, facilitando bastante a sua autoria declarativa.

Outra sugestão para trabalho futuro seria modularizar a DTD NCL e torná-la extensível, permitindo, por exemplo, que novos tipos de objetos de mídia, novos modos de expressar relacionamentos e novas formas de definir funções de custo sejam incorporadas sem causar mudanças na especificação de documentos já existentes. Esses novos módulos poderiam ser

adicionados a linguagens hipermídia já existentes ou a novas linguagens, possibilitando a incorporação das principais características do modelo NCM a qualquer outra linguagem hipermídia.

Uma nova direção a ser seguida em trabalhos futuros envolve o estudo de linguagens de descrição de arquitetura de sistemas de software. Essas linguagens definem elementos estruturais que possuem algumas semelhanças com os elementos estruturais de um documento hipermídia. Dessa forma, poderia ser feita uma análise das características apresentadas pela NCL e pelas linguagens de descrição de arquiteturas com o intuito de identificar seus pontos em comum e suas principais diferenças. A partir dessa análise, seria possível verificar se funcionalidades particulares de cada uma poderiam ser úteis ou acrescentadas a outra, visando incrementar os recursos oferecidos por essas linguagens.

Apêndice A – DTD X-SMIL

```
<!-- Begin of DTD -->
<!-- ================== General entities ================== -->

<!-- System Attributes -->
<!ENTITY % system-attribute "
      system-bitrate               CDATA             #IMPLIED
      system-language              CDATA             #IMPLIED
      system-required              NMTOKEN           #IMPLIED
      system-screen-size           CDATA             #IMPLIED
      system-screen-depth          CDATA             #IMPLIED
      system-captions              (on|off)          #IMPLIED
      system-overdub-or-caption    (caption|overdub) #IMPLIED">


<!-- ==================== X-Smil Document ==================== -->

<!ELEMENT x-smil  (head?, (body | doc-body)?) >
<!ATTLIST x-smil
      id                ID          #IMPLIED>


<!--================= Head Element ======================-->

<!ELEMENT head    (meta*,(descriptors-set)?,(layout|switch)?,
                  (descriptors-set)?, meta*)?>
<!ATTLIST head
      id                ID          #IMPLIED>


<!--=================== Layout Element ====================-->

<!ELEMENT layout ANY>
<!ATTLIST layout
      id                ID          #IMPLIED
      type              CDATA       "text/smil-basic-layout">


<!--================= Root-layout Element ====================-->

<!ELEMENT root-layout EMPTY>
<!ATTLIST root-layout
      id                ID          #IMPLIED
      title             CDATA       #IMPLIED
      height            CDATA       #IMPLIED
      width             CDATA       #IMPLIED
      background-color  CDATA       #IMPLIED
      device            CDATA       #IMPLIED
      skip-content      (true|false) 'true'>
```

```
<!--================ Region Element ======================-->

<!ELEMENT region EMPTY>
<!ATTLIST region
      id                ID          #IMPLIED
      title             CDATA       #IMPLIED
      height            CDATA       #IMPLIED
      width             CDATA       #IMPLIED
      background-color  CDATA       #IMPLIED
      left              CDATA       "0"
      top               CDATA       "0"
      z-index           CDATA       "0"
      fit               (hidden|fill|meet|scroll|slice)     "hidden"
      root-layout       IDREF       #IMPLIED
      skip-content      (true|false) 'true'>


<!--=================== Meta Element ========================-->

<!ELEMENT meta EMPTY>
<!ATTLIST meta
      name              NMTOKEN     #REQUIRED
      content           CDATA       #REQUIRED
      skip-content      (true|false) 'true'>


<!-- ================ Descriptors-Set Element ================= -->

<!ELEMENT descriptors-set (descriptor)+ >
<!ATTLIST descriptors-set
      id                ID          #IMPLIED>


<!-- =========== Descriptor Element =============== -->

<!ELEMENT descriptor EMPTY >
<!ATTLIST descriptor
      id                ID          #REQUIRED
      title             CDATA       #IMPLIED
      player            CDATA       #IMPLIED
      dur               CDATA       #IMPLIED
      repetitions       CDATA       #IMPLIED
      left              CDATA       #IMPLIED
      top               CDATA       #IMPLIED
      width             CDATA       #IMPLIED
      height            CDATA       #IMPLIED
      bk-color          CDATA       #IMPLIED
