PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Heron Vilela de Oliveira e Silva

X-SMIL: Aumentando Reuso e Expressividade em Linguagens de Autoria Hipermídia

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA

Programa de Pós-Graduação em Informática

Rio de Janeiro
Abril de 2005

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Heron Vilela de Oliveira e Silva

X-SMIL: Aumentando Reuso e Expressividade em Linguagens de Autoria Hipermídia

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientadore: Luiz Fernando Gomes Soares
Co-orientador: Rogério Ferreira Rodrigues

Rio de Janeiro, abril de 2005

Heron Vilela de Oliveira e Silva

X-SMIL: Aumentando Reuso e Expressividade em Linguagens de Autoria Hipermídia

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Luiz Fernando Gomes Soares
Orientador
Departamento de Informática - PUC-Rio

Rogério Ferreira Rodrigues
Co-orientador
Departamento de Informática - PUC-Rio

Marco Antonio Casanova
Departamento de Informática - PUC-Rio

Renato Fontoura de Gusmão Cerqueira
Departamento de Informática - PUC-Rio

José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 4 de abril de 2005

Heron Vilela de Oliveira e Silva

Formou-se em Engenheira de Computação pela PUC-Rio em 2002. Atualmente, integra o grupo de pesquisadores do Laboratório TeleMídia da PUC-Rio, desenvolvendo pesquisa na área de Sistemas Hipermídia.

Ficha Catalográfica

Silva, Heron Vilela de Oliveira e

X-SMIL: aumentando reuso e expressividade em linguagens de autoria hipermídia / Heron Vilela de Oliveira e Silva ; orientador: Luiz Fernando Gomes Soares. – Rio de Janeiro : PUC-Rio, Departamento de Informática, 2005.

210 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática .

Inclui referências bibliográficas

1. Informática – Teses. 2. Sistemas hipermídia. 3. Autoria. 4. Linguagens declarativas. 5 Relações. 6. SMIL. 7. Sincronização. 8. Conectores. 9. Templates. 10. NCL. I. Soares, Luiz Fernando Gomes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Este trabalho é dedicado:

À Deus, por ser.

Agradecimentos

À minha família, pelo amor incondicional.

Aos meus orientadores, pela amizade, compreensão e apoio constante.

Aos amigos, por estarem, mesmo quando ausentes, sempre presentes.

Ao espírito TeleMídia.

Ao DI.

À PUC, CAPES, CNPq e FUNTTEL.

Resumo

Silva, Heron Vilela de Oliveira. **X-SMIL: Aumentando Reuso e Expressividade em Linguagens de Autoria Hipermídia.** Rio de Janeiro, 2005. xxxp. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho está inserido no contexto de ambientes de autoria e execução hipermídia, sendo as linguagens declarativas para autoria de documentos o seu foco principal. Tendo-se como objetivo aumentar a expressividade e o reuso na especificação de documentos hipermídia, este trabalho introduz as linguagens X-SMIL e NCL - Nested Context Language - versão 2.1. Utilizando-se o conceito de templates, X-SMIL permite a definição de novas semânticas para composições SMIL, além dos tradicionais elementos seq, par e excl. Templates, em X-SMIL, são especificados em um perfil de XTemplate, que estende a idéia original da linguagem XTemplate de NCL. Com base nas novas facilidades para definição de templates, esse perfil foi usado para especificar a linguagem NCL 2.1. X-SMIL também permite a especificação de conectores hipermídia, tratando relações hipermídia como entidades de primeira classe - funcionalidade incorporada em X-SMIL pelo uso do módulo XConnector de NCL. Outro objetivo deste trabalho é o de apresentar um framework para o processamento de documentos XML. Utilizando-se esse framework, diversos compiladores foram implementados, o que possibilitou, entre outras funcionalidades, a conversão de documentos NCL em especificações SMIL ou X-SMIL e vice-versa.

Palavras-chave

sistemas hipermídia; autoria; linguagens declarativas; relações; elos; sincronização; conectores; templates; NCL; SMIL; XML; compilador

Abstract

Silva, Heron Vilela de Oliveira. **X-SMIL: Improving Reuse and Expressiveness in Hypermedia Authoring Languages**. Rio de Janeiro, 2005. XXXp. Master Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work is related to hypermedia authoring and execution environments, and its main focus is declarative document authoring. Aiming at improving the expressiveness and reuse in the specification of hypermedia documents, this work introduces the hypermedia authoring languages X-SMIL and NCL - Nested Context Language - version 2.1. Exploiting the concept of templates, X-SMIL allows the definition of new semantics for SMIL compositions, besides its usual seq, par and excl elements. X-SMIL templates are specified using an XTemplate profile, which extends the original idea of the NCL XTemplate language. Bringing new facilities for template definitions, this new profile is used to further improve the NCL language. X-SMIL also offers support for handling hypermedia relations as first-class entities, through the use of hypermedia connectors - brought to X-SMIL via the NCL XConnector module. Another important goal of this work is to present a framework to facilitate the development of XML documents parsing and processing tools. Based on this framework, several compilers were implemented, permitting, among other features, the conversion of NCL documents into SMIL or X-SMIL specifications and vice-versa.

Palavras-chave

hypermedia systems; authoring; declarative languages; relations; links; synchronization; connectors; templates; NCL; SMIL; XML; compiler

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

Em sua origem, a WWW - *World-Wide Web* (Berners-Lee, 1994) foi concebida como uma aplicação de hipertexto, visando apresentar informações científicas com referências cruzadas e permitindo a pesquisa automática de textos. Para especificação de documentos nessa aplicação, foi definida a primeira versão da linguagem HTML, que veio a se tornar um padrão do *W3C Consortium*[1]. Universalmente adotada nos dias atuais, HTML (W3C, 1999a) é a principal linguagem para a autoria de documentos na rede mundial de computadores.

O fenômeno de difusão da Internet ultrapassou todas as previsões e o paradigma de hipertexto, utilizado pela linguagem HTML em sua forma original, evoluiu para o paradigma hipermídia. Proporcionando a incorporação de novos formatos de mídia, o paradigma hipermídia traz também novos requisitos, como a definição de sincronismo temporal e espacial entre componentes de um documento. Tais requisitos vêm motivando uma extensa pesquisa na área de sistemas hipermídia, acarretando a definição de novos padrões, tanto para a WWW como para outros sistemas.

Contextualizando o trabalho, este capítulo apresenta, inicialmente, uma visão geral de sistemas hipermídia e conceitos relacionados. Baseados nesses conceitos, serão apontados a motivação e os objetivos desta dissertação. Finalmente, a organização do restante do texto é apresentada.

1.1.
Motivação

Um sistema hipermídia possui, tipicamente, os seguintes subsistemas[2]: o de autoria (edição), o de armazenamento e o de execução (exibição). Durante o

[1] http://www.w3c.org/

[2] Os *subsistemas* também podem ser chamados de *ambientes* ou *módulos* do sistema hipermídia.

processo de autoria, o sistema deve oferecer ferramentas que possibilitem especificar um documento hipermídia de acordo com as concepções do autor. Essa especificação pode, então, tanto ser armazenada quanto entregue para exibição. A Figura 1:1 ilustra os subsistemas de um sistema hipermídia.

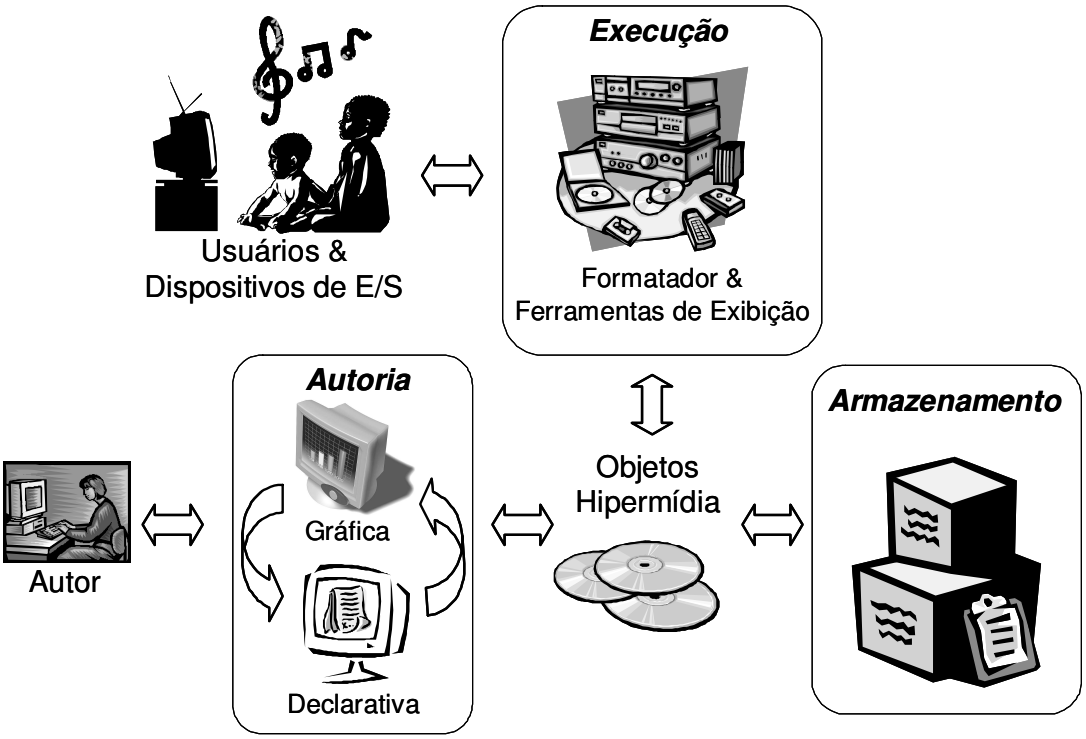


Figura 1:1. Subsistemas de um sistema hipermídia

O subsistema de autoria deve fornecer os recursos necessários para que um autor (ou autores) especifique um documento hipermídia. Nesse subsistema, é desejável a presença de ferramentas de autoria seguindo tanto o paradigma gráfico (baseado em interfaces gráficas) quanto o paradigma declarativo (baseado em linguagens textuais) - como apresentado em (Coelho, 2004).

O subsistema de armazenamento é responsável pelo armazenamento, recuperação e, em certos casos, adaptação de documentos[3]. Em geral, os dados armazenados são classificados em *descrição dos documentos* (contendo informações a respeito da estrutura do documento, os relacionamentos entre seus componentes e especificações de apresentação) e *conteúdo dos objetos de mídia* (arquivos de texto, imagem, áudio, vídeo etc. referenciados pela descrição dos documentos). Um dos formatos possíveis para o armazenamento das descrições de documentos é a especificação em uma linguagem declarativa.

Finalmente, o subsistema de execução (ou de apresentação) contém os mecanismos para interagir com os usuários e controlar a exibição dos objetos de mídia, de forma a respeitar as especificações do documento. Entre os elementos que compõem o ambiente de execução, podem ser destacados: as ferramentas de exibição e o formatador hipermídia. As ferramentas de exibição são compostas pelos recursos de hardware ou software responsáveis por tratar os dispositivos de entrada e saída (*Dispositivos de E/S* na Figura 1:1), controlando a exibição do conteúdo dos objetos e a interação com o(s) usuário(s). O formatador hipermídia é o nome dado ao elemento que reúne as funções para o controle da apresentação. Para realizar suas tarefas, o formatador deve receber a especificação de um documento que, como já mencionado, pode ser baseada em uma linguagem declarativa.

A adaptação de documentos é uma funcionalidade desejável em sistemas hipermídia. Seja no subsistema de armazenamento ou no de execução, a adaptação de documentos pode atuar sobre a estrutura, apresentação ou o próprio conteúdo dos objetos. Essa adaptação pode ser direcionada pelas características da plataforma de armazenamento (capacidade de processamento disponível, banda passante disponível para o acesso aos recursos gravados em memória secundária, número de clientes conectados, banda passante para transmissão dos dados no sistema de comunicação etc.) e, também, pelas informações provenientes do ambiente de execução (preferências e conhecimentos do usuário, recursos de hardware disponíveis na plataforma de exibição, localização etc.). Todas essas informações fazem parte do contexto de apresentação do documento (Boll & Klas, 1999; Brusilovsky, 1996; Dey et al., 2001; Schilit et al., 1994; Villard et al., 2000). Dessa forma, é desejável que linguagens declarativas hipermídia ofereçam suporte à criação de documentos adaptáveis.

Esta dissertação tem como foco linguagens para autoria declarativa de documentos hipermídia. A expressividade dessas linguagens está diretamente relacionada com o modelo hipermídia no qual elas se baseiam. Analisando os diversos modelos hipermídia propostos na literatura, podem ser observadas duas entidades básicas: nós (representando componentes de conteúdo de um documento) e elos (representando relacionamentos entre nós) (Muchaluat-Saade,

[3] O conceito de adaptação será detalhado posteriormente.

2003). Freqüentemente, uma terceira entidade também é encontrada, chamada nó de composição, ou simplesmente composição. Assim como elos, nós de composição são utilizados para representar relacionamentos entre componentes de um documento. Visando um aumento de expressividade (dentre outros benefícios), modelos mais complexos possuem outras entidades. A complexidade e o poder de expressão dos modelos são, obviamente, herdados por suas linguagens.

Com o surgimento do padrão XML (W3C, 2000b), linguagens declarativas, nele baseadas (aplicações XML), foram definidas para diferentes modelos hipermídia. O modelo hipermídia CMIF (van-Rossum et al., 1993), por exemplo, serviu como base para a linguagem *Synchronized Multimedia Integration Language* - SMIL (W3C, 2001b), enquanto o modelo *Nested Context Model* - NCM (Soares et al., 2003) originou a linguagem *Nested Context Language* - NCL (Muchaluat-Saade et al., 2003; Muchaluat-Saade, 2003).

Comparando as diversas linguagens hipermídia existentes, podem ser observadas características e funcionalidades presentes somente em algumas delas. Isso ocorre com NCL, por exemplo, que possui o conceito de conectores hipermídia e de *templates* de composição - conceitos não encontrados em outras linguagens hipermídia. Conectores hipermídia (definidos pela linguagem XConnector de NCL) são entidades que expressam a semântica de uma relação, podendo ser reutilizados para criação de diferentes relacionamentos (elos). *Templates* de composição (definidos pela linguagem XTemplate de NCL[4]), por sua vez, especificam a semântica de composições (semântica temporal paralela, semântica temporal seqüencial etc.), podendo ser reaproveitados por diferentes composições, que herdam as especificações definidas no *template.*

Diferente de NCL, composições SMIL possuem uma semântica temporal pré-definida (seqüencial, paralela etc.). Em diversos casos, as composições com semântica temporal em SMIL oferecem uma grande facilidade para autoria de documentos. Entretanto, a facilidade de autoria oferecida está restrita ao uso das composições previamente definidas pela linguagem. Além das composições, um número limitado de eventos podem ser utilizados para definição de elos

[4] As linguagens XConnector e XTemplate fazem parte do conjunto de especificações da linguagem NCL.

(relacionamentos) entre nós. Porém, na especificação de relacionamentos complexos, é necessária a combinação de diversos eventos e/ou composições SMIL. Essa abordagem dificulta o processo de autoria e, principalmente, o reuso.

Visando aumentar o reuso e facilitar ainda mais o processo de autoria, é interessante incorporar o conceito de conectores e *templates* hipermídia à linguagem SMIL. Conectores podem definir relações simples ou complexas, e permitem que qualquer relacionamento SMIL (tanto via composições, como via eventos) seja declarado da mesma maneira: através do reuso de conectores. O uso de *templates* de composição possibilita a definição de diferentes semânticas para composições, criando, dessa forma, um mecanismo para tornar SMIL extensível em relação às composições que oferece.

Este trabalho engloba não apenas a expressividade e facilidade de reuso em linguagens hipermídia (notadamente: NCL e SMIL), mas também a conversão (interoperabilidade) entre os diversos formatos abstratos por elas gerados. Assim, por exemplo, um documento SMIL pode ser editado em um ambiente de autoria utilizando-se de um outro formato de dados (como objetos Java de uma implementação do modelo NCM). De forma análoga, para que esse documento seja enviado para o subsistema de armazenamento, pode ser necessária sua conversão para um outro formato. Finalmente, para sua apresentação, pode ser ainda exigida uma conversão para o formato de dados do formatador hipermídia que irá exibi-lo. Essas conversões são realizadas por compiladores de documentos de linguagens hipermídia, como será apresentado.

1.2. Objetivos

Os dois principais objetivos desta dissertação são: (1) aumentar o reuso e a expressividade da linguagem SMIL, especificando a linguagem X-SMIL; e (2) definir *frameworks* de compiladores e implementar suas instâncias (compiladores específicos) para processar documentos nessas linguagens. Como decorrência desses objetivos, um terceiro surge naturalmente para viabilizá-los: (3) o refinamento da linguagem NCL, introduzindo a versão 2.1 dessa linguagem - que estende sua versão anterior (2.0).

Assim, inicialmente, o refinamento à linguagem NCL será proposto. Entre as principais melhoras destacam-se a definição de um módulo[5] NCL para a especificação de funções de custo para as durações de objetos de mídia (Rodrigues, 2003) e a definição de um módulo para a especificação de regras de apresentação, buscando auxiliar a autoria de documentos adaptativos. A maior contribuição nesse refinamento, no entanto, é a redefinição da linguagem XTemplate de NCL, utilizada para a especificação de *templates* de composição.

A proposta para a extensão de SMIL será desenvolvida introduzindo, nessa linguagem, o conceito de *templates* de composição e o conceito de conectores. Dessa forma, uma nova linguagem, chamada X-SMIL, é especificada. Em X-SMIL serão permitidos: a definição e reuso de especificações semânticas para composições - através de *templates* de composição; e a definição e reuso de especificações de relações - através de conectores (que podem ser entendidos, nesse contexto, como *templates* para elos).

Na definição da linguagem X-SMIL, adotou-se uma estratégia para incorporação dos conceitos mais importantes de NCL. Assim, a linguagem X-SMIL é definida por meio de duas extensões à linguagem SMIL: o perfil[6] SMIL + XTemplate (ou XT-SMIL), que introduz as diversas vantagens do uso de *templates* de composição em SMIL; e o perfil SMIL + XConnector (ou XC-SMIL), que incorpora o conceito de conectores (e a definição de elos a partir de referências a conectores) à linguagem SMIL.

Apesar de a linguagem XTemplate de NCL permitir relacionamentos outros que não apenas aqueles definidos por conectores, a versão XTemplate de NCL 2.0 (ou, simplesmente, XTemplate 2.0) era limitada a esses relacionamentos. Como, a princípio, no perfil XT-SMIL não seria viável a utilização de *templates* de composição sem o uso de conectores, foi preciso redefinir a linguagem XTemplate 2.0, dando origem à linguagem XTemplate 2.1. Essa redefinição consiste na sua estruturação em perfis, dependentes das relações para as quais provêem suporte. Um desses perfis, independente de conectores e que possibilita a

[5] O conceito de módulos de linguagens será apresentado no Capítulo 2.

[6] Perfis de linguagem reúnem um subconjunto dos módulos oferecidos pela linguagem, definindo assim um subconjunto de funcionalidades apropriadas para a construção de uma determinada classe de documentos.

definição de relações de inclusão, é o utilizado em XT-SMIL. Relações de inclusão[7] constituem a forma mais simples de se introduzir o conceito de *templates* em SMIL.

A definição do perfil XC-SMIL é simples, pois XConnector é um módulo independente em NCL. Finalmente, é definida a linguagem X-SMIL, resultante da combinação de XT-SMIL e XC-SMIL. Em X-SMIL, o perfil de XTemplate 2.1 utilizado pode definir relações através de conectores ou relações de inclusão. Esse perfil de XTemplate é, exatamente, o definido pela linguagem NCL 2.1.

Como mencionado, um dos objetivos deste trabalho é a análise da interoperabilidade entre as linguagens NCL, SMIL e X-SMIL - através de compiladores de documentos de cada uma dessas linguagens para documentos nas outras duas. A compilação de documentos nessas três linguagens para uma implementação do modelo NCM em Java e para o modelo de execução do Formatador HyperProp (Rodrigues, 2003) também será abordada, com o objetivo de visualização gráfica, edição e execução (exibição) desses documentos no sistema HyperProp (Soares et al., 2000).

Como cada grupo de compiladores (de NCL, de SMIL e de X-SMIL) analisa documentos de uma mesma linguagem de origem, várias funcionalidades podem ser reaproveitadas e implementadas uma única vez. A identificação dessas funcionalidades constitui o primeiro passo de implementação, dando origem à definição de *frameworks* para compiladores, como o *framework* para compiladores NCL.

Além das características comuns aos compiladores de uma mesma linguagem, observou-se, em outro nível de abstração, aspectos comuns que podem ser reutilizados em processadores de qualquer linguagem baseada em XML. A partir dessas observações, uma estrutura em dois níveis para especificação de *frameworks* para implementação de compiladores é proposta. Em um primeiro nível, existe o *framework genérico de processamento* (ou *meta-framework*), cujo objetivo é facilitar o desenvolvimento de *frameworks* de compiladores. O *meta-framework* define, sintática e semanticamente, os métodos de *frameworks* de compiladores e a estruturação das classes desses *frameworks* (que são gerados

[7] Relações de inclusão permitem especificar as composições nas quais determinados objetos devem estar contidos.

automaticamente, a partir das especificações de linguagens baseadas em XML). Dessa forma, o *framework* para compiladores NCL, o *framework* para compiladores SMIL e o *framework* para compiladores X-SMIL representam instâncias do *meta-framework*.

Finalmente, é importante destacar que, juntamente com a descrição desses compiladores, será apresentada a implementação do processador de *template* para os perfis da linguagem XTemplate. Esse processador também é implementado como instância do *meta-framework* de compiladores.

1.3. Organização da Dissertação

Esta dissertação está estruturada da seguinte forma. O Capítulo 2 apresenta linguagens para especificação de documentos hipermídia, dando destaque para as linguagens NCL e SMIL. O Capítulo 3 discute as principais funcionalidades introduzidas e refinadas na linguagem NCL. Nesse capítulo, são detalhados os novos módulos de NCL: os novos módulos para *templates* de composição (ou seja, a definição e estruturação da linguagem XTemplate 2.1), o módulo para especificação de regras de apresentação, o módulo para especificação de funções de custo e os refinamentos da linguagem XConnector. O Capítulo 4 contempla a linguagem X-SMIL, detalhando os perfis que compõem essa linguagem: XT-SMIL (SMIL + XTemplate) e XC-SMIL (SMIL + XConnector). O Capítulo 5 trata dos compiladores para linguagens XML, apresentando o *meta-framework* de compiladores, assim como os *frameworks* para compiladores instanciados a partir do *meta-framework*. O Capítulo 5 também descreve os compiladores instanciados a partir dos *frameworks* para compiladores. O Capítulo 6 traz comparações com os principais trabalhos relacionados da área e o Capítulo 7 tece as considerações finais da dissertação, salientando as contribuições do trabalho e possíveis pesquisas futuras.

2
Linguagens para Descrição de Documentos Hipermídia

Linguagens de programação podem ser classificadas de modos variados. Uma classificação possível distingue as linguagens entre procedurais (imperativas) e declarativas. Linguagens procedurais exigem que o programador especifique, em detalhes, os passos que o programa deve executar para realizar a tarefa projetada (são escritas muitas linhas de código para indicar os passos exatos a fim de se obter um resultado). Em linguagens declarativas, por outro lado, o programador provê uma definição da tarefa a ser realizada, não estando preocupado com os detalhes de como o computador usará essa definição (é oferecida uma descrição exata do objetivo). Em outras palavras, a diferença entre linguagens procedurais e declarativas é que, na primeira, se especifica como obter a resposta, enquanto, na segunda, se especificam as condições que devem ser satisfeitas para se obter a resposta. Programação declarativa envolve *o que* deve ser computado, mas não necessariamente *como* será computado.

Embora linguagens procedurais possam ser utilizadas na autoria de documentos hipermídia (FLEXTV, 2004a), linguagens declarativas oferecem um nível mais elevado de abstração para a programação, por evitar que o programador (ou autor, no contexto de autoria de documentos) escreva muitos dos detalhes de execução que podem ser inferidos pelo interpretador. Esse interpretador (ou executor) é o elemento responsável por aplicar um algoritmo pré-definido às especificações feitas em uma linguagem declarativa para produzir o resultado almejado (como foi visto, esse elemento, em sistemas hipermídia, é denominado formatador). Assim, um programador declarativo experiente é qualificado para descrever cuidadosamente os resultados esperados, entretanto, mantém-se ignorante de como o código interno é executado para se obter o resultado. Obviamente, a linguagem declarativa deve prover os recursos necessários para que o programador/autor seja capaz de especificar um programa/documento de acordo com sua concepção[8].

[8] Uma comparação entre autoria declarativa e procedural, apontando suas vantagens e desvantagens, pode ser consultada em (FLEXTV, 2004a).

Por apresentarem um nível de abstração mais elevado para a programação, esta dissertação irá tratar apenas de linguagens declarativas para autoria de documentos. Em especial, serão abordadas linguagens de marcação declarativas definidas utilizando o padrão XML.

Este capítulo está estruturado da seguinte maneira. Inicialmente discorre-se sobre a meta-linguagem SGML, padrão ISO (*International Organization for Standardization*[9]) que estabeleceu o conceito de linguagens de marcação para autoria de documentos eletrônicos. Em seguida, é apresentada a meta-linguagem XML, um perfil SGML[10] a partir do qual várias linguagens de marcação foram criadas e vêm sendo utilizadas nas mais diversas áreas. Finalmente, são descritas linguagens declarativas baseadas em XML voltadas para o domínio de autoria de documentos hipermídia, notadamente: XHTML, SMIL e NCL.

2.1. SGML e XML

O padrão *Standard Generalized Markup Language* - SGML (ISO, 1986) permite a definição de documentos em função do domínio de aplicação a que se destinam. Marcações SGML permitem especificar a estruturação desses documentos, independente de como os mesmos devam ser apresentados. A formatação (ou seja, características de apresentação) de um documento SGML deve ser definida externamente ao documento, utilizando, por exemplo, tecnologias de folhas de estilos - como CSS (W3C, 1998a).

Para a definição e interpretação de um documento SGML, é preciso especificar uma gramática formal. Uma gramática SGML define como as marcações devem ser interpretadas, quais as regras que restringem o uso de cada marcação nos diferentes contextos do documento e, quando for relevante, a ordem dessas marcações no documento.

A Figura 2:1 ilustra uma aplicação SGML, composta por *declarações SGML*, uma DTD (*Document Type Definition*) e um *interpretador*. De maneira simplificada, declarações SGML definem um conjunto de regras de sintaxe para uma linguagem, enquanto a DTD especifica sua semântica. Dessa forma, uma

[9] http://www.iso.org/

[10] Como será apresentado, o conceito de perfil SMGL é distinto do conceito de perfil de linguagem apresentado anteriormente.

aplicação SGML[11] tem definida a gramática a ser seguida por documentos de uma linguagem, sendo possível aos interpretadores dessa aplicação, além de verificar a sintaxe desses documentos, também interpretar sua semântica, controlando, inclusive, suas exibições.

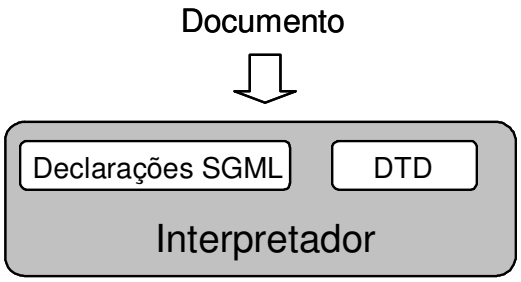


Figura 2:1 - Aplicação SGML.

SGML é suficientemente formal para possibilitar a validação de documentos; tem estrutura suficiente para permitir a especificação e o manuseamento de documentos complexos; e possui mecanismos de extensão capazes de suportar a gestão de grandes repositórios de informação. Entretanto, dada a sua generalidade, a utilização direta de SGML mostrou-se demasiadamente complexa. Visando solucionar parte desse problema, foi definido o padrão XML: um perfil SGML.

Extensible Markup Language - XML (W3C, 2004) restringe alguns dos pontos de flexibilização de SGML, facilitando o desenvolvimento de novas aplicações (linguagens de marcação), assim como o processamento e o intercâmbio de documentos declarados nessas linguagens. XML vem se mostrando um padrão *de fato* para criação de linguagens de marcação, tanto para domínios hipermídia quanto para outros domínios.

De maneira simplificada, XML é menos genérico que o padrão SGML, pois predefine as *declarações SGML* desse padrão (ver Figura 2:1). Assim, é necessária apenas a especificação de DTDs na criação de novas linguagens (aplicações XML). Devido a suas características, XML foi amplamente utilizado, levando à identificação não somente de suas diversas vantagens, mas, também, de algumas de suas limitações. A dificuldade no reuso das definições presentes nas DTDs, por exemplo, foi um dos fatores que impulsionou o desenvolvimento de

[11] Uma aplicação SGML é constituída por um perfil SGML e por uma DTD específica. Ou seja, um interpretador que possui somente *declarações SGML* constitui um perfil (*profile*) SGML.

novas tecnologias relacionadas a XML (W3C, 2001d; W3C, 1999b). Essas novas tecnologias oferecem uma funcionalidade de extrema importância quando se abordam linguagens de marcação baseadas em XML: a criação de linguagens seguindo uma abordagem modular. Módulos agrupam, de forma coerente, elementos e atributos XML[12] que possuam alguma relação semântica entre si.

Diversas são as vantagens da definição de uma linguagem de forma modular. Um primeiro benefício é a possibilidade de criação de perfis de linguagem. Perfis reúnem um subconjunto dos módulos oferecidos pela linguagem, definindo, assim, um subconjunto de funcionalidades apropriadas para a construção de uma determinada classe de documentos. Outra vantagem da estruturação da linguagem em módulos é a facilidade de reutilização. Linguagens podem ser construídas definindo novos módulos e reusando módulos oferecidos por outras linguagens. Mais ainda, é possível a criação de novas linguagens XML através da simples combinação de diversos módulos já existentes em outras linguagens, sem a necessidade de definição de novos módulos. Essas várias vantagens possibilitam que cada linguagem, ou perfil da linguagem, atenda aos requisitos de uma determinada aplicação.

Outra vantagem da abordagem modular é a estruturação da linguagem, que pode ser extremamente útil em linguagens complexas (e com um grande número de elementos e atributos). Assim, módulos podem agrupar os elementos e atributos que possuam semântica relacionada, facilitando, entre outras coisas, a manutenção da linguagem. Como o número de módulos pode ser grande, algumas linguagens definem um segundo nível de estruturação, agrupando seus módulos em *áreas funcionais*.

2.2. HTML e XHTML

HyperText Markup Language - HTML - é a principal linguagem utilizada para autoria de documentos na WWW. Tendo sua primeira versão publicada em 1992 (como uma aplicação SGML), HTML (W3C, 1999a) é, atualmente, uma linguagem declarativa de autoria padronizada pelo W3C.

[12] Elementos e atributos XML serão exemplificados nas próximas subseções.

O modelo hipermídia que fundamenta a linguagem HTML é bastante simples, possibilitando a criação de uma linguagem de fácil autoria (apesar de limitar tanto seu poder de expressão quanto seus recursos para reuso). Páginas HTML representam os *nós* do modelo, enquanto *elos* definem os relacionamentos entre essas páginas. Elos HTML são sempre de referência (do tipo *go-to*) e possuem duas características principais: especificam um relacionamento entre uma única origem e um único destino; e são sempre definidos como parte do conteúdo dos nós (mais especificamente, como parte do conteúdo dos nós de origem do elo).

Essas características impedem a identificação dos elos que fazem referência a uma determinada página; não permitem a separação entre os dados sendo referenciados e as referências propriamente ditas (dificultando a manutenção dos dados e dos elos); não possibilitam a reutilização de nós sem a herança obrigatória dos relacionamentos (por exemplo, não é possível reusar uma página HTML sem herdar todos os seus elos); e impedem a definição de elos em páginas nas quais o autor não possua direitos de escrita.

Apesar das limitações de HTML, sua simplicidade proporcionou uma ampla utilização dessa linguagem. Assim, com a maturidade e simplicidade de HTML e o surgimento do padrão XML, observou-se a possibilidade de definir uma linguagem baseada em XML oferecendo todas as funcionalidades de HTML: a linguagem XHTML (W3C, 2000c). Dessa forma, XHTML possui todos os benefícios de XML (facilidade de intercâmbio de documentos entre aplicações; facilidade na implementação de processadores de documentos nessa linguagem; variedade de bibliotecas de software que oferecem suporte a XML; variedade de tecnologias aplicáveis a XML etc.) agregados à maturidade de HTML.

2.3. SMIL

Synchronized Multimedia Integration Language - SMIL (W3C, 2001b) é uma linguagem de marcação declarativa modular (estruturada em 10 áreas funcionais), derivada de XML, para autoria de documentos hipermídia. Tendo como foco relações de sincronismo temporal, SMIL é um padrão W3C que visa incorporar esse requisito à WWW (ver Capítulo 1), de forma a superar as

limitações de HTML. Diferente de HTML, cujos elos somente possibilitam especificação de relacionamentos de referência entre páginas, SMIL também permite a definição de relacionamentos de sincronismo temporal. Esses relacionamentos são expressos por meio de eventos ou composições.

Composições SMIL contêm um conjunto de nós (objetos de mídia ou outros nós de composição) e possuem semântica temporal: a composição *par* determina que seus nós internos devem ser exibidos em paralelo; a composição *seq* determina a exibição de seus nós em seqüência; e a composição *excl* (exclusiva) especifica que seus nós não podem ser exibidos simultaneamente. Além de composições, relacionamentos em SMIL podem ser definidos através de eventos, como *begin* (início de apresentação), *end* (término de apresentação) e *click* (clique do mouse). A Figura 2:2 apresenta um documento SMIL simplificado. A composição *seq* definida nas linhas 08-18 especifica a exibição, em seqüência, de suas composições internas. A composição *par* (linhas 09-13) determina a exibição em paralelo de um vídeo, um áudio e uma imagem: *video1*, *audio1* e *img1*. O elo da linha 12, definido juntamente com a imagem[13], especifica que a exibição dessa imagem deve terminar (atributo *end*) ao final da exibição de *audio1* (valor do atributo *end* especificado como o evento *audio1.end*). A outra composição *seq* (linhas 14-17) determina que dois áudios sejam exibidos em seqüência. Pela semântica da composição *seq* mais externa, esses dois últimos áudios serão exibidos após o término da composição paralela.

```
01. <smil>
02.  <head>
03.   <layout>
04.    ...
05.   </layout>
06.  </head>
07.  <body id=" ">
08.   <seq>
09.     <par>
10.       <video id="video1" />
11.       <audio id="audio1" />
12.       <img id="img1" end="audio1.end" />
13.     </par>
14.     <seq>
15.       <audio id="audio2" />
```

[13] O nó imagem é definido por meio do elemento XML *img*, enquanto o elo é definido utilizando o atributo XML *end* desse elemento.

```
16.       <audio id="audio3" />
17.     </seq>
18.    </seq>
19.  </body>
20.</smil>
```

Figura 2:2 - Exemplo de um documento SMIL.

Visando incorporar à HTML as facilidades para autoria de documentos com sincronização temporal presentes em SMIL, foi proposta, em 1998, para padronização no W3C, a linguagem HTML+TIME (W3C, 1998c).

2.4. NCL

Nested Context Language - NCL (Muchaluat-Saade et al., 2003; Muchaluat-Saade, 2003) é uma linguagem declarativa modular baseada em XML (e estruturada em 11 áreas funcionais) para especificação de documentos hipermídia. Fazendo uso das funcionalidades XML já descritas, NCL reusou, sempre que possível, módulos definidos pela linguagem SMIL (Muchaluat-Saade, 2003).

Composições em NCL, assim como em SMIL, podem conter um conjunto de nós, que podem ser objetos de mídia ou outros nós de composição. Entretanto, em NCL, composições (elemento *composite*) não possuem semântica temporal, mas de inclusão e, conseqüentemente, sua semântica é dada por seus elos.

A Figura 2:3 mostra um exemplo de um documento NCL similar ao documento SMIL da Figura 2:2. Para se obter a semântica do documento SMIL, elos NCL devem ser especificados, sincronizando os nós de cada composição. Esses elos NCL devem refletir a sincronização paralela e seqüencial das composições SMIL e também refletir o elo SMIL entre *audio1* e *img1*. Elos NCL são definidos separadamente dos nós por eles relacionados, estando agrupados em bases de elos (elemento *linkBase*). Essa abordagem permite, por exemplo, o reuso de nós sem a herança obrigatória de elos. Outra opção para especificar esse documento NCL é através de atribuição semântica, no caso paralela ou seqüencial, a composições NCL. Como será visto, isso é possível através de *templates* de composição.

```
01.<ncl>
02.  <head>
03.   <layout>
```

```
04.    ...
05.   </layout>
06.    ...
07. </head>
08. <body id=" ">
09.   <composite>
10.     <composite>
11.       <video id="video1" />
12.       <audio id="audio1" />
13.       <img id="img1"/>
14.       <linkBase>
15.         ...
16.        <link id="link1" xconnector="finishes.xml">
17.         <bind component="audio1" role="on_x_presentation_end"/>
18.         <bind component="img1" role="stop_y"/>
19.        </link>
20.         ...
21.       </linkBase>
22.     </composite>
23.     <composite>
24.       <audio id="audio2" />
25.       <audio id="audio3" />
26.       <linkBase>
27.         ...
28.        <link id="link2" xconnector="meets-start.xml">
29.         <bind component="audio2" role="on_x_presentation_end"/>
30.         <bind component="audio3" role="start_y"/>
31.        </link>
32.         ...
33.       </linkBase>
34.     </composite>
35.     <linkBase>
36.       ...
37.     </linkBase>
38.    </composite>
39. </body>
40.</ncl>
```

Figura 2:3 - Exemplo de um documento NCL.

Elos NCL são definidos através de conectores hipermídia. Um conector especifica uma relação de forma independente do relacionamento, ou seja, não especifica quais serão os nós relacionados. Elos que representam um mesmo tipo de relação, mas que interligam nós distintos, podem reusar um mesmo conector (aproveitando toda sua especificação). Um conector hipermídia especifica um conjunto de pontos de interface, chamados papéis (*roles*). Para a criação de um elo, faz-se referência a um conector e define-se um conjunto de *binds*, que associam cada extremidade do elo (ponto de interface de um nó) a um papel do

conector. Conectores são especificados através da linguagem XConnector (Muchaluat-Saade, 2003) de NCL.

Na Figura 2:3 é possível observar a especificação de dois elos (por clareza, os demais elos do documento foram omitidos). O elo (elemento *link*) *link1* define que, ao término de *audio1*, deve ser interrompida a apresentação de *img1*. Essa semântica é obtida por meio do conector especificado no documento *finishes.xml*, onde são declarados os papéis *on_x_presentation_end* e *stop_y*. Os elementos do tipo *bind* desse elo fazem a associação do áudio e da imagem com os papeis definidos pelo conector. Já o elo *link2* define que *audio3* deve ser iniciado após o término de *audio2*, utilizando o conector *meets-start.xml*, que possui essa semântica. Os *binds* desse elo mapeiam os dois componentes do tipo áudio com os papéis *on_x_presentation_end* e *start_y* desse conector. Como pode ser observado, o elo *link1* do documento NCL representa o elo SMIL definido por meio de evento no documento da Figura 2:2, enquanto o elo *link2* representa o comportamento seqüencial obtido pelo uso da composição *seq* em SMIL. Ao contrário de SMIL, ambos os relacionamentos são definidos da mesma forma em NCL.

Para melhor ilustrar o uso de conectores, a Figura 2:4 apresenta um conector *R* representando uma relação com três papéis distintos, que significam três tipos de participantes da relação. A Figura 2:4 também mostra dois elos, *l1* e *l2*, reusando *R*. Enquanto o conector define o tipo de relação, o conjunto de *binds* de um elo define os participantes. O elo *l1* especifica três *binds*, interligando os nós *A*, *B* e *C* aos papéis de *R*. O mesmo ocorre com *l2*, só que interligando um conjunto diferente de nós (*B*, *C* e *D*). Os elos *l1* e *l2* definem relacionamentos diferentes, já que interligam conjuntos distintos de nós, mas representam o mesmo tipo de relação, pois usam o mesmo conector.

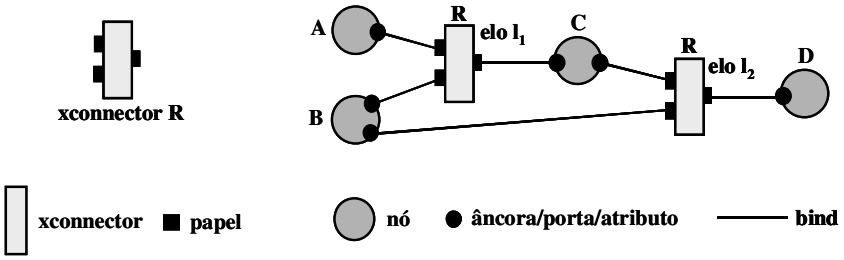


Figura 2:4 - Exemplo de elos NCL e de reuso de conectores.

Um *template* de composição, como mencionado, pode ser utilizado para embutir semântica, por exemplo temporal, em um nó de composição. *Templates* de composição são definidos através da linguagem XTemplate (Muchaluat-Saade, 2003) de NCL. A definição de um *template* de composição é feita através de um vocabulário, que especifica tipos de componentes, tipos de relações (modeladas por conectores) e pontos de interface. A definição também pode incluir restrições sobre elementos do vocabulário e conter instâncias de componentes e relacionamentos entre componentes (modelados por elos).

Um exemplo do uso de um *template* de composição pode ser visto na Figura 2:5, onde é definido que um áudio deve ser sincronizado com um *logo* (sincronismo de início e término de apresentação), e que cada trecho do áudio deve ser sincronizado, de forma similar, com a respectiva legenda[14]. Quando esse *template* é herdado por uma composição com um nó de áudio e três legendas (como na Figura 2:5), a composição passa a conter elos de sincronismo entre as legendas e o áudio, e entre o áudio e o *logo*. É importante ressaltar que o *logo*, definido como instância de componente no *template*, também passou a ser contido pela composição, após o processamento do *template*[15].

[14] Note que o conector utilizado na especificação dos elos de sincronismo para término de apresentação (conector *P*, ou *finishes*, na Figura 2:5) é o mesmo utilizado no exemplo da Figura 2:3.