      z-index           CDATA       #IMPLIED
      fit               (fill|hidden|meet|scroll|slice) #IMPLIED
      fill              (freeze|remove) #IMPLIED
      region            IDREF       #IMPLIED
      root-layout       IDREF       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      %system-attribute;>


<!-- =============== Body  Element =============== -->
```

```
<!ELEMENT body  ( (context|par|seq|audio|video|text|img|animation|
                  textstream|ref|switch|a|composition-anchor|
                  link)*, (presentation)? )>
<!ATTLIST body
      id                ID          #IMPLIED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      dur               CDATA       #IMPLIED
      repeat            CDATA       "1"
      region            IDREF       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED
      %system-attribute;>


<!-- ================ Doc-body Element ================ -->

<!ELEMENT doc-body  (context|par|seq|audio|video|text|img|animation|
                  textstream|ref|switch)>
<!ATTLIST doc-body
      id                ID          #IMPLIED>


<!-- ================ Context Element ================ -->

<!ELEMENT context ( (context|par|seq|audio|video|text|img|animation|
                  textstream|ref|switch|a|composition-anchor|
                  link)*, (presentation)? )>
<!ATTLIST context
      id                CDATA       #REQUIRED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED
      %system-attribute;>


<!-- ================ Context-Ref Element ================ -->

<!ELEMENT context-ref (composition-anchor)*>
<!ATTLIST context-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--================ Parallel Element ==================-->

<!ELEMENT par  ( (context|par|seq|audio|video|text|img|animation|
                  textstream|ref|switch|a|composition-anchor|
                  link)*, (presentation)? )>
<!ATTLIST par
      id                CDATA       #IMPLIED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
```

```
      copyright         CDATA       #IMPLIED
      endsync           CDATA       "last"
      dur               CDATA       #IMPLIED
      repeat            CDATA       "1"
      region            IDREF       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED
      %system-attribute;>


<!--================ Parallel-Ref Element ==================-->

<!ELEMENT par-ref  (composition-anchor)*>
<!ATTLIST par-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--============== The Sequential Element ==================-->

<!ELEMENT seq  ( (context|par|seq|audio|video|text|img|animation|
                  textstream|ref|switch|a|composition-anchor|
                  link)*, (presentation)? )>
<!ATTLIST seq
      id                CDATA       #IMPLIED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      dur               CDATA       #IMPLIED
      repeat            CDATA       "1"
      region            IDREF       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED
      %system-attribute;>


<!--============== The Sequential-Ref Element ==================-->

<!ELEMENT seq-ref  (composition-anchor)*>
<!ATTLIST seq-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--==================== The Switch Element ======================-->
<!-- In the head, a switch may contain only layout elements,
in the body or doc-body, only container elements. However, this
constraint cannot be expressed in the DTD (?), so
we allow both:
-->

<!ELEMENT switch (layout |(context|par|seq|audio|video|text|img|
                  animation|textstream|ref|switch|a)* )>
<!ATTLIST switch
      id                CDATA       #IMPLIED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
```

```
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>


<!--================= The Switch-Ref Element ================-->

<!ELEMENT switch-ref EMPTY>
<!ATTLIST switch-ref
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED>


<!--================ Presentation Element ==================-->

<!ELEMENT presentation  (object)*>
<!ATTLIST presentation
      id                CDATA       #IMPLIED>


<!--================ Object Element ==================-->

<!ELEMENT object    EMPTY>
<!ATTLIST object
      id                CDATA       #IMPLIED
      object-id         CDATA       #REQUIRED
      descriptor-list   IDREFS      #IMPLIED>


<!--=============== Media Object Elements ====================-->

<!ENTITY % mo-attributes "
      id                CDATA       #IMPLIED
      uid               CDATA       #IMPLIED
      title             CDATA       #IMPLIED
      abstract          CDATA       #IMPLIED
      author            CDATA       #IMPLIED
      copyright         CDATA       #IMPLIED
      region            IDREF       #IMPLIED
      alt               CDATA       #IMPLIED
      longdesc          CDATA       #IMPLIED
      src               CDATA       #IMPLIED
      type              CDATA       #IMPLIED
      dur               CDATA       #IMPLIED
      repeat            CDATA       '1'
      fill              (remove|freeze) 'remove'
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED
      %system-attribute;">


<!ENTITY % clip-attrs "
      clip-begin        CDATA       #IMPLIED
      clip-end          CDATA       #IMPLIED">


<!ELEMENT ref           (anchor|audio-anchor|img-anchor|
                         video-anchor|text-anchor)*>
<!ELEMENT audio         (anchor|audio-anchor)*>
<!ELEMENT img           (anchor|img-anchor)*>
<!ELEMENT video         (anchor|video-anchor)*>
```

```
<!ELEMENT text          (anchor|text-anchor)*>
<!ELEMENT textstream    (anchor|text-anchor)*>
<!ELEMENT animation     (anchor|video-anchor)*>



<!ATTLIST ref           %mo-attributes; %clip-attrs;>
<!ATTLIST audio         %mo-attributes; %clip-attrs;>
<!ATTLIST img           %mo-attributes;>
<!ATTLIST video         %mo-attributes; %clip-attrs;>
<!ATTLIST text          %mo-attributes;>
<!ATTLIST textstream    %mo-attributes; %clip-attrs;>
<!ATTLIST animation     %mo-attributes; %clip-attrs;>


<!--=============== Media Object Ref Elements ==============-->

<!ENTITY % mo-ref-attributes "
      id                CDATA       #REQUIRED
      uid               CDATA       #REQUIRED">


<!ELEMENT ref-ref       (audio-anchor|img-anchor|
                         video-anchor|text-anchor)*>
<!ELEMENT audio-ref           (audio-anchor)*>
<!ELEMENT img-ref             (img-anchor)*>
<!ELEMENT video-ref           (video-anchor)*>
<!ELEMENT text-ref            (text-anchor)*>
<!ELEMENT textstream-ref      (text-anchor)*>
<!ELEMENT animation-ref       (video-anchor)*>

<!ATTLIST ref-ref             %mo-ref-attributes;>
<!ATTLIST audio-ref           %mo-ref-attributes;>
<!ATTLIST img-ref             %mo-ref-attributes;>
<!ATTLIST video-ref           %mo-ref-attributes;>
<!ATTLIST text-ref            %mo-ref-attributes;>
<!ATTLIST textstream-ref      %mo-ref-attributes;>
<!ATTLIST animation-ref       %mo-ref-attributes;>


<!--=================== A Element ====================-->

<!ELEMENT a (par|seq|audio|video|text|img|animation|
             textstream|ref|switch|context)*>
<!ATTLIST a
      id                ID          #IMPLIED
      title             CDATA       #IMPLIED
      href              CDATA       #REQUIRED
      show              (replace|new|pause)  'replace'>


<!--=================== Anchor Element ================-->

<!ELEMENT anchor EMPTY>
<!ATTLIST anchor
      id                ID          #IMPLIED
      title             CDATA       #IMPLIED
      href              CDATA       #REQUIRED
      show              (replace|new|pause)  'replace'
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      coords            CDATA       #IMPLIED
```

```
      skip-content      (true|false) 'true'>


<!-- ============== Text-Anchor Element ============== -->

<!ELEMENT text-anchor EMPTY >
<!ATTLIST text-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      text              CDATA       #REQUIRED
      position          CDATA       #IMPLIED
      case-sensitive    CDATA       'false'>


<!-- ============== Audio-Anchor Element ============== -->

<!ELEMENT audio-anchor EMPTY >
<!ATTLIST audio-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED
      first-sample      CDATA       #IMPLIED
      last-sample       CDATA       #IMPLIED>


<!-- ============== Video-Anchor Element ============== -->

<!ELEMENT video-anchor EMPTY >
<!ATTLIST video-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      first-frame       CDATA       #IMPLIED
      last-frame        CDATA       #IMPLIED
      left              CDATA       '0'
      top               CDATA       '0'
      width             CDATA       'right'
      height            CDATA       'bottom'
      begin             CDATA       #IMPLIED
      end               CDATA       #IMPLIED>


<!-- ============== Img-Anchor Element ============== -->

<!ELEMENT img-anchor EMPTY >
<!ATTLIST img-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      left              CDATA       '0'
      top               CDATA       '0'