[15] O *Processador de Templates* é responsável por atribuir a semântica de um *template* à composição que o referencia.

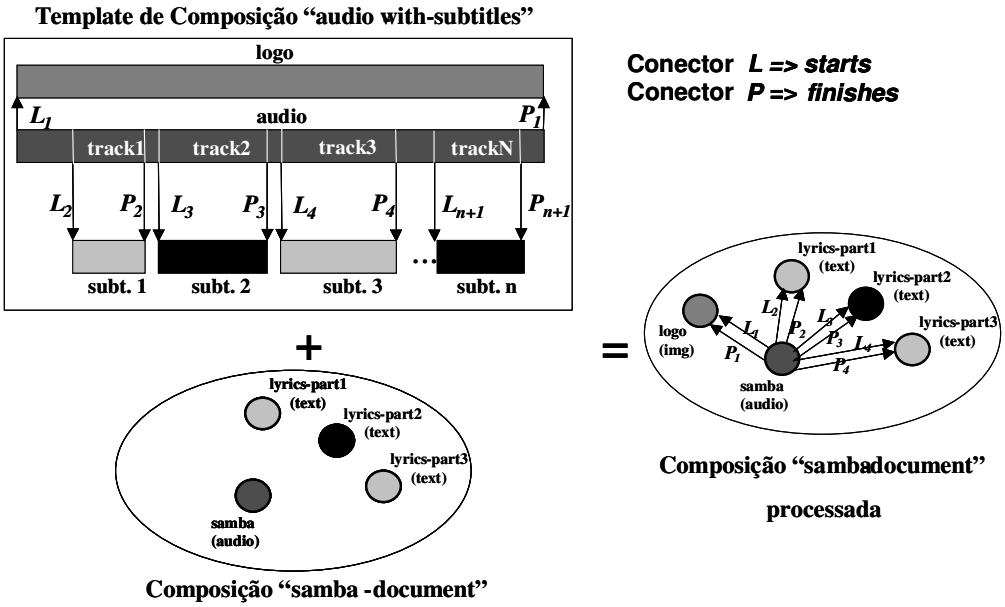


Figura 2:5 - Exemplo do uso de *templates* de composição.

Maiores detalhes sobre as especificações de *templates* de composição serão abordados no próximo capítulo, ao se apresentar a proposta de extensão da linguagem XTemplate.

3
Nested Context Language 2.1

NCL - *Nested Context Language* - é uma linguagem declarativa para autoria de documentos hipermídia baseada no modelo conceitual NCM - *Nested Context Model* (Casanova et al., 1991; Soares et al., 1995; Soares et al., 2000; Soares et al., 2003). A primeira versão de NCL (Antonacci, 2000; Antonacci et al., 2000a; Antonacci et al., 2000b) foi especificada por meio de uma única DTD. Utilizando as tecnologias XML descritas no Capítulo 2, NCL 2.0 (Silva et al., 2004a; Muchaluat-Saade et al., 2003; Muchaluat-Saade, 2003) foi definida de forma modular, usufruindo de todas as vantagens citadas para essa forma de definição. Além da abordagem modular, NCL 2.0 apresenta como principais diferenças, em relação à versão anterior da linguagem, os conceitos de *templates* de composição e de conectores hipermídia. Este capítulo descreve a linguagem NCL 2.1 (cuja especificação encontra-se no Apêndice B), destacando suas diferenças em relação à NCL 2.0.

3.1.
Funções de Custo

CostFunctions é o primeiro módulo introduzido por NCL 2.1, possibilitando a definição de funções de custo. Funções de custo permitem especificar a duração de objetos de maneira flexível, ou seja, ao invés de possuir um único valor, essa duração pode ser definida como uma faixa de valores aceitáveis. A partir dos custos associados a cada valor, é oferecida uma métrica que auxilia o formatador hipermídia na escolha da melhor configuração temporal para uma determinada apresentação (Bachelet et al., 2000).

O módulo *CostFunctions* define os seguintes elementos: *costFunctionBase*, *costFunction* e *costFunctionParam*. O conteúdo e atributos desses elementos podem ser consultados na Tabela 1[16].

Elemento	**Atributos**	**Conteúdo**
costFunctionBase	*id*, *ref*	*costFunction+*
costFunction	*id*, *type*, *ref*, *mathMLRef*	*costFunctionParam**
costFunctionParam	*name*, *value*	

Tabela 1 - Elementos do módulo *CostFunctions*.

Funções de custo são agrupadas em bases de funções de custo (elemento *costFunctionBase*), declaradas, como será visto, no cabeçalho (elemento *head*) de um documento NCL. Bases de funções de custo podem declarar um atributo *ref* para referenciar uma base de um outro documento NCL. A base *CFB2* (linha 14 da Figura 3:1), por exemplo, referencia a base de funções de custo *costFuncBaseA* definida em "*doc2.ncl*".

```
01  <costFunctionBase id="CFB1">
02    <costFunction id="costFunction22" type="linear">
03     <costFunctionParam name="deltaShrink" value="20%"/>
04     <costFunctionParam name="deltaStretch" value="20%"/>
05     <costFunctionParam name="maxShrinkCost" value="2000"/>
06     <costFunctionParam name="maxStretchCost" value="2000"/>
07    </costFunction>
08    <costFunction id="costFunction24" ref="costFunction22">
09     <costFunctionParam name="maxShrinkCost" value="4000"/>
10    </costFunction>
11    <costFunction id="costFunction02" ref="doc1.ncl#costFunctionA" />
12    <costFunction id="costFunction03" mathMLRef="mathFunction1.xml" />
13   </costFunctionBase>
14   <costFunctionBase id="CFB2" ref="doc2.ncl#costFuncBaseA/>
```

Figura 3:1 - Exemplo de funções de custo em NCL 2.1.

Funções de custo (elemento *costFunction*) possuem um identificador (atributo *id*), um tipo[17] (atributo *type*) e, possivelmente, uma referência ou para outra função de custo (atributo *ref*) ou para um arquivo em MathML (W3C, 2003) (atributo *mathMLRef*)[18]. A função de custo *costFunction02* (linha 11) referencia a

[16] Os símbolos das tabelas de elementos apresentadas neste texto possuem os seguintes significados: (?) opcional, (|) ou, (*) zero ou mais ocorrências, (+) uma ou mais ocorrências.

[17] O tipo da função de custo não é restringido por NCL, sendo necessário que os compiladores de documentos dessa linguagem (ver capítulo 5) interpretem o valor desse atributo.

[18] A interpretação de funções de custo definidas por documentos MathML é responsabilidade dos compiladores NCL.

função *costFunctionA* de "*doc1.ncl*", enquanto *costFunction03* (linha 12) referencia a função declarada pelo arquivo MathML "*mathFunction1.xml*".

Parâmetros (elemento *costFunctionParam*) também podem ser definidos para funções de custo, especificados através de um nome (atributo *name*) e um valor (atributo *value*). Na Figura 3:1, a função de custo *costFunction22* (linhas 2-7), do tipo linear, define quatro parâmetros (linhas 3-6). Esses parâmetros determinam que os objetos que referenciam essa função podem ter suas durações reduzidas (*deltaShrink*) ou aumentadas (*deltaStretch*) em até 20%, sendo o custo máximo respectivo para cada ajuste (*maxShrinkCost* e *maxStretchCost*) de 2000 (o custo de ajuste aumenta linearmente até atingir seu valor máximo, já que a função é do tipo linear). Combinando o uso de parâmetros com o atributo *ref*, é possível redefinir parcialmente uma função de custo. Por exemplo, a função *costFunction24* (linhas 8-10) referencia a função *costFunction22* (herdando seus atributos e seu tipo) e redefine o custo máximo de redução (*maxShrinkCost*) para 4000. Ou seja, objetos referenciando *costFunction24* apresentam um maior grau de dificuldade para terem suas durações reduzidas quando comparados a objetos que referenciem *costFunction22*[19].

3.2. Regras de Apresentação

NCL permite a especificação de alternativas *de conteúdo*, assim como alternativas *de exibição*, para documentos. Conteúdos (nós) alternativos são agrupados em elementos *switch*, devendo uma entre as alternativas ser escolhida para exibição, dependendo do contexto de apresentação (ver Capítulo 1). De forma análoga, alternativas de exibição são oferecidas por meio de *switches* de descritores (elemento *descriptorSwitch*), que contêm conjuntos de descritores alternativos. Descritores (elemento *descriptor*) especificam as informações de apresentação de nós de forma independente da definição do nó (ou seja, pode-se definir diferentes especificações de apresentação para um mesmo nó, associando-se diferentes descritores a ele).

[19] A associação de objetos a funções de custo é realizada pelo uso do atributo *costFunction* (também definido pelo módulo de funções de custo), como será visto na Seção 3.3.

Em NCL 2.0, a escolha entre alternativas (de nós ou de descritores) é realizada através de atributos de teste, da mesma maneira que em SMIL[20]. Assim, atributos de teste devem ser declarados juntamente com elementos NCL, e, caso todos os atributos sejam avaliados como verdadeiros no contexto de apresentação, esse elemento está apto a ser escolhido entre as alternativas.

Na Figura 3:2 é apresentado um nó *switch* em NCL 2.0. Entre as alternativas, existem 3 nós de áudio (*aEn1*, *aPt1* e *aFr1*), que somente podem ser escolhidos para a apresentação se a largura de banda disponível for maior ou igual a 128Kbps e a língua definida for inglês, português ou francês, respectivamente. Note que o autor, ao desejar que o mesmo áudio seja apresentado caso o leitor do documento possua conhecimento de qualquer uma dessas três línguas, teve de definir três nós referenciando - pelo atributo *src* - o mesmo arquivo de áudio. Em seguida, estão declarados um nó de vídeo (*vid1*) e um nó de imagem (*img1*), que são elegíveis para exibição se a banda disponível for maior ou igual a 128Kbps e 64Kbps, respectivamente. Finalmente, há um nó de texto que, por não possuir atributos de teste, pode ser exibido sem restrições. Quando o *switch* for avaliado, o primeiro nó apto para exibição (na seqüência do documento) será o escolhido entre as alternativas.

```
01  <switch id="intro">
02   <audio id="aEn1" src="a1.wav" systemLanguage="en" systemBitrate="128000" />
03   <audio id="aPt1" src="a1.wav" systemLanguage="pt" systemBitrate="128000" />
04   <audio id="aFr1" src="a1.wav" systemLanguage="fr" systemBitrate="128000" />
05   <video id="vid1" src="v1.mpg" systemBitrate="128000" />
06   <img   id="img1" src="i1.jpg" systemBitrate="64000"/>
07   <text  id="txt1" src="t1.txt" />
08  </switch>
```

Figura 3:2 - Exemplo de um nó *switch* em NCL 2.0

NCL 2.1 introduz o módulo *TestRules* para definição de regras de apresentação a serem usadas para a escolha entre alternativas de conteúdo e de descritores. Os elementos definidos por esse módulo estão apresentados na Tabela

[20] SMIL predefine os seguintes atributos de teste: *systemAudioDesc*, *systemBitrate* (ou *system-bitrate*), *systemCaptions* (ou *system-captions*), *systemComponent*, *systemCPU* e *systemLanguage* (ou *system-language*). Outros atributos podem ser definidos através do uso de *Custom Test Attributes*.

2. O elemento *presentationRuleBase* define uma base[21] que agrupa os elementos que representam regras de apresentação (ver Figura 3:3). Essas regras podem ser simples (elemento *presentationRule*) ou compostas (elemento *compositePresentationRule*). Regras simples definem um identificador (*id*), uma variável (*var*), um operador (*op*) e um valor (*value*). O operador relaciona a *variável* ao *valor* e pode ser uma das constantes: *lt* (*less than* - menor que), *le* (*less than or equal to* - menor que ou igual a), *gt* (*greater than* - maior que), *ge* (*greater than or equal to* - maior que ou igual a), *ne* (*not equal* - diferente de) ou *eq* (*equal to* - igual a). Regras compostas, por sua vez, possuem um identificador (*id*) e um operador (*op*), que pode ter os valores *and* (operador booleano "e") ou *or* (operador booleano "ou"). Esse operador se aplica às regras (simples ou compostas) declaradas como elementos filhos da regra composta. Assim como bases de função de custo e funções de custo, bases de regras de apresentação e regras de apresentação podem referenciar outras bases ou regras pelo atributo *ref*.

Elemento	**Atributos**	**Conteúdo**
presentationRuleBase	*id*, *ref*,	(*presentationRule* | *compositePresentationRule*)+
presentationRule	*id*, *var*, *op*, *value*, *ref*	
compositePresentationRule	*id*, *op*	(*presentationRule* | *compositePresentationRule*)+
bindRule	*component*, *rule*	

Tabela 2 - Elementos do módulo *TestRules*.

Enquanto em NCL 2.0 a escolha entre alternativas é realizada a partir dos atributos de teste, em NCL 2.1, essa escolha é baseada nas regras de apresentação. Dessa forma, NCL 2.1 elimina o uso dos atributos de teste e define o elemento *bindRule* (do módulo *TestRules*), que foi adicionado aos elementos *switch* e *descriptorSwitch*. O elemento *bindRule* possui os atributos componente (*component*) e regra (*rule*) que são utilizados para associar regras de apresentação a componentes pertencentes a um nó com alternativas. De forma similar à NCL 2.0, o primeiro componente, na ordem do documento, que tem sua regra avaliada

[21] Bases de regras de apresentação, como será visto, são declaradas no cabeçalho do documento NCL.

como verdadeira (ou que não possui regra associada), é o escolhido entre as alternativas. A associação entre regras de apresentação e componentes por meio do elemento *bindRule* permite que um componente seja reusado (em um outro contexto - Soares et al., 2003) sem herança de regras.

O *switch* "*intro*" na Figura 3:3 (linhas 18-26), por exemplo, é semanticamente igual ao *switch* homônimo na Figura 3:2. Como pode ser observado, a regra composta *rEnPtFrBBand* (linhas 8-15), ao ser associada ao componente *aud1* (linha 19), corresponde à definição dos três nós de áudio e seus respectivos atributos de teste na Figura 3:2 (ilustrando a facilidade introduzida pelo uso de regras compostas com o operador "ou"). A regra *rBBand* (linha 6) - reusada na definição de *EnPtFrBBand* - é associada ao vídeo *vid1* (linha 20), enquanto *rNBand* (linha 7) é associada à imagem *img1* (linha 21). Um exemplo que envolve alternativas de conteúdo e alternativas de apresentação será apresentado na próxima seção.

```
01  ...
02  <presentationRuleBase>
03    <presentationRule id="rEn" var="systemLanguage" op="eq" value="en" />
04    <presentationRule id="rFr" var="systemLanguage" op="eq" value="fr" />
05    <presentationRule id="rPt" var="systemLanguage" op="eq" value="pt" />
06    <presentationRule id="rBBand" var="systemBitrate" op="ge" value="128000" />
07    <presentationRule id="rNBand" var="systemBitrate" op="ge" value="64000" />
08    <compositePresentationRule id="rEnFrPtBBand" op="and">
09     <compositePresentationRule id="EnFrPt" op="or">
10      <presentationRule ref="ruleEn" />
11      <presentationRule ref="ruleFr" />
12      <presentationRule ref="rulePt" />
13     </compositePresentationRule>
14     <presentationRule ref="rBBand" />
15    </compositePresentationRule>
16   </presentationRuleBase>
17  ...
18  <switch id="intro">
19    <bindRule rule="rEnPtFrBBand" component="aud1"/>
20    <bindRule rule="rBBand" component="vid1"/>
21    <bindRule rule="rNBand" component="img1"/>
22    <audio id="aud1" src="a1.wav" implicitDur="10" />
23    <video id="vid1" src="v1.mpg" />
24    <img   id="img1" src="i1.jpg" />
25    <text  id="txt1" src="t1.txt" />
26   </switch>
```

Figura 3:3 - Exemplo de regras de apresentação e nó *switch* em NCL 2.1.

3.3. Refinamentos de NCL 2.1

Em conjunto com os novos módulos especificados por NCL 2.1, pode-se destacar alguns refinamentos propostos para os módulos existentes em NCL 2.0. Esses refinamentos serão apresentados a partir do documento NCL da Figura 3:4, que estende os exemplos das seções anteriores.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <ncl ...>
03  <head>
04   <layout>...</layout>
05   <presentationRuleBase>
06    <presentationRule id="rBegin" var="userLevel" op="eq" value="beginner"/>
07    <presentationRule id="rExpt" var="userLevel" op="eq" value="expert"/>
08    <presentationRule id="rSynthT" var="synthEnabled" op="eq" value="true"/>
09    <presentationRule id="rSynthF" var="synthEnabled" op="eq" value="false"/>
10    <presentationRule id="rEn" var="systemLanguage" op="eq" value="en" />
11    <presentationRule id="rFr" var="systemLanguage" op="eq" value="fr" />
12    <presentationRule id="rPt" var="systemLanguage" op="eq" value="pt" />
13    <presentationRule id="rBBand" var="systemBitrate" op="ge" value="128000" />
14    <presentationRule id="rNBand" var="systemBitrate" op="ge" value="64000" />
15    <compositePresentationRule id="rSynAcntEn" op="and">
16     <presentationRule ref="rSynthT"/>
17     <presentationRule id="rAcntEn" var="speechAccent" op="eq" value="en"/>
18    </compositePresentationRule>
19    <compositePresentationRule id="rSynAcntUk" op="and">
20     <presentationRule id="rSynthB" ref="rSynthT"/>
21     <presentationRule id="rAcntUk" var="speechAccent" op="eq" value="en-uk"/>
22    </compositePresentationRule>
23    <compositePresentationRule id="rEnFrPtBBand" op="and">
24     <compositePresentationRule id="EnFrPt" op="or">
25      <presentationRule ref="ruleEn" />
26      <presentationRule ref="ruleFr" />
27      <presentationRule ref="rulePt" />
28     </compositePresentationRule>
29     <presentationRule ref="rBBand" />
30    </compositePresentationRule>
31   </presentationRuleBase>
32   <costFunctionBase id="CFB1">
33    <costFunction id="costFunction22" type="linear">
34     <costFunctionParam name="deltaShrink" value="20%"/>
35     <costFunctionParam name="deltaStretch" value="20%"/>
36     <costFunctionParam name="maxShrinkCost" value="2000"/>
37     <costFunctionParam name="maxStretchCost" value="2000"/>
38    </costFunction>
39    <costFunction id="costFunction24" ref="costFunction22">
40     <costFunctionParam name="maxShrinkCost" value="4000"/>
41    </costFunction>
```

```
42    <costFunction id="costFunction02" ref="doc1.ncl#costFunctionA" />
43    <costFunction id="costFunction03" mathMLRef="mathFunction1.xml" />
44   </costFunctionBase>
45   <costFunctionBase id="CFB2" ref="doc2.ncl#costFuncBaseA/>
46   <descriptorBase>
47    <descriptor id="videoDesc" dur="30" .../>
48    <descriptor id="subtDesc" nodeRule="rSynthT".../>
49    <descriptorSwitch id="explanationDesc">
50     <bindRule rule="rSynAcntEn" component="speechDescEn"/>
51     <bindRule rule="rSynAcntUk" component="speechDescUk"/>
52     <bindRule rule="rSynthT" component="textDesc"/>
53     <bindRule rule="rSynthF" component="textDesc"/>
54     <descriptor id="dEthEn" costFunction="costFunction24" player="ETH">
55      <descriptorParam name="idiom" value="en"/>
56     </descriptor>
57     <descriptor id="dEthUk" costFunction="costFunction24" player="ETH">
58      <descriptorParam name="idiom" value="en-uk"/>
59     </descriptor>
60     <descriptor id="dTTS" costFunction="costFunction22" player="TTS" />
61     <descriptor id="textDesc" .../>
62    </descriptorSwitch>
63   </descriptorBase>
64  </head>
65  <body>
66   <port id="entryPoint" component="intro">
67   <switch id="intro">
68    <bindRule rule="rEnPtFrBBand" component="aud1"/>
69    <bindRule rule="rBBand" component="vid1"/>
70    <bindRule rule="rNBand" component="img1"/>
71    <audio id="aud1" src="a1.wav" implicitDur="10" />
72    <video id="vid1" src="v1.mpg" />
73    <img   id="img1" src="i1.jpg" />
74    <text  id="txt1" src="t1.txt" />
75   </switch>
76   <composite id="c1">
77    <video id="video" descriptor="videoDesc" src="..."/>
78    <text id="subtitle" descriptor="subtDesc" src="..."/>
79    <switch id="explanation">
80     <bindRule rule="rBegin" component="beginnerExplanation"/>
81     <bindRule rule="rExpt" component="expertExplanation"/>
82     <text id="beginnerExplanation" src="..." descriptor="explanationDesc"/>
83     <text id="expertExplanation" src="..." descriptor="explanationDesc"/>
84    </switch>
85    <linkBase>...</linkBase>
86   </composite>
87   <composite id="audios">
88    <bindDescriptor component="subtitleb" descriptor="dTTS" />
89    <audio id="aud1b" ref="aud1" />
90    <text id="subtitleb" ref="subtitle"/>
91  </composite>
92   <linkBase>...</linkBase>
```