      width             CDATA       'right'
      height            CDATA       'bottom'>


<!-- ============== Composition-Anchor Element ============== -->

<!ELEMENT composition-anchor EMPTY >
<!ATTLIST composition-anchor
      id                CDATA       #REQUIRED
      title             CDATA       #IMPLIED
      node-list         CDATA       #REQUIRED>
```

```
<!-- ================== Link Element ================== -->

<!ELEMENT link (source-ep+, target-ep*, meeting-point)  >
<!ATTLIST link
      id                CDATA       #IMPLIED
      title             CDATA       #IMPLIED >
<!-- =========== Source-EP Element =========== -->

<!ELEMENT source-ep EMPTY >
<!ATTLIST source-ep
      id                CDATA       #REQUIRED
      node-list         CDATA       #IMPLIED
      anchor            CDATA       #IMPLIED
      event       (presentation|selection|attribution)  'selection'
      attr-name         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>


<!-- ========== Target-EP Element =========== -->

<!ELEMENT target-ep EMPTY >
<!ATTLIST target-ep
      id                CDATA       #REQUIRED
      node-list         CDATA       #IMPLIED
      anchor            CDATA       #IMPLIED
      event             (presentation|attribution)    'presentation'
      attr-name         CDATA       #IMPLIED
      descriptor-list   IDREFS      #IMPLIED>


<!-- =========== Meeting-Point Element ================ -->

<!ELEMENT meeting-point (condition, action?) >


<!-- =============== Condition Element ================ -->

<!ELEMENT condition ((simple-condition, delay-condition?) |
                     compound-condition) >


<!-- ============= Simple-Condition Element ============== -->

<!ELEMENT simple-condition EMPTY >
<!ATTLIST simple-condition
      id                CDATA       #REQUIRED
      previous          (true)      #IMPLIED
      previous-SEP      CDATA       #IMPLIED
      previous-function  (state|occurrence|repeat|attribute)  'state'
      previous-operator  (eq|dif|gt|get|lt|let)  'eq'
      previous-value    CDATA       'prepared'
      previous-SEP2     CDATA       #IMPLIED
      previous-type-attr (long | short | string) 'long'
      current           (true)      #IMPLIED
      current-SEP       CDATA       #IMPLIED
      current-function  (state|occurrence|repeat|attribute)  'state'
      current-operator  (eq|dif|gt|get|lt|let)  'eq'
      current-value     CDATA       'occurring'
      current-SEP2      CDATA       #IMPLIED
      current-type-attr (long|short|string)  'long' >
```

```
<!-- ========== Compound-Condition Element ========== -->

<!ELEMENT compound-condition (simple-condition+, delay-condition*,
                              expression-condition)>



<!-- ========= Expression-Condition Element ========== -->

<!ELEMENT expression-condition EMPTY>
<!ATTLIST expression-condition
      exp               CDATA       #REQUIRED>


<!-- ================ Delay-condition Element ================== -->

<!ELEMENT delay-condition EMPTY >
<!ATTLIST delay-condition
      id                CDATA       #REQUIRED
      t-begin           CDATA       #REQUIRED
      t-end             CDATA       #IMPLIED>


<!-- ================ Action Element ================ -->

<!ELEMENT action ((simple-action,delay-action?) | compound-action) >


<!-- ============= Simple-Action Element ============= -->

<!ELEMENT simple-action EMPTY >
<!ATTLIST simple-action
      id                CDATA       #REQUIRED
      TEP               CDATA       #REQUIRED
      action-name       (prepare|start|stop|pause|resume|
                        abort|relative-assign|absolute-assign|
                        enable|disable|activate)  'start'
      repetitions       CDATA       #IMPLIED
      attr-value        CDATA       #IMPLIED
      type-attr         (long|short|string)  #IMPLIED>


<!-- ============= Compound-Action Element ============ -->

<!ELEMENT compound-action (simple-action+, delay-action*,
   expression-action) >


<!-- ========== Expression-Action Element ========== -->

<!ELEMENT expression-action EMPTY>
<!ATTLIST expression-action
      exp               CDATA       #REQUIRED>

<!-- ================ Delay-action Element ================== -->

<!ELEMENT delay-action EMPTY >
<!ATTLIST delay-action
      id                CDATA       #REQUIRED
      t-min             CDATA       #IMPLIED
```

```
      t-opt             CDATA       #REQUIRED
      t-max             CDATA       #IMPLIED
      cost-function     CDATA       #IMPLIED>


<!-- End of DTD -->
```