```
93  </body>
94  </ncl>
```

Figura 3:4 - Exemplo de documento NCL 2.1.

O documento da Figura 3:4 representa uma palestra, onde existe uma introdução (*switch* "*intro*", linhas 67-75) e uma explicação, representada pela composição "*c1*" (linhas 76-86). A porta (elemento *port*, linha 66) (Muchaluat-Saade et al., 2003) determina o ponto de entrada do documento (ou seja, o componente que representa o início da apresentação) como sendo o componente "*intro*". Pelos elos (omitidos na figura) definidos na base de elos da composição *body* (linha 92), o término do *switch* "*intro*" deve iniciar a apresentação da composição "*c1*". A composição "*c1*" possui três componentes (um vídeo, um texto e um *switch*) a serem exibidos em paralelo, pela semântica obtida dos elos dessa composição (também omitidos na figura e agrupados na base de elos da linha 85). Finalmente, o exemplo apresenta uma composição "*audios*" que agrupa os áudios do documento. Essa composição ilustra o uso de composições para estruturação de um documento independente de como ele deve ser apresentado (Muchaluat-Saade, 2003) e não apresenta nenhum sincronismo temporal com os outros componentes do documento.

O *switch* "*intro*", explicado na Seção 3.2, reflete o desejo do autor de apresentar o áudio *aud1* (linha 71) caso o usuário tenha *conhecimento* de inglês, francês ou português; e a largura de banda disponível seja maior ou igual a 128Kbps. Caso contrário, dependendo da largura de banda disponível, será exibido um vídeo (linha 72), uma imagem (linha 73) ou um texto (linha 74).

O outro nó *switch* do documento, chamado "*explanation*" (linhas 79-84), pertence à composição "*c1*" e oferece duas alternativas para um texto explicativo, uma para usuários experientes (regra *rExpt* na linha 7) e outra para usuários iniciantes (regra *rBegin* na linha 6). Entretanto, existem mais alternativas para exibição desses textos, já que suas características de apresentação são especificadas pelo *switch* de descritores *explanationDesc* (linhas 49-62) - referenciado pelo atributo *descriptor* dos nós de texto. Esse *descriptorSwitch* oferece quatro alternativas, que devem ser escolhidas avaliando as regras de apresentação a elas associadas. As duas primeiras alternativas (linhas 54-56 e 57-59) definem que nós referenciando esses descritores devem ser exibidos como áudios e sincronizados com uma animação facial - atributo *player* do descritor é

especificado como ETH: *Expressive Talking Heads* (Lucena, 2002; Rodrigues et al., 2004). Essas alternativas são válidas quando é admitida a apresentação de áudios sintetizados (*synthEnabled* tem valor verdadeiro) e a caracterização da voz (*speechAccent*) é, respectivamente, inglês americano (*en*) ou inglês britânico (*en-uk*) - ver regras *rSynAcntEn* (linhas 15-18) e *rSynAcntUk* (linhas 19-22). A terceira alternativa (linha 60), válida quando *synthEnabled* é verdadeiro (regra *rSynthT*, linha 8), especifica a exibição de nós de texto como áudios por meio de uma ferramenta de conversão de texto para voz (*player* TTS - *Text-to-Speech*), sem a apresentação de uma animação facial. Finalmente, a última alternativa (linha 61) determina a utilização do exibidor padrão para nós (por não declarar o atributo *player*), apresentando-os no formato original.

Observando os descritores citados, pode-se perceber o uso do atributo *costFunction* (definido pelo módulo de funções de custo), que foi adicionado, por NCL 2.1, ao elemento *descriptor* para possibilitar que descritores referenciem uma função de custo. Na Figura 3:4, os descritores *dEthEn* e *dEthUk* (linhas 54-56 e 57-59) referenciam a função de custo *costFunction24* (linhas 39-41), determinando os custos para ajustes realizados pelo *player* ETH; enquanto o descritor *dTTS* referencia a função *costFunction22* (linhas 33-38). Analisando as funções de custo referenciadas, é possível observar que o custo de ajustes é menor no *player* TTS quando comparado ao ETH. Além desse novo atributo, descritores em NCL 2.1 também podem declarar elementos filhos *descriptorParam*, que, de forma similar ao elemento *costFunctionParam*, são utilizados para parametrizar os *players* especificados pelos descritores. Na Figura 3:4, esse elemento é utilizado para determinar o idioma do sintetizador de voz do ETH (linhas 55 e 58).

A Figura 3:5 ilustra as alternativas de apresentação obtidas pela combinação do *switch* de conteúdo "*explanation*" com o *switch* de descritores "*explanationDesc*" da Figura 3:4. Também estão ilustradas as regras de apresentação para escolha entre alternativas e as funções de custo referenciadas pelos descritores. Como pode ser observado, são oito as possibilidades de exibição.

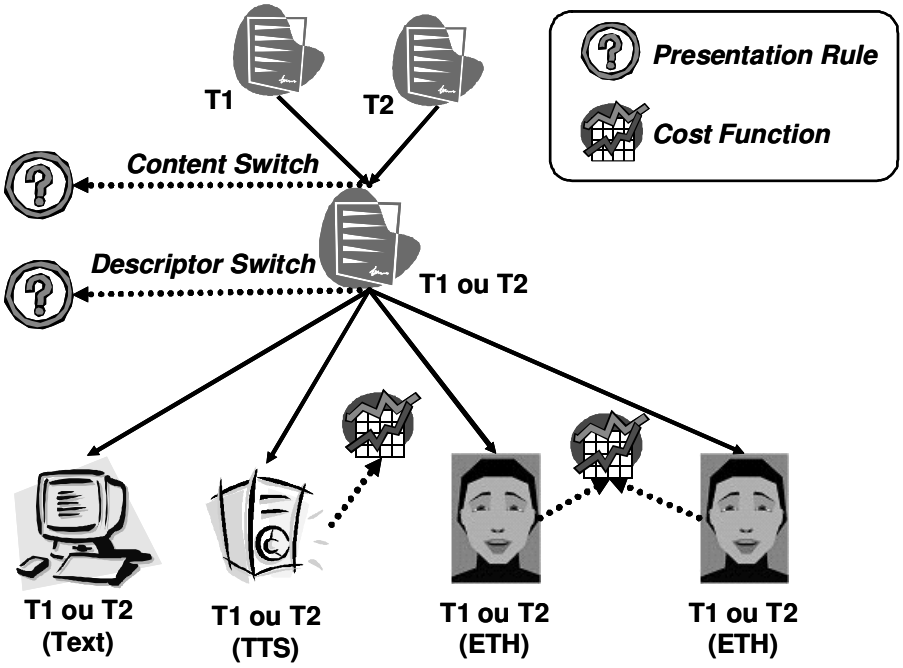


Figura 3:5 - *Switch* de conteúdo e de descritores.

Descritores, em NCL 2.1, também podem definir um atributo *nodeRule* (do módulo *TestRules*) para especificar uma regra de apresentação a ser aplicada diretamente a nós. Esse atributo faz com que os nós que referenciam esses descritores somente sejam exibidos se a regra apontada por *nodeRule* for avaliada como verdadeira (ver exemplo na linha 48). Na Figura 3:4, a definição do nó de texto na linha 78 equivaleria à definição de um *switch*, no qual a única alternativa seria esse nó, e a regra para escolha seria aquela referenciada pelo atributo *nodeRule*: *rSynthT*.

A versão 2.1 de NCL altera a maneira como elementos são reusados. Em NCL 2.0, existe o elemento *ref*, que pode referenciar outros nós do documento. Entretanto, em NCL 2.1, com a eliminação do *elemento ref*, o reuso de elementos foi uniformizado, sendo feito sempre através de um *atributo ref* (adicionado a diversos elementos da linguagem). Assim, composições e outros nós podem especificar um atributo *ref* (linhas 89 e 90), reusando outros componentes do documento. O mesmo é válido para o reuso de descritores, base de descritores, regras de exibição, base de regras de exibição, *layout*, funções de custo e base de funções de custo (como foi visto anteriormente). Atributos *ref* podem referenciar um elemento do próprio documento (pelo seu *id*) ou elementos em outros documentos (seguindo a sintaxe: *nome-do-arquivo#id-do-elemento*).

NCL 2.1 redefine o nome do elemento *componentPresentation* para *bindDescriptor*, refletindo a nova nomenclatura adotada pela linguagem. Esse elemento possibilita que composições e *switches* especifiquem descritores para os nós por eles contidos. Na Figura 3:4, a composição "*áudios*" (linhas 87-91) contem um nó de áudio e um nó de texto (que referenciam outros nós do documento) e especifica o descritor *dTTS* (linha 60) para o nó de texto. Dessa forma, esse nó de texto será apresentado como áudio utilizando o *player* TTS.

NCL 2.1 também define o atributo *implicitDur* (adicionado aos elementos que representam objetos de mídia) que é utilizado para determinar a duração implícita de nós (ver linha 71). Essa duração, que corresponde à duração do arquivo de mídia que esses nós referenciam (por exemplo, o tempo de exibição de um vídeo), pode guiar o formatador na elaboração do seu plano de apresentação e de pré-busca (Rodrigues, 2003). Apesar das durações implícitas de nós poderem ser, na maioria das vezes, obtidas pela simples análise do conteúdo de arquivos, ao defini-las em um documento NCL, evita-se, por exemplo, que o formatador tenha que consultar um servidor remoto para obter essa informação.

O atributo *implicitDur*, é importante ressaltar, tem semântica diferente do atributo *explicitDur* (também adicionado por NCL 2.1) que é definido por descritores (linha 47 na Figura 3:4). O atributo *explicitDur* especifica a duração explicita que nós devem ter em uma apresentação, independente de suas durações implícitas - ou seja, podem ser necessários ajustes para garantir esse tempo de exibição.

A Tabela 3 resume as alterações propostas por NCL 2.1, destacando os nomes alterados; os atributos e elementos adicionados (e a quais elementos eles foram adicionados); e os elementos e atributos que deixaram de existir em NCL 2.1. A tabela completa dos elementos NCL 2.0, com seus conteúdos e atributos, pode ser consultada em (Muchaluat-Saade et al., 2003), estando reproduzida no Apêndice A, juntamente com a tabela completa de NCL 2.1.

Elementos Renomeados	
De	Para
presentationSpecification	*bindDescriptor*
Atributos Adicionados	
Nome	Ao(s) elemento(s)

nodeRule	*descriptor*
explicitDur	*descriptor*
implicitDur	*animation*, *audio*, *img*, *text*, *textstream*, *video*
ref	*animation*, *audio*, *img*, *text*, *textstream*, *video*, *composite*, *descriptor*, *descriptorSwitch*, *switch*, *layout*
Elementos Adicionados	
Nome	Ao(s) elemento(s)
descriptorParam	*descriptor*
bindRule	*switch, descriptorSwitch*
Elemento Retirado	
ref	
Atributos Retirados	
TestAttributes (os mesmos de SMIL 2.0)	

Tabela 3 - Diferenças entre NCL 2.1 e NCL 2.0.

3.4. Linguagem XConnector

Conectores hipermídia (ver Seção 2.4) são especificados através da linguagem XConnector de NCL (Muchaluat-Saade et al., 2002). A linguagem XConnector permite a definição de relações multiponto com semântica causal ou de restrição, que são usadas na especificação de relacionamentos (elos) em NCL. Em uma relação causal, uma condição deve ser satisfeita para que uma ação seja executada (por exemplo: a seleção de uma âncora de um nó de origem causa a navegação para um nó de destino), enquanto, em uma relação de restrição, é especificada uma restrição sem qualquer causalidade envolvida (por exemplo: dois nós devem ter suas exibições encerradas simultaneamente).

A definição de um conector é feita por um conjunto de papéis e um *glue*. Papéis determinam a função dos participantes da relação, por meio de eventos (Figura 3:6) e transições da máquina de estados dos evento (Tabela 4). Os tipos básicos de eventos em XConnector são: *presentation* (apresentação de um conjunto de unidades de informação de um objeto de mídia), *mouseClick* (clique do mouse sobre um conjunto de unidades de informação de um objeto de mídia),

mouseOver (posicionamento do mouse sobre um conjunto de unidades de informação de um objeto de mídia), *focus* (foco no elemento de interface do usuário representando um conjunto de unidades de informação de um objeto de mídia), *prefetch* (pré-busca de um conjunto de unidades de informação de um objeto de mídia) e *attribution* (atribuição de um valor a um atributo de um objeto de mídia). O *glue*, por sua vez, descreve como os papéis interagem, especificando a combinação desses papéis de acordo com a semântica de causalidade ou de restrição (Muchaluat-Saade, 2003).

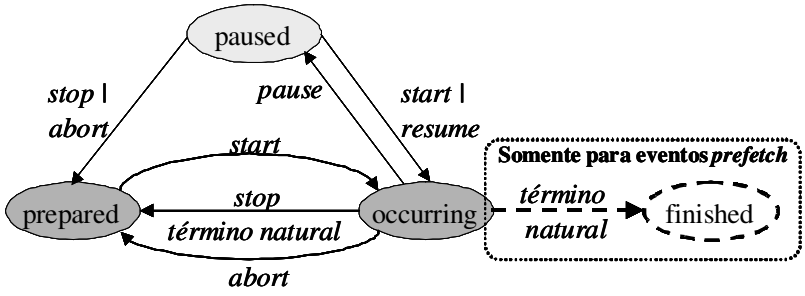


Figura 3:6 - Máquina de estados de um evento.

Transição (causada pela ação)	**Nome da Transição**
prepared → occurring (start)	*starts*
occurring → prepared (stop ou término natural)	*stops*
occurring → prepared (abort)	*aborts*
occurring → finished (término natural)	*ends*
occurring → paused (pause)	*pauses*
paused → occurring (resume ou start)	*resumes*
paused → prepared (stop ou abort)	*abortsFromPaused*

Tabela 4 - Nomes das transições para a máquina de estados de um evento.

Papéis são do tipo condição (*conditionRole*), ação (*actionRole*) ou propriedade (*propertyRole*). O elemento g*lue* pode ser causal - usado em conectores causais - definindo uma expressão de disparo (*triggerExpression*) e uma expressão de ação (*actionExpression*); ou pode ser de restrição - usado em conectores de restrição - definindo uma expressão relacionando papéis do tipo propriedade (Muchaluat-Saade, 2003).

A Figura 3:7 ilustra a especificação de dois conectores declarados na linguagem XConnector de NCL 2.0: *finishes* e *overlaps* (representando as relações

de sincronização temporal homônimas propostas por Allen, 1983[22]). O conector causal *finishes*, referenciado no exemplo da Seção 2.4, possui a seguinte semântica (ver Figura 3:8): quando o componente no papel *on_x_presentation_end* terminar sua apresentação (*eventType*="*presentation*" e *transition*="*stops*"), deve ser terminada a apresentação do componente no papel *stop_y* (*eventType*="*presentation*" e *actionType*="*stop*"). Essa semântica é obtida pelo *glue* causal, que define a expressão de disparo e a expressão de ação, referenciando, respectivamente, o papel de condição *on_x_presentation_end* e o papel de ação *stop_y*.

```
15  <xconnector id="finishes" xsi:type="CausalHypermediaConnector" >
16   <conditionRole id="on_x_presentation_end" eventType="presentation">
17    <condition xsi:type="EventTransitionCondition" transition="stops"/>
18   </conditionRole>
19   <actionRole id="stop_y" eventType="presentation" actionType="stop"/>
20   <glue>
21    <triggerExpression                          xsi:type="SimpleTriggerExpression"
conditionRole="on_x_presentation_end" />
22    <actionExpression xsi:type="SimpleActionExpression" actionRole="stop_y"/>
23   </glue>
24  </xconnector>


25  <xconnector id="overlaps" xsi:type="ConstraintHypermediaConnector" >
26   <propertyRole id="xb" eventType="presentation">
27    <property xsi:type="EventTransitionProperty" transition="starts"/>
28   </propertyRole>
29   <propertyRole id="xe" eventType="presentation">
30    <property xsi:type="EventTransitionProperty" transition="stops"/>
31   </propertyRole>
32   <propertyRole id="yb" eventType="presentation">
33    <property xsi:type="EventTransitionProperty" transition="starts"/>
34   </propertyRole>
35   <propertyRole id="ye" eventType="presentation">
36    <property xsi:type="EventTransitionProperty" transition="stops"/>
37   </propertyRole>
38   <glue>
39    <propertyExpression xsi:type="CompoundPropertyExpression" operator="and">
40     <firstProperty                        xsi:type="PropertyToPropertyExpression"
firstPropertyRole="xb" secondPropertyRole="yb" comparator="lt"/>
41     <secondProperty xsi:type="CompoundPropertyExpression" operator="and">
42      <firstProperty                       xsi:type="PropertyToPropertyExpression"
firstPropertyRole="xe" secondPropertyRole="ye" comparator="lt"/>
```

[22] A especificação de todas as relações de Allen em XConnector pode ser encontrada em Muchaluat-Saade, 2003.

```
43      <secondProperty                      xsi:type="PropertyToPropertyExpression"
firstPropertyRole="yb" secondPropertyRole="xe" comparator="lt"/>
44     </secondProperty>
45    </propertyExpression>
46   </glue>
47  </xconnector>
```

Figura 3:7 - Exemplos de conectores em NCL 2.0.

O conector de restrição *overlaps* (ver Figura 3:8), também especificado na Figura 3:7, relaciona dois componentes, denominados *x* e *y* neste texto, de forma que o início da apresentação de *x* (papel *xb*: *x begin*) deve ser anterior ao início da apresentação de *y* (papel *yb*: *y begin*) e que o término da apresentação de *x* (papel *xe*: *x end*) deve ser após o início de *y* (*yb*) e antes do término de *y* (*ye*: *y end*). Essa semântica é determinada pelo *glue* de restrição, através de três expressões de propriedade (*PropertyToPropertyExpression*) agrupadas em duas expressões compostas (*CompoundPropertyExpression*).

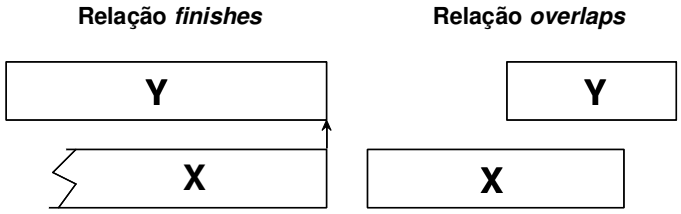


Figura 3:8 - Relações *finishes* e *overlaps*.

A Figura 3:9 ilustra a definição dos mesmos conectores da Figura 3:7 em XConnector de NCL 2.1 (cuja tabela completa de elementos pode ser consultada no Apêndice A e cuja especificação encontra-se no Apêndice C). A principal diferença dessa linguagem, em relação à sua versão anterior, é a alteração dos nomes dos elementos para refletir seus tipos diretamente - eliminando o uso de um nome genérico em conjunto com o atributo *xsi:type*. Dessa forma, os elementos que representam conectores causais e de restrição são nomeados *causalConnector* e *constraintConnector*. Os elementos filhos do tipo *glue* desses conectores recebem o nome de *causalGlue* e *constraintGlue*, respectivamente. Os demais elementos que possuem atributo *xsi:type* em XConnector 2.0 geraram, em XConnector 2.1, novos elementos com nomes iguais aos valores desses atributos, tais como: *eventTransitionCondition*, *simpleTriggerExpression*, *simpleActionExpression,* *eventTransitionProperty* e *propertyToPropertyExpression*.

```
01  <causalConnector id="finishes">
02   <conditionRole id="on_x_presentation_end" eventType="presentation">
```

```
03    <eventTransitionCondition transition="stops"/>
04   </conditionRole>
05   <actionRole id="stop_y" eventType="presentation" actionType="stop"/>
06   <causalGlue>
07    <simpleTriggerExpression conditionRole="on_x_presentation_end" />
08    <simpleActionExpression actionRole="stop_y"/>
09   </causalGlue>
10  </causalConnector>


11  <constraintConnector id="overlaps" >
12   <propertyRole id="xb" eventType="presentation">
13     <eventTransitionProperty transition="starts"/>
14   </propertyRole>
15   <propertyRole id="xe" eventType="presentation">
16    <eventTransitionProperty transition="stops"/>
17   </propertyRole>
18   <propertyRole id="yb" eventType="presentation">
19    <eventTransitionProperty transition="starts"/>
20   </propertyRole>
21   <propertyRole id="ye" eventType="presentation">
22    <eventTransitionProperty transition="stops"/>
23   </propertyRole>
24   <constraintGlue>
25    <compositePropertyExpression op="and" >
26     <propertyToPropertyExpression firstPropertyRole="xb"
secondPropertyRole="yb" op="lt" />
27     <propertyToPropertyExpression firstPropertyRole="xe"
secondPropertyRole="ye" op="lt" />
28     <propertyToPropertyExpression firstPropertyRole="yb"
secondPropertyRole="xe" op="lt" />
29    </compositePropertyExpression>
30   </constraintGlue>
31  </constraintConnector>
```

Figura 3:9 - Exemplo de conectores em NCL 2.1.

Elementos compostos também tiveram seus nomes alterados, segundo a regra: "*compoundXXX* é alterado para *compositeXXX*" (por exemplo, o elemento representando regras de exibição compostas foi renomeado de *compoundPropertyExpression* para *compositePropertyExpression*). Além da alteração de seus nomes, esses elementos compostos em XConnector 2.1 podem ter um número qualquer de elementos filhos, e não somente dois, como na versão anterior. Finalmente, os atributos operador (*operator*) e comparador (*comparator*) tiveram seus nomes alterados para *op*, sendo seus possíveis valores modificados a fim de coincidirem com os definidos pelo módulo de regras de apresentação (Seção 3.2).

3.5. Linguagem XTemplate

A maior diferença entre NCL 2.0 e NCL 2.1 se refere à redefinição da linguagem XTemplate, usada na especificação de *templates* de composição (ver Seção 2.4). XTemplate 2.0 somente permite a definição de relacionamentos através de conectores (apesar de prever uma extensão para relacionamentos de inclusão). XTemplate 2.1 (cuja especificação encontra-se no Apêndice D) estende a versão anterior da linguagem ao permitir, também, a definição de relacionamentos de inclusão. Esse tipo de relacionamento permite determinar em qual componente composto um determinado componente está contido.

A Tabela 5 apresenta os elementos de XTemplate 2.0. A definição de *templates* nessa linguagem possui duas partes (Muchaluat-Saade, 2003):

- Vocabulário (elemento *vocabulary*), que define tipos de componentes (e seus pontos de interface) e conectores presentes em uma composição; e
- Restrições (elemento *constraints*), que definem um conjunto de restrições sobre elementos do vocabulário; um conjunto de instâncias de componentes e conectores; e relacionamentos entre componentes.

Elemento	**Atributos**	**Conteúdo**
xtemplate		(*vocabulary*, *constraints*?)
vocabulary		(*component*, *connector*)*
component	*type*, *ctype*, *maxOccurs*, *minOccurs*	*port**
Port	*type, maxOccurs, minOccurs*	
connector	*type*, *src*, *maxOccurs*, *minOccurs*	
constraints		(*constraint* | *resource* | *link* | *XSLT*[23])*
Link	*type*	*bind**

[23] O conteúdo *XSLT* refere-se ao uso da linguagem XSLT (W3C, 1999d) para especificação de relacionamentos em XTemplate, como será apresentado.

Bind	*role*, *select*	
constraint	*select, description*	
resource	*src*, *type*, *label*	

Tabela 5 - Elementos da linguagem XTemplate de NCL 2.0.

A Figura 3:10 ilustra a definição do *template* utilizado como exemplo na Seção 2.4 (Figura 2:5). Conforme comentado, esse *template*, quando herdado por uma composição com um nó de áudio e legendas relativas a cada trecho do áudio, estabelece o sincronismo entre os trechos do áudio e suas respectivas legendas, além de sincronizar o início e término do áudio com um *logo* (definido pelo próprio *template*). A composição, cuja visão temporal é ilustrada na Figura 3:11, pode ser definida pela especificação de todos os seus nós componentes e dos elos que os relacionam. Alternativamente e de uma forma bem mais simples, a composição pode ser especificada apenas contendo o nó de áudio e as legendas, e fazendo referência ao *template* definido na Figura 3:10, de onde herdará o restante de sua especificação (a imagem *logo* e os elos de sincronização).

```
01  <xtemplate id="audio-with-subtitles">
02  <vocabulary>
03   <component type="song" ctype="audio" maxOccurs="1">
04    <port type="track" maxOccurs="unbounded" />
05   </component>
06    <component type="subtitle" ctype="text" maxOccurs="unbounded" />
07   <component type="logo" ctype="img" maxOccurs="1" />
08   <connector src="starts.xml" type="L" maxOccurs="unbounded" />
09   <connector src="finishes.xml" type ="P" maxOccurs="unbounded" />
10  </vocabulary>
11  <constraints>
12    <constraint select="count(child::*[@type!='song'] | child::*[@type!='logo']
| child::*[@type!='subtitle']) = (count(child::*)-count(child::linkBase))"
description="All components must be songs or logos or subtitles."/>
13   <resource src="logo.jpg" type="logo" label="logoJPG"/>
14   <link type="L">
15    <bind role="on_x_presentation_begin" select="child::*[@type='song']"/>
16    <bind role="start_y" select="child::*[@label='logoJPG']"/>
17   </link>
18   <link type="P">
19    <bind role="on_x_presentation_end" select="child::*[@ type='song']"/>
20    <bind role="stop_y" select="child::*[@label='logoJPG']"/>
21   </link>
22   <for-each select="child::*[@type='audio']/child::*[@type='track']" >
23    <variable name="i" select="position()"/>
24     <link type="L">
25      <bind role="on_x_presentation_begin" select="current()" />
26      <bind role="start_y" select="//*/child::*[@type='subtitle'][$i]"/>
```

```
27     </link>
28     <link type="P">
29      <bind role="on_x_presentation_end" select="current()" />
30      <bind role="stop_y" select="//*/child::*[@type='subtitle'][$i]"/>
31     </link>
32   </for-each>
33  </constraints>
```

Figura 3:10 - Exemplo de *template* de composição em NCL 2.0.

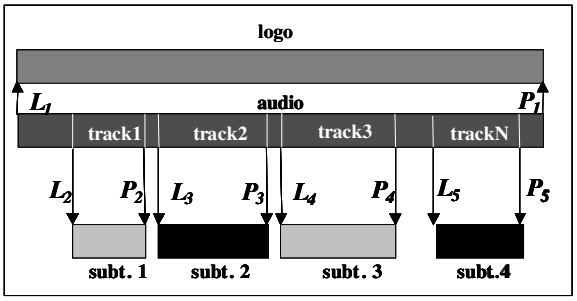


Figura 3:11 - Visão temporal de uma composição NCL.

Como pode ser observado na Figura 3:10, o vocabulário (linhas 2-9) de um *template* define tipos de componentes, através do elemento *component*. O tipo de componente (*type*) *song*, do tipo de conteúdo (*ctype*) áudio[24], é definido nas linhas 3-5; o tipo de componente *subtitle*, do tipo de conteúdo texto, é definido na linha 6; e o tipo de componente *logo*, do tipo de conteúdo imagem, é definido na linha 7. Cada componente de um dado tipo pode conter pontos de interface, declarados através do elemento *port* (porta). No exemplo (linha 4), uma instância do tipo *song* pode possuir âncoras do tipo *track*, marcando trechos de seu conteúdo. Conectores, por sua vez, são definidos através do elemento *connector* (linhas 8-9), e são usados no *template* para a criação de elos. Componentes, pontos de interface e conectores são referenciados, na segunda parte do *template*, por seus tipos (*type*) e podem definir sua cardinalidade (através dos atributos *minOccurs* e *maxOccurs*). Apesar de semanticamente igual a uma restrição, a definição de cardinalidade no vocabulário é um recurso da linguagem que visa facilitar a especificação dos *templates*.

A segunda parte do *template* define suas restrições (linhas 10-21). Uma restrição (elemento *constraint*) pode se referir a componentes, conectores e pontos

[24] XTemplate 2.1 altera os nomes dos atributos *label* e *type* de XTemplate 2.0 para *type* e *ctype*, respectivamente. Para facilitar a comparação entre *templates* especificados em XTemplate 2.0 e 2.1, este texto utiliza os nomes de atributos definidos por XTemplate 2.1.

de interface. Como mencionado, elas são adicionais às restrições quanto à cardinalidade e ao tipo, definidas no vocabulário. Na linha 12, por exemplo, tem-se a restrição de que a composição somente pode conter componentes dos tipos *song*, *logo* e *subtitles*.

Restrições são definidas através do atributo *select*. Esse atributo contém uma expressão na linguagem XPath (W3C, 1999c), que retorna um valor booleano. O atributo *description* pode ser utilizado para reportar uma mensagem de erro quando uma restrição não é satisfeita (ou seja, quando a expressão do atributo *select* retorna um valor falso).

No conjunto de restrições de um *template*, também podem ser declaradas instâncias de componentes, através do elemento *resource*. Instâncias de componentes devem referenciar um tipo de componente (atributo *type*) já declarado no vocabulário e a URI do conteúdo (atributo *src*), além de especificar uma identificação para a instância (atributo *label*). A linha 13 exemplifica uma instância do tipo de componente *logo*, com sua identificação definida como *logoJPG* e a URI de seu conteúdo especificada como "*logo.jpg*".

Instâncias de conectores (elos), fornecendo a semântica de um *template* de composição, também são definidas na segunda parte do *template* (restrições). Para a definição de elos (elementos *link*), devem ser referenciados os tipos de componentes e conectores declarados no vocabulário, ou instâncias de componentes declaradas nas restrições do *template*. Como os elos são criados a partir de referências a conectores, os elementos *link* (ver Capítulo 2) possuem elementos filhos do tipo *bind*, que relacionam os papéis do conector a componentes do *template*. Esses componentes são selecionados a partir de uma expressão XPath, no atributo *select* do elemento *bind*.

Duas instâncias de elos, neste texto nomeadas *L1* (linhas 14-17) e *P1* (linhas 18-21), são declaradas pelo *template*: *L1* é do tipo de conector *L* (linha 8), enquanto *P1* é do tipo de conector *P* (linha 9). Os elos *L1* e *P1* representam relacionamentos entre um áudio (a ser especificado por uma composição que venha a utilizar esse *template*[25]) e a imagem *logoJPG* (especificada pelo próprio *template*), onde o início do áudio irá provocar o início da apresentação da imagem (elo *L1*), e o término do áudio irá causar o término da apresentação da imagem

[25] A forma como composições referenciam *templates* será abordada no próximo capítulo.

(elo *P1*). Esse sincronismo entre o *logo* e o áudio é obtido pela referência aos conectores *L* e *P*, e pelos elementos *bind* dos dois elos - que relacionam os papéis *on_x_presentation_begin* e *on_x_presentation_end* dos conectores com a instância do tipo de componente *song*; e os papéis *start_y* e *stop_y* dos conectores com a imagem instanciada pelo próprio *template* (*logoJPG*).

É importante ressaltar que, na especificação dos relacionamentos (elos) do *template*, é possível utilizar instruções para declaração de variáveis (*variable*), instruções para realizar repetição (*for-each*) e outras funções especificadas pelo padrão XSLT (W3C, 1999d). Isso é exemplificado na especificação do sincronismo entre trechos do áudio (*track*) e suas legendas (*subtitle*). A definição desse sincronismo é similar à utilizada no exemplo de *template* que pode ser consultado em (Muchaluat-Saade, 2003). De maneira simplificada, os elos entre as faixas do áudio e suas legendas são especificados por um *loop* (*for-each*, linhas 22-32), que percorre os trechos do áudio em seqüência, sendo sua posição armazenada na variável *i*. Os *binds* são definidos referenciando o i-ésimo trecho e a i-ésima legenda.

Cabe comentar que, no exemplo, o *logo* foi definido como um recurso do próprio *template* para ilustrar essa facilidade da linguagem. Evidentemente, o *template* ilustrado na Figura 3:10 pode ser facilmente modificado para que tanto o áudio como o *logo* sejam definidos pelas composições que o referenciem, sendo apenas os elos de sincronização especificados pelo próprio *template*.

NCL 2.1 redefine a linguagem XTemplate usando uma abordagem modular, permitindo, assim, a definição de perfis de XTemplate. A Tabela 6 apresenta os elementos de XTemplate 2.1 (e seus respectivos módulos), estando em negrito os elementos adicionados ou alterados pela nova versão da linguagem. A estrutura básica para especificações de *templates* é definida no módulo *Structure* de XTemplate. O elemento *raiz* é chamado *xtemplate*, enquanto o cabeçalho e o corpo do *template* são chamados, respectivamente, de *head* e *body* (seguindo a terminologia adotada por outras linguagens padronizadas pelo W3C). O cabeçalho do *template* define seu vocabulário, restrições e os recursos que podem ser especificados sem o uso de XSLT (W3C, 1999d). O corpo de um *template* especifica recursos e relações entre componentes por meio de transformadas XSLT. Relações podem ser de inclusão (especificadas por transformadas declaradas como elementos filho de *body*) e por meio de conectores (especificadas

por transformadas agrupadas no elemento *linkBase* filho de *body*). Ou seja, a semântica dos relacionamentos de um *template* é definida no elemento *body*.

Módulo	**Elemento**	**Atributos**	**Conteúdo**
Structure	*xtemplate*		(***head***, ***body***?)
Structure	***head***		(*vocabulary*, ***constraints****, ***resources****)
BasicVocabulary	*vocabulary*		(*component*, *connector*)*
BasicVocabulary	*component*	*type*, *ctype*, *maxOccurs*, *minOccurs*	*port**, ***component****
BasicVocabulary	*port*	*type, maxOccurs, minOccurs*	
ConnectorVocabulary	*connector*	*type*, *src*, *maxOccurs*, *minOccurs*	
BasicConstraints	***constraints***		*constraint**
BasicConstraints	*constraint*	*select, description*	
BasicResources	***resources***		*resource**
BasicResources	*resource*	*src*, *type*, *label*	
Structure	***body***		(***linkBase*** | ***stylesheet*** [26])*
BasicLinking	***linkBase***	*select, description*	(*link* | ***stylesheet***)*
BasicLinking	*link*	*type*	*bind**
BasicLinking	*bind*	*role*, *select*	

Tabela 6 - Elementos da linguagem XTemplate de NCL 2.1.

O módulo *BasicVocabulary* define o elemento *vocabulary*[27], declarado no cabeçalho do *template*. Esse módulo também define os elementos *component* e

[26] O conteúdo *stylesheet* se refere ao uso da linguagem XSLT (W3C, 1999d) para especificação de relacionamentos em XTemplate 2.1, como será apresentado.

port. Diferente da versão anterior de XTemplate, é possível que um perfil XTempltate 2.1 estenda a especificação dos elementos *component*, permitindo que eles possuam outros elementos *component* como filhos - ou seja, é permitida a declaração de componentes compostos no vocabulário. O módulo *ConnectorVocabulary* define o elemento *connector*, que deve ser declarado como elemento filho do elemento *vocabulary* em perfis de XTemplate.

Ainda no cabeçalho, é possível definir restrições adicionais àquelas especificadas pela cardinalidade dos componentes e conectores, e pelo aninhamento de componentes. Isso é realizado pelo uso dos elementos definidos pelo módulo *BasicConstraints*: *constraint* (igual ao elemento homônimo da versão anterior) e *constraints* (que, na nova versão de XTemplate, possui apenas elementos *constraint* como filhos). O cabeçalho também pode conter o elemento *resources*, que agrupa declarações de instâncias de componentes do vocabulário, feitas pelo elemento *resource* - ambos especificados pelo módulo *BasicResources*.

O corpo de um *template* (elemento *body*) permite a especificação de relacionamentos de dois tipos: relacionamentos definidos por conectores (elos) e relacionamentos de inclusão (composições). Como esses relacionamentos podem ser especificados utilizando as funcionalidades do padrão XSLT, foi definido o módulo *TemplateXSLT*, que estende esse padrão (adicionando os elementos[28] *link*, *bind* e *resource*). O módulo *TemplateXSLT* é referenciado em XTemplate por meio de um *namespace* (W3C, 1999b), que neste texto será considerado *xsl*. Assim, as transformadas para definição de relacionamentos no corpo de um *template* são agrupadas em elementos *xsl:stylesheet* (W3C, 1999d).

Relacionamentos definidos por conectores devem ser especificados no elemento *linkBase*, através dos elementos *link* e seus elementos filhos do tipo *bind*, todos definidos pelo módulo *BasicLinking*. Adicionalmente, um perfil XTemplate pode definir o elemento *xsl:stylesheet* como filho de *linkBase*. Esse

---

[27] Os elementos especificados por XTemplate 2.1, a menos que dito o contrário, possuem a mesma semântica, os mesmos atributos e o mesmo conteúdo dos elementos homônimos de XTemplate 2.0.

[28] Esses elementos são usados de forma semelhante ao elemento *element* (definido por XSLT), sendo função do processador de *templates* (ver Capítulo 5) interpretá-los para gerar uma folha de estilo (*stylesheet*) no padrão XSL (W3C, 2001a).

elemento define uma transformada para especificação de elos (como ilustrado na Figura 3:12 - linhas 29-43), que deve utilizar os elementos *xsl:link* e *xsl:bind*.

De forma semelhante, para possibilitar a especificação de relacionamentos de inclusão no *template*, um perfil de XTemplate deve definir o elemento *xsl:stylesheet* (do módulo *TemplateXSLT*) como filho do elemento *body*. Esse elemento *xsl:stylesheet* especifica uma transformada a ser aplicada diretamente ao conteúdo da composição que herdar a especificação do *template* e deve utilizar o elemento *xsl:resource* para declarar novos recursos[29].

Na Figura 3:12, apresenta-se o mesmo *template* da Figura 3:10, estruturado segundo a nova linguagem XTemplate. Como esse *template* não declara componentes compostos e nem define relações de inclusão, ele pode ser especificado sem a declaração do elemento *xsl:stylesheet* como filho do elemento *body*; e sem permitir que o elemento *component* possua elementos filhos do tipo *component*. Por esse motivo, o *template* da Figura 3:12 pode ser definido em um perfil de XTemplate 2.1 que equivale à linguagem XTemplate 2.0. Na figura, o elemento *vocabulary* foi declarado no cabeçalho, assim como a restrição (elemento *constraint*) quanto ao número de componentes e o recurso (elemento *resource*) *logoJPG*. No corpo do *template*, os *links* foram agrupados no elemento *linkBase* e o *loop* (*for-each*) foi definido no elemento *xsl:stylesheet*, conteúdo também de *linkBase*.

```
01  <xtemplate id="audio-with-subtitles21"... >
02  <head>
03   <vocabulary>
04    <component type="song" ctype="audio" maxOccurs="1" minOccurs="1">
05     <port type="track" minOccurs="1" maxOccurs="unbounded" />
06    </component>
07    <component type="subtitle" ctype="text" maxOccurs="unbounded" />
08    <component type="logo" ctype="img" maxOccurs="1" />
09    <connector src="starts.xml" type="L" maxOccurs="unbounded" />
10    <connector src="finishes.xml" type="P" maxOccurs="unbounded" />
11   </vocabulary>
12   <constraints>
13    <constraint  select="count(child::*[@type!='song']  |  child::*[@type!='logo']
|    child::*[@type!='subtitle'])    =    (count(child::*)-count(child::linkBase))"
description="All components must be songs or logos or subtitles."/>
14   </constraints>
15   <resources>
16    <resource src="logo.jpg" type="logo" label="logoJPG" />
```

[29] Relacionamentos de inclusão serão exemplificados no final desta seção.

```
17   </resources>
18  </head>
19  <body>
20   <linkBase>
21    <link type="L">
22     <bind role="on_x_presentation_begin" select="child::*[@label='audio']"/>
23     <bind role="start_y" select="child::*[@label='logoJPG']"/>
24    </link>
25    <link type="P">
26     <bind role="on_x_presentation_end" select="child::*[@label='audio']"/>
27     <bind role="stop_y" select="child::*[@label='logoJPG']"/>
28    </link>
29    <xsl:stylesheet>
30     <xsl:template>
31      <xsl:for-each select="child::*[@type='audio']/child::*[@type='track']" >
32       <xsl:variable name="i" select="position()"/>
33        <xsl:link type="L">
34         <xsl:bind role="on_x_presentation_begin" select="current()" />
35         <xsl:bind role="start_y" select="//*/child::*[@type='subtitle'][$i]"/>
36        </xsl:link>
37        <xsl:link type="P">
38         <xsl:bind role="on_x_presentation_end" select="current()" />
39         <xsl:bind role="stop_y" select="//*/child::*[@type='subtitle'][$i]"/>
40        </xsl:link>
41      </xsl:for-each>
42     </xsl:template>
43    </xsl:stylesheet>
44   </linkBase>
45  </body>
46  </xtemplate>
```

Figura 3:12 - Exemplo de *template* de composição em NCL 2.1.

O exemplo da Figura 3:12 demonstra como um *template* especificado em XTemplate 2.0 pode ser convertido para uma especificação em XTemplate 2.1. Para demonstrar as novas facilidades introduzidas pela nova versão de XTemplate, é utilizada uma variação do exemplo anterior de *template*, contendo a especificação de componentes compostos e de relações de inclusão. Composições que herdam desse *template* devem declarar um áudio (com suas faixas) e um par de legendas para cada trecho do áudio, sendo uma em português e outra em inglês. A especificação do *template* define que cada par de legendas seja incluído em elementos *switch* (sendo a escolha entre alternativas baseada na língua do contexto de apresentação), e que cada *switch* seja sincronizado com o respectivo trecho do áudio. A Figura 3:13 e a Figura 3:14 ilustram, respectivamente, a visão temporal de uma composição que herde desse *template* e a especificação do *template*.

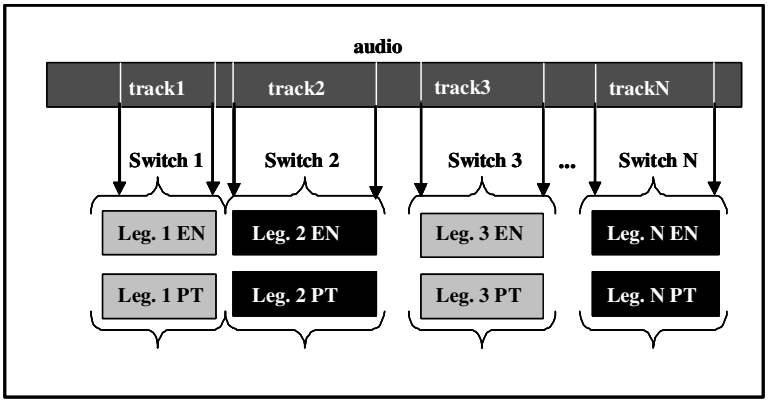


Figura 3:13 - Visão temporal de uma composição NCL.