Referências Bibliográficas

[Alle83] Allen, J. F. “Maintaining Knowledge about Temporal Intervals”. *CACM*, Vol. 26, No. 11. pp. 832-843, Novembro 1983.

[AnSo00a] Antonacci, M. J.; Soares, L. F. G. “Especificação NCL”. *Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio,* Rio de Janeiro, 2000.

[AnSo00b] Antonacci, M. J.; Soares, L. F. G. “Especificação X-SMIL”. *Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio,* Rio de Janeiro, 2000.

[Anto99] Antonacci, M. J.; “Conversores XML”. *Projeto Final de Programação, Departamento de Informática da PUC-Rio,* Rio de Janeiro, Junho 1999.

[BCGP92] Berners-Lee, T. J.; Cailliau, R.; Groff, J. F.; Pollermann, B. “World Wide Web: The information universe”. *Electronic Network Research, Application and Policy*. 1992.

[BFFGM97] Berners-Lee, T.; Fielding, R.; Frystick, H.; Gettys, J.; Mogul, J. “Hypertext Transport Protocol – HTTP/1.1”. *RFC2068*, MIT/LCS, UC Irvine, DEC. Janeiro 1997.

[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, Dezembro 1991.

[CSS298] “Cascading Style Sheets, level 2 (CSS2) Specification”. *W3C Recommendation*, Maio 1998. Disponível em: http://www.w3.org/TR/REC-CSS2

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

[HTML98] “HTML 4.0 Specification”. *W3C Recommendation*, Abril 1998. Disponível em: http://www.w3.org/TR/REC-html40

[HyTime97] “Hypermedia/Time-Based Structuring Language (HyTime)”. *ISO/IEC 10744*, Agosto 1997. Disponível em: http://www.ornl.gov/sgml/wg8/document/n1920/html/n1920.html

[JLRST98] Jourdan, M.; Layaïda, N.; Roisin, C.; Sabry-Ismail, L.; Tardif, L. “Madeus, an Authoring Environment for Interactive Multimedia Documents”, *Proceedings of the ACM Multimedia Conference 98*, pp. 267-272, Inglaterra, Setembro 1998.

[JRT98] Jourdan, M.; Roisin, C.; Tardif, L. “Constraints Techniques for Authoring Multimedia Documents” *ECAI 98 – Workshop on Constraints for Artistic Applications*, Bringhton, Agosto 1998.

[PeLi96] Pérez-Luque, M.J.; Little, T.D.C. “A Temporal Reference Framework for Multimedia Synchronization”. *IEEE Journal on Selected Areas in Communications (Special Issue: Synchronization Issues in Multimedia Communication*), Vol. 14, No. 1, pp. 36-51, Janeiro 1996.

[RFC1766] “Tags for the Identification of Languages”. *RFC1766*, Março 1995. Disponível em: ftp://ftp.isi.edu/in-notes/rfc1766.txt

[RMS98] Rodrigues, R.F.; Muchaluat-Saade, D.C.; Soares, L.F.G.; “Composite Nodes, Contextual Links and Graphical Structural Views on the WWW”, *Journal of the Brazilian Computer Society, special issue on World-Wide Web*, Vol. 5, No. 2, Novembro 1998.

[Rodr97] Rodrigues, R.F. “Formatação Temporal e Espacial no Sistema HyperProp” *Tese de Mestrado, Departamento de Informática da PUC-Rio*, Rio de Janeiro, Maio, 1997.

[RoDu98] Rousseau, F.; Duda. A. “Synchronized Multimedia for the WWW”. *Proc. Seven International World-Wide Web Conference*, Brisbane, Abril 1998. Também publicado em: *Computer Networks and ISDN Systems*, 30(1998), pp 417-429.

[RRMS99] Rodrigues, L.M.; Rodrigues, R.F.; Muchaluat-Saade, D.C.; Soares, L.F.G. “Improving SMIL Documents with NCM Facilities”, *Proceedings of the Multimedia Modeling Conference'99*, Canada, Outubro 1999. Também publicado em: *V Simpósio Brasileiro de Sistemas Multimídia e Hipermídia*, Goiânia, Junho 1999.

[SGML86] “Standard Generalized Markup Language (SGML)”. *ISO 8879*; Outubro 1986.

[SMIL98] “Synchronized Multimedia Integration Language (SMIL) 1.0 Specification”. *W3C Recommendation*, Junho 1998. Disponível em: http://www.w3.org/TR/REC-smil

[Soar00] Soares, L.F.G. “Modelo de Contextos Aninhados Versão 2.3”. *Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio*, Rio de Janeiro, 2000.

[TIMEH98] "Timed Interactive Multimedia Extensions for HTML (HTML+TIME)". *W3C Note*, Setembro 1998. Disponível em: http://www.w3.org/TR/NOTE-HTMLplusTIME

[XHTML99] "XHTML 1.0: The Extensible HyperText Markup Language". *W3C Proposed Recommendation*, Dezembro 1999. Disponível em: http://www.w3.org/TR/xhtml1

[XML98] "Extensible Markup Language (XML) 1.0". *W3C Recommendation*, Fevereiro 1998. Disponível em: http://www.w3.org/TR/REC-xml

[XMLPv2] "Oracle's XML Parser for Java v2". *OTN Technologies*. Disponível em: http://technet.oracle.com/tech/xml/parser_java2/