```
<xtemplate id="audioWithSubtitlesEnPt21" >
<head>
<vocabulary>
 <component type="song" ctype="audio" maxOccurs="1" minOccurs="1">
  <port type="track" minOccurs="1" maxOccurs="unbounded" />
 </component>
 <component type="switch" ctype="switch">
  <component type="subtitleEn" ctype="text" maxOccurs="1" minOccurs="1" />
  <component type="subtitlePt" ctype="text" maxOccurs="1" minOccurs="1" />
 </component >
 <connector src="starts.xml" type="L" maxOccurs="unbounded" />
 <connector src="finishes.xml" type="P" maxOccurs="unbounded" />
</vocabulary>
</head>
<body>
<xsl:stylesheet>
 <xsl:template match="/*/*/*">
  <xsl:if test="not(./@type='subtitleEn') and not(./@type='subtitlePt')">
   <xsl:copy-of select="." />
  </xsl:if>
 </xsl:template>
 <xsl:for-each select="//*[@type='subtitleEn']" >
  <xsl:variable name="i" select="position()"/>
  <xsl:resource type="switch">
   <xsl:attribute name="id">Switch<xsl:value-of select="$i"/></xsl:attribute>
   <xsl:copy>
    <xsl:for-each select="text()|@*">
     <xsl:copy/>
    </xsl:for-each>
    <xsl:apply-templates/>
   </xsl:copy>
   <xsl:call-template name="bindRuleElement">
    <xsl:with-param name="component" select="./@id"></xsl:with-param>
    <xsl:with-param name="rule">RuleEnUS</xsl:with-param>
   </xsl:call-template>
   <xsl:call-template name="ptSwitchElement">
    <xsl:with-param name="i" select="$i"></xsl:with-param>
```

```
38     </xsl:call-template>
39    </xsl:resource>
40   </xsl:for-each>
41   <xsl:template name="ptSwitchElement">
42    <xsl:param name="i"></xsl:param>
43    <xsl:for-each select="/*/*/child::*[@type='subtitlePt'][$i]" >
44     <xsl:copy>
45      <xsl:for-each select="text()|@*">
46       <xsl:copy/>
47      </xsl:for-each>
48      <xsl:apply-templates/>
49     </xsl:copy>
50     <xsl:call-template name="bindRuleElement">
51      <xsl:with-param name="component" select="./@id"></xsl:with-param>
52      <xsl:with-param name="rule">RuleEnPT</xsl:with-param>
53     </xsl:call-template>
54    </xsl:for-each>
55   </xsl:template>
56   <xsl:template name="bindRuleElement">
57    <xsl:param name="component"></xsl:param>
58    <xsl:param name="rule"></xsl:param>
59    <xsl:element name="bindRule">
60      <xsl:attribute name="component"><xsl:value-of
select="$component"/></xsl:attribute>
61      <xsl:attribute name="rule"><xsl:value-of select="$rule"/></xsl:attribute>
62    </xsl:element>
63   </xsl:template>
64   </xsl:stylesheet>
65   <linkBase>
66    <xsl:stylesheet>
67    <xsl:template>
68     <xsl:for-each select="child::*[@type='song']/child::*[@type='track']" >
69      <xsl:variable name="i" select="position()"/>
70      <xsl:link type="L">
71       <xsl:bind role="on_x_presentation_begin" select="current()" />
72       <xsl:bind role="start_y" select="//*[@type='switch'][$i]"/>
73      </xsl:link>
74      <xsl:link type="P">
75       <xsl:bind role="on_x_presentation_end" select="current()" />
76       <xsl:bind role="stop_y" select="//*[@type='switch'][$i]"/>
77      </xsl:link>
78     </xsl:for-each>
79    </xsl:template>
80    </xsl:stylesheet>
81   </linkBase>
82  </body>
83  </xtemplate>
```

Figura 3:14 - Exemplo de *template* com relações de inclusão e por conectores em NCL 2.1.

No vocabulário do *template* da Figura 3:14 já pode ser observada a primeira diferença entre as versões de XTemplate, pois existe a definição de um componente composto (linhas 7-10). Esse componente, do tipo *switch*, deve conter um elemento de texto do tipo *subtitleEn* (legenda em inglês) e um elemento de texto do tipo *subtitlePt* (legenda em português). Os outros elementos do vocabulário são idênticos aos do exemplo anterior.

No corpo do *template* temos um elemento *xsl:stylesheet* (linhas 16-64), filho do elemento *body*, que especifica uma transformada a ser aplicada diretamente aos elementos da composição que herdar desse *template*. Essa transformada irá definir as relações de inclusão do *template*, representando, portanto, uma nova funcionalidade de XTemplate 2.1. As linhas 17-21 especificam que os elementos dos tipos *subtitleEn* e *subtitlePt* não devem permanecer como filhos diretos da composição (como será visto no próximo parágrafo, esses tipos de elementos serão definidos como filhos de outros elementos - *switches* - contidos pela composição).

O *loop* (*for-each*) das linhas 22-40 define uma iteração pelos elementos do tipo *subtitleEn*. Para cada elemento desse tipo, é criado um recurso do tipo *switch* (linhas 24-39) a ser adicionado à composição que herdar desse *template*. Na linha 25 é definido o atribuo *id* do i-ésimo *switch*, e as linhas 26-31 adicionam o elemento do tipo *subtitleEn*, referente ao passo *i* da iteração, ao *switch i*. Em seguida, uma chamada ao *xsl:template* "*bindRuleElement*", definido nas linhas 56-63, adiciona um elemento *bindRule* ao *switch* - com seu atributo *component* referenciando a i-ésima legenda em inglês e com a regra (*rule*) associada a esse componente sendo *RuleEnUS*. O valor desses atributos foi passado como parâmetro para o *xsl:template bindRuleElement* (W3C, 1999d). Em seguida, as linhas 36-38 efetuam uma chamada ao *xsl:template* "*ptSwitchElement*", definido nas linhas 41-55. Esse *xsl:template* adiciona a i-ésima legenda em português ao *switch*, e, também por meio de uma chamada (linhas 50-53) ao *xsl:template* "*bindRuleElement*", adiciona um elemento *bindRule* ao *switch* associando essa legenda à regra *RuleEnPT*[30]. Finalmente, como no exemplo anterior, o elemento *linkBase* (linhas 65-81) do *template* define uma transformada com os

[30] As regras de apresentação *RuleEnUS* e *RuleEnPT* devem ser especificadas, com a semântica descrita, pelo documento NCL que utilize o *template*.

relacionamentos por conectores, sincronizando cada trecho do áudio com o respectivo *switch*.

4
X-SMIL

Visando aumentar o reuso e a expressividade da linguagem SMIL (W3C, 2001b), descrita no Capítulo 2, este capítulo apresenta a linguagem X-SMIL. X-SMIL é a combinação de duas extensões à SMIL (conforme citado no Capítulo 1): XT-SMIL, que introduz o conceito de *templates* de composição; e XC-SMIL, que permite a definição de elos por meio do reuso de conectores hipermídia.

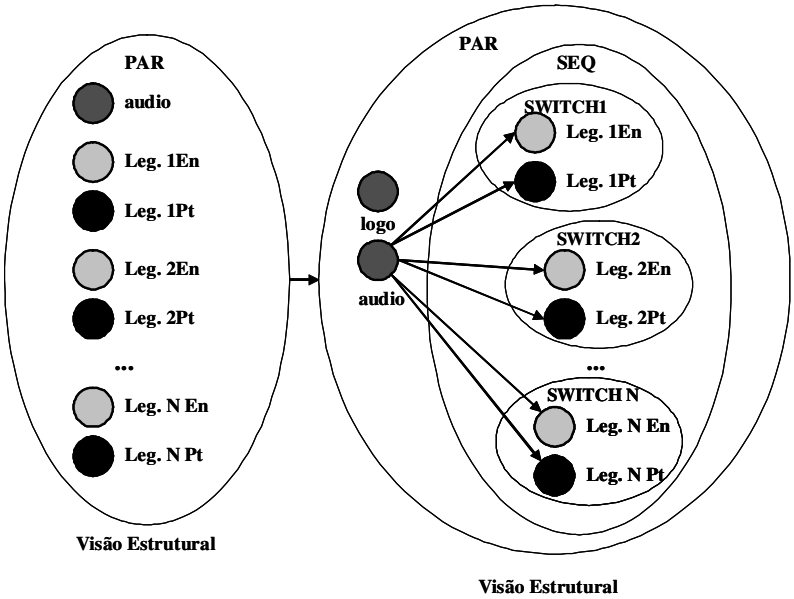
Este capítulo descreve, inicialmente, a extensão XT-SMIL. Em seguida, é definida a linguagem XC-SMIL e, como conseqüência, apresenta-se X-SMIL.

4.1.
XT-SMIL: SMIL + XTemplate

O capítulo anterior definiu a linguagem XTemplate 2.1 para especificação de *templates* de composição. XTemplate é estruturada em módulos, podendo, dessa forma, definir perfis de linguagem. Para incorporar o conceito de *templates* em SMIL, gerando a linguagem XT-SMIL (Silva et al., 2004b), é utilizado um perfil de XTemplate 2.1 que reúne todos os módulos dessa linguagem, exceto *ConnectorVocabulary* e *BasicLinking*. Esse perfil XTemplate, independente de conectores, permite, entre outras funcionalidades, a definição de componentes compostos e de relações de inclusão. A fim de permitir a utilização de *templates* em documentos XT-SMIL, adotou-se uma abordagem similar ao uso do módulo *XTemplateUse* em NCL, sendo adicionado o atributo *xtemplate* às composições *seq*, *par* e *excl*, e o atributo *type* aos elementos que podem estar contidos nessas composições. O uso desses atributos é ilustrado no final desta seção.

As principais funcionalidades adicionadas por XT-SMIL serão analisadas a partir de um exemplo de *template*: *audioComLegendasEnPt*. Esse *template* define

uma nova semântica[31] para composições XT-SMIL *par* que contenham um áudio e pares de legendas (do tipo texto) para cada trecho do áudio, sendo uma legenda em português e outra em inglês. Quando uma composição XT-SMIL paralela, como a exemplificada à esquerda da Figura 4:1, referencia esse *template*, ela ganha outra semântica, herdando todas as definições da configuração do *template*. O *template* define que seus pares de legendas serão agrupados em elementos do tipo *switch*, que serão sincronizados com as faixas do áudio. Esse sincronismo será obtido incluindo os *switches* em um container *seq*, e relacionando o término da apresentação de cada *switch* (ou seja, de suas legendas internas) com o término do trecho de áudio correspondente. Uma imagem (*logo*) será também adicionada à composição e exibida em paralelo com o áudio, devido à semântica tradicional da composição *par*. A visão estrutural da nova composição XT-SMIL gerada após o processamento do *template*[32] *audioComLegendasEnPt* é ilustrada na parte direita da Figura 4:1, onde é possível observar que o *template* definiu novas relações de inclusão. A visão temporal da mesma composição, após seu processamento, é apresentada na Figura 4:2.



[31] As composições XT-SMIL *par*, *seq* e *excl* possuem a mesma semântica temporal que as composições homônimas em SMIL. Porém, quando referenciam *templates* XT-SMIL, essas composições podem adquirir novas semânticas que estendam suas semânticas originais.

[32] O mecanismo de processamentos de *templates* e a geração das composições processadas serão discutidos no próximo capítulo.

Figura 4:1 - Visão estrutural de uma composição XT-SMIL *par* antes e após o processamento de *template*.

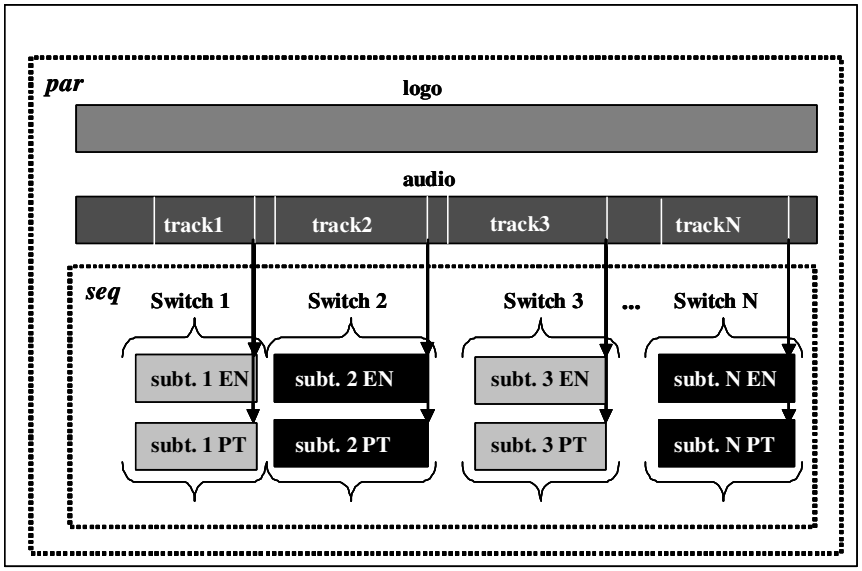


Figura 4:2 - Visão temporal de uma composição *par* em SMIL.

A especificação do *template audioComLegendasEnPt* está ilustrada na Figura 4:3. Em primeiro lugar, como pode ser verificado nas linhas 8 a 13 da Figura 4:3, existe a declaração de um componente composto. Esse componente é do tipo *seq*, que pode possuir uma quantidade ilimitada de componentes do tipo *switch*, sendo que cada *switch* possui, exatamente, dois componentes: um do tipo *subtitleEn* e outro do tipo *subtitlePt*. Esse aninhamento de componentes, como visto na Seção 3.5, é uma nova facilidade oferecida por XTemplate 2.1. A definição de componentes compostos no vocabulário do *template* permite que, além de restrições sobre cardinalidade e tipo, sejam especificadas restrições sobre o aninhamento de componentes. Na Figura 4:3, por exemplo, foram especificados os tipos dos componentes que podem estar contidos nos componentes de tipo *seq* e *switch*.

```
01  <xtemplate>
02  <head>
03   <vocabulary>
04    <component type="song" ctype="audio" maxOccurs="1" minOccurs="1">
05     <port type="track" maxOccurs="unbounded" />
06    </component>
07    <component type="logo" ctype="img" maxOccurs="1" />
08    <component type="seq" ctype="seq">
09     <component type="switch" ctype="switch" maxOccurs="unbounded">
10      <component type="subtitleEn" ctype="text" maxOccurs="1" minOccurs="1" />
11      <component type="subtitlePt" ctype="text" maxOccurs="1" minOccurs="1" />
12     </component>
```

```
13    </component>
14   </vocabulary>
15   <constraints>
16    <constraint select="count(//*[@type='track']) =
count(//*[@type='subtitleEn'])" description="The number of tracks must be equal to
the number of subtitleEn"/>
17    <constraint select="count(//*[@type='track']) =
count(//*[@type='subtitlePt'])" description="The number of tracks must be equal to
the number of subtitlePt"/>
18   </constraints>
19   <resources>
20    <resource src="logo.jpg" type="logo" label="logoJPG" />
21   </resources>
22  </head>
23  <body>
24   <xsl:stylesheet>
25    <xsl:template match="/*/*/*">
26     <xsl:if test="not(./@type='subtitleEn') and not(./@type='subtitlePt')">
27      <xsl:copy-of select="." />
28     </xsl:if>
29    </xsl:template>
30    <xsl:resource type="seq" >
31     <xsl:for-each select="child::*[@ type ='subtitleEn']" >
32      <xsl:variable name="i" select="position()"/>
33      <xsl:resource type="switch">
34      <xsl:attribute name="id">switch<xsl:value-of select="$i"/></xsl:attribute>
35      <xsl:copy>
36       <xsl:for-each select="text()|@*">
37         <xsl:copy/>
38       </xsl:for-each>
39       <xsl:attribute name = "systemLanguage">en-us</xsl:attribute>
40       <xsl:attribute name="end">
41        <xsl:value-of
select="//*/child::*[@type='song']/child::*[@type='track'][$i]/@id"/>.end
42       </xsl:attribute>
43       <xsl:apply-templates/>
44      </xsl:copy>
45      <xsl:call-template name="ptSwitchElement">
46        <xsl:with-param name="i" select="$i" />
47      </xsl:call-template>
48      </xsl:resource>
49     </xsl:for-each>
50    </xsl:resource>
51    <xsl:template name="ptSwitchElement">
52     <xsl:param name="i"></xsl:param>
53     <xsl:for-each select="/*/*/child::*[@ type ='subtitlePt'][$i]" >
54      <xsl:copy>
55       <xsl:for-each select="text()|@*">
56        <xsl:copy/>
57       </xsl:for-each>
58       <xsl:attribute name = "systemLanguage">pt-br</xsl:attribute>
```

```
59       <xsl:attribute name="end">
60        <xsl:value-of
select="//*/child::*[@type='song']/child::*[@type='track'][$i]/@id"/>.end
61       </xsl:attribute>
62       <xsl:apply-templates/>
63      </xsl:copy>
64     </xsl:for-each>
65    </xsl:template>
66   </xsl:stylesheet>
67  </body>
68  </xtemplate>
```

Figura 4:3 - *Template audioComLegendasEnPt* em XT-SMIL.

As linhas 16 e 17 definem duas restrições adicionais às restrições definidas no vocabulário: o número de componentes do tipo *subtitleEn* e do tipo *subtitlePt* devem ser iguais ao número de pontos de interface do tipo *track*. A linha 20 define uma instância do componente *logo*, sendo seu identificador (*label*) *logoJPG*.

As linhas 24-66 demonstram uma outra facilidade de XTemplate 2.1: a possibilidade de aplicar uma transformada XSLT diretamente aos elementos de uma composição, pelo uso do elemento *xsl:stylesheet* como filho direto do elemento *body* (o que possibilita, por exemplo, a definição de relações de inclusão). A explicação dessa transformada será realizada por partes, iniciando pelas linhas 25-29. Essas linhas especificam que os elementos do tipo *subtitleEn* e *subtitlePt* deixam de ser filhos diretos da composição. Como será visto a seguir, elementos desses tipos passam a ser contidos *recursivamente* pela mesma (ou seja, eles são definidos como filhos de outros componentes da composição). Esse trecho do *template* declara, portanto, que todos os componentes de uma composição, à exceção dos de tipo *subtitleEn* e *subtitlePt*, devem ser mantidos como seus filhos.

As linhas 30-50 definem a criação de um elemento *seq*. O elemento *seq* é declarado como um recurso de *TemplateXSLT* (elemento *xsl:resource* referenciando o tipo de componente *seq* do vocabulário) e tem como conteúdo elementos do tipo *switch*. Mais especificamente, as linhas 31-49 definem uma iteração sobre os componentes do tipo *subtitleEn* para definição dos *switches* que compõem o elemento *seq*. Os *switches* são definidos nas linhas 33-48, referenciando o tipo de componente *switch* do vocabulário.

O valor do atributo *id* de cada *switch* é definido na linha 34. As linhas 35-47 contêm a definição do conteúdo de cada elemento *switch*, feita através da iteração

sobre os elementos do tipo texto definidos na composição. Primeiramente, define-se o elemento *subtitleEn*, referente ao passo *i* da iteração, como conteúdo do *switch i* (linhas 35-44). Em seguida, o mesmo ocorre para o elemento *subtitlePt*, através de uma chamada (linhas 45-47) ao *xsl:template* (W3C, 1999d) *templatePtSwitchElement*, definido nas linhas 51-65.

As linhas 39 e 58 definem, respectivamente, o atributo *systemLanguage* para os elementos de tipo *subtitleEn* e *subtitlePt* como sendo *en-us* (inglês americano) e *pt-br* (português brasileiro); ou seja, a escolha entre as alternativas para os componentes (textos) do *switch* deve ser baseada no idioma do contexto de exibição. As linhas 40-42 e 59-61 especificam a sincronização das legendas com as faixas de áudio. Para isso, é incluído um atributo *end* nos elementos de texto representando legendas. Esse atributo determina que cada legenda termine juntamente com a i-ésima faixa de áudio. O início da legenda seguinte é obtido pela semântica do elemento *seq*, que contém todas as legendas. Note que a inclusão de atributos em elementos, como os atributos *systemLanguage* e *end*, é, também, uma nova facilidade introduzida por XTemplate (por possibilitar que folhas de estilo sejam aplicadas diretamente aos nós de uma composição).

A Figura 4:4 ilustra uma composição XT-SMIL *par*, que referencia o *template audioComLegendasEnPt* da Figura 3:2 através do atributo *xtemplate*. Os tipos de seus elementos filho são declarados através do atributo *type*. Quando um documento contendo a composição da Figura 4:4 tem seus *templates* processados (o processamento de *templates* é apresentado em detalhes no próximo capítulo), o resultado é um documento contendo a composição XT-SMIL *par* da Figura 4:5. Note que uma composição XT-SMIL, quando tem seu *template* processado, torna-se uma composição válida, também, em SMIL.

```
01  <par id="coisaPele" xtemplate="audioComLegendasEnPt.xml">
02   <audio type="song" region="r0" id="samba" src="coisadepele.wav">
03    <area id="part1" type ="track" begin="8.4s" end="18s"/>
04    <area id="part2" type ="track" begin="18.5s" end="28s"/>
05    <area id="part3" type ="track" begin="29s" end="39s"/>
06   </audio>
07   <text type ="subtitleEn" region="r1" id="lyrics1a" src="versos01en.html"/>
08   <text type ="subtitlePt" region="r1" id="lyrics1b" src="versos01pt.html"/>
09   <text type ="subtitleEn" region="r1" id="lyrics2a" src="versos02en.html"/>
10   <text type ="subtitlePt" region="r1" id="lyrics2b" src="versos02pt.html"/>
11   <text type ="subtitleEn" region="r1" id="lyrics3a" src="versos03en.html"/>
```

```
12   <text type ="subtitlePt" region="r1" id="lyrics3b" src="versos03pt.html"/>
13  </par>
```

Figura 4:4 - Composição XT-SMIL *par* utilizando um *template*.

```
01  <par id="coisaPele">
02   <img id="logoResource0" src="logoTele.jpg"/>
03   <audio region="r0" id="samba" src="coisadepele.wav">
04    <area begin="8.4s" end="18s" id="part1"/>
05    <area begin="18.5s" end="28s" id="part2"/>
06    <area begin="29s" end="39s" id="part3"/>
07   </audio>
08   <seq>
09    <switch id="switch1">
10     <text   region="r1"   end="part1.end"   id="lyrics1a"   src="versos01en.html"
systemLanguage="en-us"/>
11     <text   region="r1"   end="part1.end"   id="lyrics1b"   src="versos01pt.html"
systemLanguage="pt-br"/>
12    </switch>
13    <switch id="switch2">
14     <text   region="r1"   end="part2.end"   id="lyrics2a"   src="versos02en.html"
systemLanguage="en-us"/>
15     <text   region="r1"   end="part2.end"   id="lyrics2b"   src="versos02pt.html"
systemLanguage="pt-br"/>
16    </switch>
17    <switch id="switch3">
18     <text   region="r1"   end="part3.end"   id="lyrics3a"   src="versos03en.html"
systemLanguage="en-us"/>
19     <text   region="r1"   end="part3.end"   id="lyrics3b"   src="versos03pt.html"
systemLanguage="pt-br"/>
20    </switch>
21   </seq>
22  </par>
```

Figura 4:5 - Resultado do processamento de *template* em uma composição XT-SMIL *par*.

4.2. SMIL + XConnector (XC-SMIL) e X-SMIL

SMIL 2.0 (W3C, 2001b) somente permite a especificação de elos ponto-a-ponto, que podem ser disparados por eventos temporais predefinidos pela linguagem (ver Capítulo 2). Ao permitir o uso de eventos na especificação temporal de um documento, SMIL 2.0 oferece, em relação à sua versão anterior (W3C, 1998b), uma maior flexibilidade para a autoria dos documentos. Entretanto, quando comparada à NCL, as possibilidades de especificação de elos em SMIL são limitadas. NCL oferece elos multiponto, representando relações com semântica causal ou de restrição, de acordo com o conector usado pelo elo.

Além disso, uma mesma relação causal, por exemplo, pode relacionar eventos de diversos tipos, além dos tradicionais eventos de seleção e apresentação, contemplados por SMIL (ver Capítulo 3).

O autor de documentos SMIL deve mesclar o uso das composições temporais (*par*, *seq* e *excl*) com o uso de elos para especificar relacionamentos que envolvem a ocorrência de vários tipos de evento. Em NCL, uma relação, por mais complexa que seja, é representada por um único conector[33] (Muchaluat-Saade, 2003); e elos NCL, simples ou complexos, são sempre especificados da mesma forma: referenciando conectores.

Para aumentar a expressividade e o reuso da linguagem SMIL, é proposta a extensão XC-SMIL, que introduz o conceito de conectores hipermídia àquela linguagem. A linguagem XC-SMIL é formada pela adição do módulo *Linking* de NCL à linguagem SMIL. Assim, essa extensão de SMIL adiciona o elemento *linkBase* aos elementos *body*, *par*, *seq* e *excl*, permitindo a definição de bases de elos. Elementos *linkBase*, assim como em NCL, possuem elementos filhos do tipo *link*, para definição de elos referenciando conectores. Cada *link* possui um conjunto de elementos *bind*, que relaciona papéis do conector a componentes do documento XC-SMIL (pelos atributos *role* e *component*).

A Figura 4:6 ilustra uma composição *par* em SMIL, com um elo relacionando a imagem *img1* com o áudio *audio1*. O atributo *end* declarado pela imagem determina que seu término deve coincidir com o valor desse atributo, ou seja, deve ser junto com o término da apresentação de *audio1* (evento de término de apresentação - "*end*" - do componente *audio1*: "*audio1.end*")

```
01. <par>
02.   <video id="video1" />
03.   <audio id="audio1" />
04.   <img id="img1" end="audio1.end" />
05. </par>
```

Figura 4:6 - Exemplo de elo em uma composição SMIL.

A Figura 4:7 ilustra a definição do elo "*link1*" em XC-SMIL, que referencia o conector "*finishes.xml*". Esse elo é semanticamente igual ao elo entre *audio1* e *img1* da Figura 4:6. Elos multiponto, em XC-SMIL, são especificados de forma semelhante ao "*link1*", como apresentados na Figura 4:8. Nessa figura, o elo

[33] Obviamente, algumas relações podem necessitar de mais de um conector para serem especificadas em XConnector, como relações que envolvem tanto causalidade quanto restrição.

"*link2*" determina que o término do áudio deve ocasionar o término da exibição tanto do vídeo quanto da imagem. Utilizando a mesma estrutura, mas redefinindo o conector sendo usado (assim como os papéis de cada componente), elos representando qualquer relacionamento entre o áudio, a imagem e o vídeo podem ser definidos em XC-SMIL. Isso também é *representado* na Figura 4:8 pelo elo fictício "*link3*" (referenciando o conector "*xxx.xml*") e pela associação dos papéis *a*, *b* e *c* do conector " *xxx.xml* " com os componentes da composição XC-SMIL.

```
01. <par>
02.   <video id="video1" />
03.   <audio id="audio1" />
04.   <img id="img1 />
05.   <linkBase>
06.     <link id="link1" xconnector="finishes.xml">
07.       <bind component="audio1" role="on_x_presentation_end"/>
08.       <bind component="img1" role="stop_y"/>
09.     </link>
10.   </linkBase>
11. </par>
```

Figura 4:7 - Exemplo de elo em uma composição XC-SMIL.

```
01. <par>
02.   <video id="video1" />
03.   <audio id="audio1" />
04.   <img id="img1 />
05.   <linkBase>
06.     <link id="link2" xconnector="finishes.xml">
07.       <bind component="audio1" role="on_x_presentation_end"/>
08.       <bind component="img1" role="stop_y"/>
09.       <bind component="video1" role="stop_y"/>
10.     </link>
11.     <link id="link3" xconnector="xxx.xml">
12.       <bind component="audio1" role="a"/>
13.       <bind component="img1" role="b"/>
14.       <bind component="video1" role="c"/>
15.     </link>
16.   </linkBase>
17. </par>
```

Figura 4:8 - Exemplo de elos multiponto em uma composição XC-SMIL.

Combinando-se o perfil XT-SMIL com XC-SMIL, obtém-se a linguagem X-SMIL. Nessa linguagem, são possíveis a definição de elos pelo reuso de conectores e o uso de *templates* de composição para especificar a semântica de uma composição. Ao permitir o uso de conectores, X-SMIL utiliza o perfil completo de XTemplate. Assim, as semânticas temporais obtidas pelo uso das composições *par*, *seq* e *excl* em SMIL podem, em X-SMIL, ser obtidas por

templates de composição que usam conectores para especificar essas semânticas. Por esse motivo, em X-SMIL, é desencorajado o uso das composições *par*, *seq* e *excl*, sendo sugerido o uso do elemento *composite* (adicionado à X-SMIL utilizando o módulo *BasicComposite* de NCL), que deve ser usado no lugar dessas composições. Para obter a semântica temporal oferecida anteriormente pelas composições SMIL, deve-se utilizar os *templates* pré-definidos por X-SMIL: *templatePar*, *templateSeq*, *templateExcl*[34]. Um exemplo de uma composição X-SMIL, similar à composição da Figura 4:7, é ilustrado na Figura 4:9.

Em X-SMIL, assim com em NCL 2.1, pode-se definir *templates* de composição utilizando todas as facilidades de XTemplate 2.1. Ou seja, é possível definir tanto relações de inclusão (como exemplificado na seção anterior) quanto relações por meio de conectores (como exemplificado no Capítulo 3). Compiladores para documentos nessas linguagens, assim como o processador de *templates* de XTemplate 2.1, serão descritos no próximo capítulo.

```
01.<composition xtemplate="templatePar" >
02.   <video id="video1" />
03.   <audio id="audio1" />
04.   <img id="img1 />
05.   <linkBase>
06.     <link id="link2" xconnector="finishes.xml">
07.       <bind component="audio1" role="on_x_presentation_end"/>
08.       <bind component="img1" role="stop_y"/>
09.     </link>
10.   </linkBase>
11.</composition>
```

Figura 4:9 - Exemplo de uma composição X-SMIL.

[34] Para refletir o uso do atributo *endsync* (W3C, 2001b), definido em SMIL para composições *par* e *excl*, também foram especificados os *templates templateParFirst*, *templateParLast*, *templateParAll*, *templateExclFirst*, *templateExclLast* e *templateExclAll*, que correspondem, respectivamente, aos valores *first*, *last* e *all* que esse atributo pode possuir. Assim, pela semântica definida por SMIL, os *templates templatePar* e *templateExcl* equivalem, respectivamente, aos t*emplates templateParLast* e *templateExclLast*.

templates de composição que usam conectores para especificar essas semânticas. Por esse motivo, em X-SMIL, é desencorajado o uso das composições *par*, *seq* e *excl*, sendo sugerido o uso do elemento *composite* (adicionado à X-SMIL utilizando o módulo *BasicComposite* de NCL), que deve ser usado no lugar dessas composições. Para obter a semântica temporal oferecida anteriormente pelas composições SMIL, deve-se utilizar os *templates* pré-definidos por X-SMIL: *templatePar*, *templateSeq*, *templateExcl*[34]. Um exemplo de uma composição X-SMIL, similar à composição da Figura 4:7, é ilustrado na Figura 4:9.

Em X-SMIL, assim com em NCL 2.1, pode-se definir *templates* de composição utilizando todas as facilidades de XTemplate 2.1. Ou seja, é possível definir tanto relações de inclusão (como exemplificado na seção anterior) quanto relações por meio de conectores (como exemplificado no Capítulo 3). Compiladores para documentos nessas linguagens, assim como o processador de *templates* de XTemplate 2.1, serão descritos no próximo capítulo.

```
01.<composition xtemplate="templatePar" >
02.   <video id="video1" />
03.   <audio id="audio1" />
04.   <img id="img1 />
05.   <linkBase>
06.     <link id="link2" xconnector="finishes.xml">
07.       <bind component="audio1" role="on_x_presentation_end"/>
08.       <bind component="img1" role="stop_y"/>
09.     </link>
10.   </linkBase>
11.</composition>
```

Figura 4:9 - Exemplo de uma composição X-SMIL.

[34] Para refletir o uso do atributo *endsync* (W3C, 2001b), definido em SMIL para composições *par* e *excl*, também foram especificados os *templates templateParFirst*, *templateParLast*, *templateParAll*, *templateExclFirst*, *templateExclLast* e *templateExclAll*, que correspondem, respectivamente, aos valores *first*, *last* e *all* que esse atributo pode possuir. Assim, pela semântica definida por SMIL, os *templates templatePar* e *templateExcl* equivalem, respectivamente, aos t*emplates templateParLast* e *templateExclLast*.

5
Framework para Compiladores

Um dos princípios adotados pelo W3C para a criação de linguagens de marcação baseadas em XML sugere uma abordagem modular. Módulos agrupam, de forma coerente, elementos e atributos XML que possuam alguma relação semântica entre si. SMIL (W3C, 2001b) e XHTML (W3C, 2000c) são exemplos de linguagens W3C que seguem esse princípio. As vantagens obtidas pelo uso de uma abordagem modular são possíveis devido às características de XML e de tecnologias relacionadas (W3C, 2001d; W3C, 1999b).

Documentos especificados de acordo com a definição de uma linguagem modular baseada em XML devem ser processados para que suas descrições possam ser utilizadas em um determinado aplicativo. Muitas vezes, diversos processadores de documentos são implementados, por aplicativos diferentes, para uma mesma linguagem. Por exemplo, há um processador de documentos SMIL para cada *player* SMIL.

No entanto, quando processadores analisam documentos de uma mesma linguagem de origem, várias funcionalidades podem ser reaproveitadas e implementadas uma única vez. A identificação dessas funcionalidades deu início ao desenvolvimento do trabalho apresentado neste capítulo. Além das características comuns aos processadores de uma mesma linguagem, observou-se, em outro nível de abstração, aspectos comuns que podem ser reutilizados em processadores de qualquer linguagem modular baseada em XML. A partir dessas observações, uma estrutura em dois níveis para especificação de *frameworks* para implementação de processadores (ou compiladores) é proposta, como se segue.

Em um primeiro nível, existe o *framework genérico de processamento* para aplicações XML. O intuito desse *framework* é reunir as características comuns para construção de compiladores genéricos independentes das linguagens. Por servir como um *framework* para a construção de outros *frameworks*, o *framework* genérico de processamento é também designado *meta-framework*.

Assim, escolhida a linguagem de origem, tem-se um *framework* para compiladores dessa linguagem, gerado a partir do *meta-framework*. Tendo por base esse *framework* de compiladores, em um segundo nível, compiladores específicos podem ser instanciados (como, por exemplo, compiladores para *players* SMIL). O objetivo é que cada compilador possa receber um documento descrito segundo a linguagem de origem e seja capaz de gerar uma especificação em um determinado *modelo de dados de destino*, que pode ser o modelo de um aplicativo, ou mesmo de uma outra linguagem declarativa.

O *meta-framework* foi aplicado a algumas linguagens baseadas em XML, resultando na construção dos *frameworks* de compiladores NCL, SMIL e X-SMIL, entre outros. A partir desses *frameworks*, foram desenvolvidos compiladores para conversão NCL em descrições SMIL e X-SMIL; para conversão SMIL em descrições NCL e X-SMIL; para conversão X-SMIL em descrições NCL e SMIL; para conversão de NCL, SMIL e X-SMIL para uma implementação de objetos do modelo NCM (Soares et al., 2003); e para uma implementação de objetos do modelo de execução do formatador HyperProp (Rodrigues et al., 2003; Rodrigues, 2003) - tornando-o, assim, capaz de exibir documentos dessas linguagens.

Além da compilação entre linguagens e modelos, o *framework* também foi utilizado na adaptação de documentos NCL, SMIL e X-SMIL. Essas linguagens oferecem suporte para ajustar a especificação dos documentos em função de parâmetros do contexto de exibição (ver Capítulo 1), tais como, preferências do usuário, características da plataforma etc. Quando tais parâmetros podem ser conhecidos antes de iniciar a apresentação do documento, a escolha da melhor alternativa de ajuste pode ser realizada por uma ferramenta de análise, que consulta os valores dos parâmetros e modifica a descrição do documento. A adaptação pode, assim, ser vista como a compilação de um documento com alternativas de conteúdos e exibições, para outro documento na mesma linguagem, agora com algumas (ou mesmo todas) alternativas resolvidas.

Este capítulo está organizado da seguinte forma. A primeira seção descreve o *framework* genérico (*meta-framework*) de processamento para linguagens modulares definidas por meio de XML. Nas seções seguintes, detalha-se o *framework* para o processamento de documentos NCL, SMIL e X-SMIL. Cada

uma dessas seções também discute os aspectos de implementação das instâncias desses *frameworks*.

5.1. *Framework* Genérico para Processamento

Diversas linguagens baseadas em XML apresentam uma grande complexidade. Essa complexidade reflete-se, normalmente, em um grande número de elementos e atributos. Com o objetivo de melhorar a estruturação e obter as vantagens mencionadas no Capítulo 2, várias aplicações XML definem módulos para agrupar elementos e atributos que possuam semântica relacionada. Como o número de módulos pode ser grande, algumas linguagens (como NCL e SMIL) definem um segundo nível de estruturação, agrupando seus módulos em *áreas funcionais*.

A complexidade de linguagens modulares também pode ser refletida na interdependência entre seus módulos: elementos de um módulo podem conter elementos pertencentes a outros módulos. Analogamente, atributos de elementos de um módulo são, muitas vezes, definidos em outros módulos.

Observando essa complexidade, este trabalho propõe um *framework* genérico de processamento (ou *meta-framework*), cujo principal objetivo é facilitar o desenvolvimento de *frameworks* de compiladores para aplicações XML modulares. O *meta-framework* define sintática e semanticamente os métodos de *frameworks* de compiladores, além de estabelecer a estruturação das classes desses *frameworks*.

A principal entrada do *meta-framework* é a definição de uma linguagem em XML Schema (W3C, 2001d). Por simplicidade, neste texto, essa entrada é representada utilizando as seguintes tabelas: a *tabela de elementos* e a *tabela de grupos de elementos*. Para linguagens que definem áreas funcionais, uma terceira tabela, que não pode ser obtida pela análise do XML Schema da linguagem, também é necessária: a *tabela de áreas funcionais*.

A tabela de elementos lista todos os elementos da linguagem, conforme exemplificado na Tabela 1. Cada elemento é identificado pelo seu nome (como *composite*) e pelo seu módulo (como *BasicComposite*), estando listados na primeira coluna. A segunda coluna da tabela descreve o conteúdo dos elementos.

Esse conteúdo é constituído de elementos atômicos (como *linkBase*) ou grupos de elementos (como *mediaContentGroup*). Elementos atômicos e grupos de elementos são listados da mesma forma, possuindo um nome e o módulo ao qual pertencem. A Tabela 1 apresenta uma definição parcial do elemento *composition* de NCL, que será utilizado como exemplo neste capítulo.

Elemento	**Conteúdo**
composite / BasicComposite	*linkBase / Linking* *mediaContentGroup / BasicMedia*

Tabela 7 - Tabela de Elementos.

A segunda tabela de entrada para o *meta-framework* lista todos os grupos de elementos da linguagem, conforme exemplificado na Tabela 8. A primeira coluna identifica o grupo de elementos, por seu nome e módulo, enquanto a segunda coluna lista o conteúdo de cada grupo (outros elementos da linguagem).

Grupo	**Conteúdo**
mediaContentGroup / BasicMedia	*text / BasicMedia* *img / BasicMedia* *audio / BasicMedia* *animation / BasicMedia* *video / BasicMedia* *textstream / BasicMedia*

Tabela 8 - Tabela de Grupos de Elementos.

A terceira e última tabela, chamada tabela de áreas funcionais, é necessária apenas para as linguagens que agrupam seus módulos em áreas funcionais, estando exemplificada na Tabela 9. A primeira coluna identifica o nome da área funcional, enquanto a segunda coluna lista os nomes dos módulos que compõem cada área funcional, devendo esses nomes ser únicos na linguagem.

Área Funcional	**Módulos**
Components	*BasicMedia*; *BasicComposite*
Linking	*Linking*

Tabela 9 - Tabela de Áreas Funcionais.

A abordagem adotada pelo *meta-framework* para construção dos *frameworks* de compiladores foi a seguinte: cada *framework* de compiladores implementa determinados métodos concretos e declara outros métodos abstratos para serem tratados pelas suas instâncias (os compiladores propriamente ditos),

dividindo-se assim as responsabilidades entre o *framework* e suas instâncias. Essa divisão segue o padrão de projeto *Template Method* (Gamma et al., 1995), que define um esqueleto para os algoritmos de cada classe (os métodos concretos) e delega certos passos às subclasses (através da definição de métodos abstratos). A definição da nomenclatura dos métodos e das especificações do corpo dos métodos concretos são, assim, de responsabilidade do *meta-framework*. A Figura 5:1 ilustra a abordagem proposta.

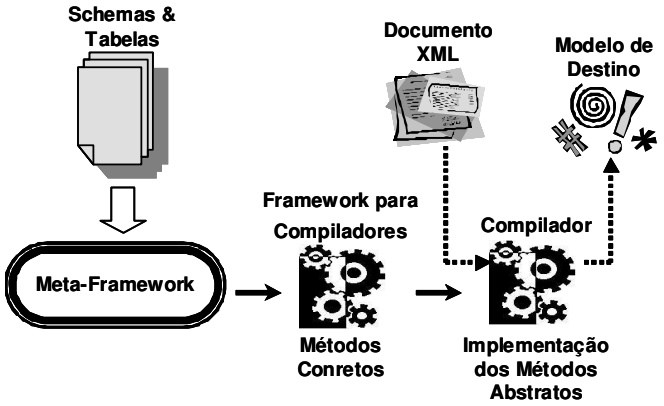


Figura 5:1 - . Visão geral da estruturação em dois níveis dos *frameworks* para compiladores de linguagens modulares.

A nomenclatura definida, pelo *meta-framework*, para os métodos dos *frameworks* de compiladores é a seguinte:

- métodos iniciados pelo nome *parse*, como *parseComposite*, são responsáveis por receber um elemento XML (elemento do tipo *composite*, por exemplo) como parâmetro, analisá-lo e retornar um objeto do modelo de destino representando esse elemento. Para a criação desse objeto, o método realiza chamadas a outros métodos do *framework* (concretos ou abstratos). Deve existir um método concreto do tipo *parse* para cada elemento e para cada grupo de elementos, listados na tabela de elementos e na tabela de grupos de elementos da linguagem de entrada, respectivamente. Os métodos *parse* fornecem a principal funcionalidade a ser herdada pelos compiladores, pois implementam o caminhamento pela estrutura dos documentos;
- métodos iniciados pelo nome *create*, como *createComposite*, são responsáveis por receber um elemento XML (*composite*) e criar o objeto representando o elemento no modelo de destino. Esses métodos são sempre

abstratos e são chamados pelos métodos (concretos) do tipo *parse*, devendo existir um método *create* para cada método *parse* criado;

- métodos iniciados pelo nome *add* são responsáveis por adicionar um objeto do modelo de destino a outro objeto do modelo de destino que o contém. Por exemplo, *addMediaContentGroupToComposite* recebe duas referências, uma para o objeto representando o elemento *composite* e outra para o objeto representando o elemento do grupo *mediaContentGroup* no modelo de destino; e adiciona o objeto representando o elemento *mediaContentGroup* ao objeto representando o elemento *composite*. Esses métodos são sempre abstratos e chamados pelos métodos *parse*. Para cada elemento da linguagem, deve existir um conjunto de métodos *add*; um para cada elemento (ou grupo) de seu conteúdo (entradas da segunda coluna na tabela de elementos).
- métodos iniciados pelo nome *preCompile* e *posCompile*, como *preCompileComposite* e *posCompileComposite*, são, respectivamente, o primeiro e último método chamado pelos métodos do tipo *parse*. Esses métodos concretos são declarados pelos *frameworks* de compiladores para possibilitar (mas não obrigar) que compiladores os re-implementem a fim de obter uma funcionalidade específica, como será visto. Devem existir métodos desses tipos para cada método do tipo *parse*.
- métodos concretos iniciados pelo nome *handleElementIn*, como *handleElementInComposite*, são chamados pelos métodos *parse* quando um elemento filho não é reconhecido como conteúdo de outro elemento. Compiladores (ou outros *frameworks* de compiladores) podem re-implementar esses métodos para tratar elementos não conhecidos previamente pelo *framework* de compiladores, que, dessa forma, oferece um mecanismo de extensão[35]. Deve ser implementado um método desse tipo para cada método do tipo *parse*.

Pode-se exemplificar a sintaxe e a semântica propostas para os métodos dos *frameworks* de compiladores a partir do elemento *composite* da Tabela 7 (que representa parcialmente o elemento *composite* de NCL). Um *framework* de

[35] Um exemplo do uso desse método será visto no *framework* para compiladores X-SMIL.

compiladores, gerado a partir dessa tabela[36], deve implementar o método concreto *parseComposite*. Esse método, representado em pseudo-código na Figura 5:2, recebe um elemento XML do tipo *composite* (*compositeElement*) como parâmetro e, inicialmente, efetua uma chamada ao método concreto *preCompileComposite*. Esse método pode alterar o *compositeElement* ao retornar um elemento do tipo *composite* pré-processado. Em seguida, *parseComposite* faz uma chamada ao método abstrato *createComposite* para que seja criado o objeto *objComposite*, representando o elemento XML do tipo *composite* no modelo de destino.

```
01  public Object parseComposite(compositeElement) {
02    //pré-compilar elemento composite
03    compositeElement = preCompileComposite(compositeElement);
04    objComposite = createComposite(compositeElement);
05    para cada elemento childCompositeElement filho de compositeElement {
06      se childCompositeElement do tipo LinkBase {
07        //elemento filho do tipo linkBase, criar linkBase
08        childCompositeObject = parseLinkBase(childCompositeElement);
09        //adicionar linkBase à composite
10        addLinkBaseToComposite(objComposite, childCompositeObject);
11      } se childCompositeElement do tipo mediaContentGroup {
12        //elemento filho do tipo mediaContentGroup, criar mediaContentGroup
13        childCompositeObject
                    = parseMediaContentGroup (childCompositeElement);
14        //adicionar mediaContentGroup à composite
15        addMediaContentGroupToComposite
                            (objComposite, childCompositeObject);
16      }se não{
17        //elemento filho de composite não reconhecido.
18        //criar e adicionar elemento à composite
19        handleElementInComposite
                                (objComposite, childCompositeElement);
20      } //fim do bloco "se"
21    } //fim do bloco "para cada"
22    //pós-compilar objComposite
23    objComposite = posCompileComposite(objComposite);
24    //retornar objComposite
25    return objComposite;
26  }
27
28
29  public Element preCompileComposite(compositeElement) {
30    return compositeElement;
```

[36] Cabe observar que, na realidade, o *framework* é gerado a partir do XML Schema da linguagem. Conforme comentado anteriormente, as tabelas de elementos e de grupo de elementos foram utilizadas apenas para facilitar a descrição do processamento.

```
31  }
32
33  public Object posCompileComposite(objComposite) {
34    return objComposite;
35  }
36
37  public void handleElementInComposite
                          (objComposite, childCompositeElement) {
38    //método vazio
39  }
```

Figura 5:2 - Exemplo de um método do tipo *parse* de um *framework* de compiladores.

O próximo passo de *parseComposite* é analisar o conteúdo do elemento XML recebido como parâmetro, ou seja, todos os seus elementos filhos (listados na segunda coluna da tabela de elementos). De acordo com o tipo de cada elemento filho, o método *parseComposite* chama um dos métodos *parse* a seguir: *parseLinkBase* ou *parseMediaContentGroup*. Esses métodos deverão retornar representações dos elementos contidos em *composite* no modelo de destino. Cada um desses objetos é, então, adicionado ao objeto *objComposite* pelos métodos *addLinkBaseToComposite* e *addMediaContentGroupToComposite*, respectivamente. Se o elemento filho do elemento *composite* não estiver listado na tabela de elementos[37], o método *parse* chama o método concreto *handleElementInComposite*, responsável por compilar e adicionar esse elemento a *objComposite*.

Finalmente, o método *parseComposite* efetua uma chamada ao método concreto *posCompileComposite*, que pode alterar o objeto *objComposite* que será, em seguida, retornado pela função *parse*.

Os métodos concretos *preCompileComposite*, *posCompileComposite* e *handleElementInComposite* também estão apresentados na Figura 5:2. O *framework* de compiladores declara esses métodos como concretos para facilitar a implementação de compiladores que não necessitem utilizá-los. O comportamento padrão do primeiro método é retornar o elemento XML recebido como parâmetro, enquanto o segundo retorna o objeto recebido e, finalmente, o terceiro não executa qualquer código.

Além da definição da assinatura dos métodos e das especificações do corpo dos métodos concretos, o *meta-framework* também determina uma estrutura de

classes a ser seguida por cada *framework* de compiladores. Em primeiro lugar, cada um desses *frameworks* deve implementar uma subclasse da classe gerenciadora *CFDocumentParser*[38]. Essa subclasse[39] deve conter métodos para leitura e compilação de arquivos XML especificados na linguagem de entrada, além de referências para todas as outras classes do *framework* de compiladores. Caso a linguagem para a qual se pretende construir um *framework* de compiladores seja organizada em áreas funcionais, propõe-se que seja criada uma classe representando cada área funcional; caso contrário, deve ser criada uma classe para cada módulo da linguagem[40]. Essas classes devem herdar da classe abstrata *CFModuleParser*, que define uma referência para a classe gerenciadora *CFDocumentParser*. Como módulos (ou áreas funcionais) dependem de outros módulos (ou áreas funcionais), a classe gerenciadora também deve realizar as chamadas para atribuir as referências entre as classes, como será visto.

Outra recomendação do *meta-framework* é a criação de *frameworks* para compiladores refletindo o perfil completo de uma linguagem. Dessa forma, outros perfis da linguagem, sempre mais restritos que o perfil completo, podem utilizar o mesmo *framework*, e os mesmos compiladores, para o processamento de seus documentos. Porém, nada impede a definição de classes gerenciadoras distintas, ou até mesmo *frameworks* de compiladores diferentes (mais simples), para perfis específicos de uma linguagem.

Para geração automática dos métodos e das classes de um *framework* de compiladores, foi desenvolvida uma aplicação baseada no *meta-framework*, chamada *gerador de framework de compiladores*. O diagrama de classes em UML

---

[37] Isso pode ocorrer quando o método *parse* está sendo usado para compilar documentos em outra linguagem (ou perfil) que não a utilizada para gerar o *framework*.

[38] O prefixo *CF* das classes do *framework* significa "*Compiler Framework*" (*framework* de compiladores).

[39] A classe *CFDocumentParser* implementa alguns métodos que são herdados por suas subclasses. Entre esses métodos, encontram-se: um método para geração de uma árvore DOM (W3C, 2000a) a partir de um arquivo XML qualquer; e métodos para criação de tabelas, adição de objetos a essas tabelas, e acesso a objetos nelas contidos. Um exemplo do uso dessas tabelas é apresentado na próxima seção.

[40] Linguagens que não são estruturadas em módulos podem ser consideradas como uma linguagem modular com um único módulo. Assim, *frameworks* de compiladores também podem ser gerados para essas linguagens.

(Rumbaugh et al., 1999) dessa aplicação está ilustrado na Figura 5:3. A classe *CFGenerator* possui métodos tanto para geração de *framework* de compiladores de uma linguagem quanto para geração de esqueletos de código de compiladores para essa linguagem.

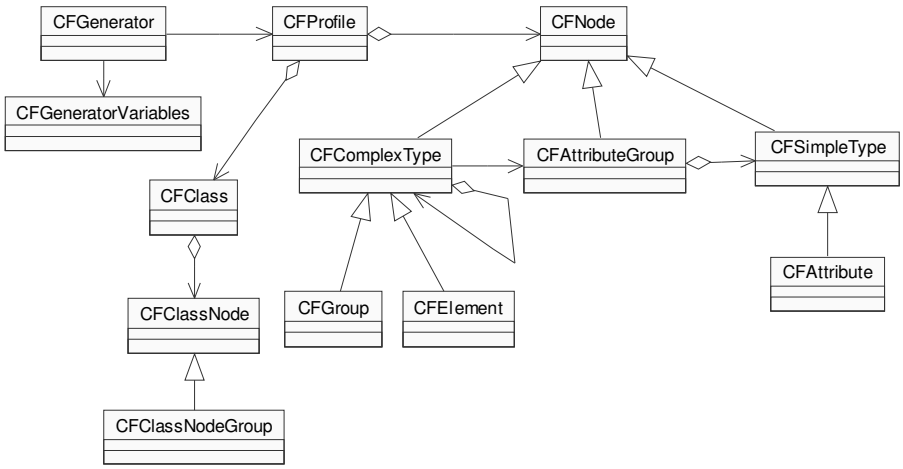


Figura 5:3 - Diagrama de classes do gerador automático de *frameworks* de compiladores.

A geração automática de código necessita de algumas informações, que devem ser parametrizadas por uma instância da classe *CFGeneratorVariables*. Essa classe permite que *CFGenerator* consulte a localização do *schema* que define uma linguagem, e dados que não podem ser obtidos diretamente pela análise desse *schema*, como o nome da linguagem e um mapeamento entre seus módulos e áreas funcionais (informações contidas na Tabela 9).

A partir das informações contidas em *CFGeneratorVariables*, a classe *CFGenerator* constrói um objeto perfil (da classe *CFProfile*) refletindo a especificação de uma determinada linguagem (ou perfil de linguagem). Para isso, a definição de uma linguagem - realizada por meio dos elementos *attribute*, *attributeGroup*, *complexType* , *element*, *group* e *simpleType* de XML Schema (W3C, 2001d) - é usada para geração de objetos das classes *CFAttribute*, *CFAttributeGroup*, *CFComplexType*, *CFElement*, *CFGroup* e *CFSimpleType*, respectivamente[41]. Em um primeiro passo, o perfil (*CFProfile*) gerado reflete

[41] Existem outros elementos de XML Schema, além dos citados, que podem ser usados na especificação de linguagens. Porém, como esses elementos não definem informações relevantes para a geração de *frameworks*, não foram implementadas classes para representá-los. Esses

exatamente o *schema* da linguagem, porém, como XML Schema possui diversas formas de definição de tipos através de referência a outros tipos, um segundo passo é necessário: computar essas referências. A partir desse perfil (*CFProfile*) com as referências calculadas, têm-se disponíveis as informações representadas pela Tabela 7 e pela Tabela 8, sendo possível, assim, a geração das classes do *framework* de compiladores.

Para cada área funcional (ou módulo) da linguagem, a classe *CFProfile* gera, em um terceiro passo, um objeto da classe *CFClass*. Esse objeto possui objetos das classes *CFClassNode* e *CFClassNodeGroup* referentes, respectivamente, aos elementos e aos grupos de elementos da área funcional (ou módulo) que ele representa. Cada objeto *CFClass* irá gerar uma classe do *framework* de compiladores, cujos métodos são criados a partir de cada um de seus nós (*CFClassNode* e *CFClassNodeGroup*).

Dando continuidade ao exemplo do elemento *composite* da Tabela 7, e supondo que o nome da linguagem seja definido como NCL, o gerador de *frameworks* irá criar (a partir de objetos *CFClass*) uma classe *NclComponentsParser*[42] (representando a área funcional *Components*) e uma classe *NclLinkingParser* (representando a área funcional *Linking*). Essas classes herdam de *CFModuleParser* e terão seus métodos do tipo *parse*, *create*, *add*, *preCompile*, *posCompile* e *handleElementIn* definidos para cada um dos seus *nós de classe* (objetos das classes *CFClassNode* e *CFClassNodeGroup*). Por exemplo, na classe *NclComponentsParser*, teremos os métodos *parseComposite*, *createComposite*, *parseMediaContentGroup*, *addLinkBaseToComposite*, *addMediaContentGroupToComposite* etc. Já na classe *NclLinkingParser*, serão gerados os métodos *parseLinkBase* e *createLinkBase*, entre outros. Além das classes representando áreas funcionais, será gerada, também, a classe gerenciadora *NclDocumentCompiler*[43], herdando de *CFDocumentParser*.

---

elementos são utilizados durante a análise do *schema* da linguagem para geração dos objetos das classes definidas pelo *meta-framework*.

[42] O nome das classes geradas é obtido pela concatenação do nome da linguagem com o nome da área funcional, e com a palavra "*parser*".

[43] O nome da classe gerenciadora é obtido pela concatenação do nome da linguagem com a palavra "*DocumentCompiler*".

Como comentado, de forma semelhante à geração de *framework* de compiladores, a classe *CFGenerator* também pode gerar esqueletos para compiladores específicos de uma linguagem, a partir do nome do modelo de destino do compilador. Supondo que o modelo de destino seja o NCM e a linguagem seja NCL, seguindo com o exemplo, podem ser geradas as classes *NclNcmDocumentCompiler*, *NclNcmComponentsParser* e *NclNcmLinkingParser*[44]. Essas classes declaram métodos concretos para cada método abstrato definido por suas superclasses (*NclDocumentCompiler*, *NclComponentsParser* e *NclLinkingParser*). O corpo desses métodos deve ser implementado pelo programador do compilador de documentos NCL para objetos do NCM.

Finalmente, em alguns casos, compiladores específicos para uma determinada linguagem também podem ser gerados automaticamente pelo gerador de *frameworks*. Isso ocorre quando é possível estabelecer um mapeamento genérico entre um documento XML e a estrutura de dados de destino. Um exemplo implementado foi a geração automática de compiladores que produzem simplesmente cópias de documentos de uma determinada linguagem. Apesar de à primeira vista pouco necessário, esse gerador foi extremamente útil na implementação de um adaptador de documentos NCL, como será discutido na Seção 5.2.2. Outro exemplo implementado foi a geração automática de um compilador que constrói uma visualização gráfica em árvore de um documento XML. Java foi a linguagem de programação utilizada na implementação dos *frameworks* e dos compiladores deste trabalho, usando a API JAXP (Sun, 2002).

5.2. *Framework* para Compiladores de Documentos NCL

O *framework* para compiladores NCL é uma instância do *meta-framework* descrito na seção anterior. A Figura 5:4 apresenta o diagrama UML das classes do *framework* gerado automaticamente, onde cada classe representa uma área funcional da linguagem NCL. Existe também uma classe gerenciadora, chamada *NclDocumentCompiler*, que reúne as áreas funcionais e representa o ponto de

[44] A nomenclatura utilizada para as classes dos compiladores é similar àquelas utilizadas para classes do *framework*, adicionando o nome do modelo de destino após o nome da linguagem.

entrada para compilação de documentos que sigam o perfil completo da linguagem. As setas entre classes de áreas funcionais simbolizam relações de dependência entre os seus módulos.

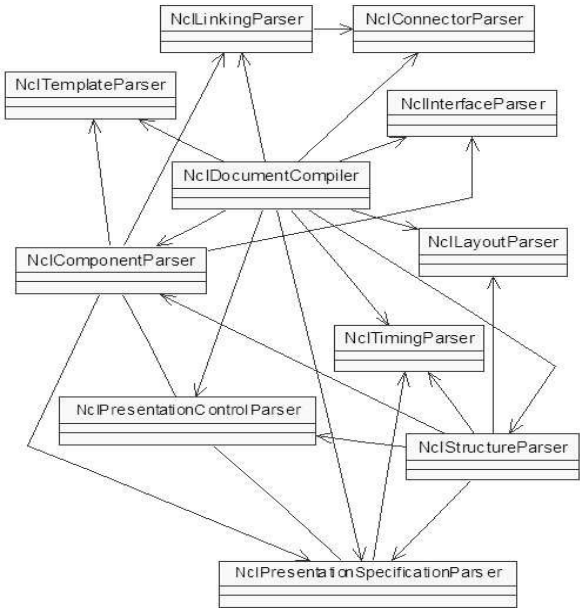


Figura 5:4 - Diagrama de classes do *framework* para compiladores NCL.

5.2.1. Funcionamento do *Framework* de Compiladores NCL

O documento NCL exemplificado na Figura 5:5 servirá de base para ilustrar as principais características do funcionamento do *framework* de compiladores NCL. A compilação de um documento NCL inicia-se quando a classe gerenciadora *NclDocumentCompiler* chama o método *parseNcl* da classe *NclStructureParser*. Esse método analisa o elemento *ncl* e seus elementos filhos: *head* e *body*.

```
69  <ncl>
70   <head>
71    <layout>...</layout>
72    <descriptorBase>
73     <descriptor id="d0" dur="30s" region="audioRegion"/>
74     <descriptor id="d1" region="title"/>
75     <descriptor id="d2" region="textRegion"/>
76     <descriptor id="d3" region="imageRegion" player="quickTime"/>
77    </descriptorBase>
78   </head>
79   <body>
80    <composite id="C">
81     <composite id="C1">
```

```
82      <port id="portA1" component="audio1" />
83      <audio id="audio1" descriptor="d0" src="">
84       <area id="part1" begin="8.4s" end="18s"/>
85       <area id="part2" begin="18.5s" end="28s"/>
86      </audio>
87      <text id="title1" descriptor="d1" src=""/>
88      <text id="lyr11" descriptor="d2" src=""/>
89      <text id="lyr12" descriptor="d2" src=""/>
90     </composite>
91     <composite id="C2">
92      <port id="portA2" component="audio2" />
93      <audio id="audio2" descriptor="d0" src=""/>
94      <composite id="C21">
95       <text id="title2" descriptor="d1" src=""/>
96       <text id="lyr21" descriptor="d2" src=""/>
97      </composite>
98     </composite>
99     <img id="img-1" descriptor="d3" />
100    <linkBase>
101     <link id="L1" xconnector="finishes.xml">
102      <bind component="C1" port="portA1" role="on_x_presentation_end" />
103      <bind component="C2" port="portA2" role="stop_y" />
104      </link>
105    </linkBase>
106   </composite>
107  </body>
108 </ncl>
```

Figura 5:5 - Exemplo simplificado de documento NCL.

Durante a compilação do elemento *head* (método *parseHead*[45]) o método *parseDescriptorBase* é chamado para compilar a base de descritores (elemento *descriptorBase*). Esse método concreto chama, para cada descritor presente na base (elemento *descriptor*), outro método concreto do tipo *parse*: *parseDescriptor*. Esse método, por sua vez, chama o método abstrato *createDescriptor*, que deve retornar a representação de um descritor no modelo de destino do compilador. Evidentemente, a implementação do método *createDescriptor* é de responsabilidade de cada compilador que venha a ser instanciado a partir do *framework*.

Seguindo com o exemplo da Figura 5:5, durante a compilação do corpo do documento NCL, o método concreto *parseComposite*, declarado na classe *NclComponentsParser*, será chamado para compilar a composição *C*. Esse método

[45] Para facilitar a explicação do exemplo, chamadas a alguns métodos, tais como os métodos de pré e pós processamento, foram omitidas.

retornará um objeto representando essa composição, criado através de uma chamada ao método abstrato *createComposite*. Em seguida, cada elemento interno à composição será compilado. Inicialmente, as composições *C1* e *C2* serão tratadas por chamadas recursivas ao método *parseComposite*. Os objetos representando *C1* e *C2* retornados por essas chamadas serão adicionados à composição *C* através de uma chamada ao método abstrato *addCompositeToComposite*.

Encerrada a compilação das composições *C1* e *C2*, a imagem *img-1* será compilada pelo método *parseMediaContentGroup* (já que o elemento *img*, que define uma imagem, faz parte do grupo de elementos *mediaContentGroup*). O objeto retornado pela chamada a esse método será adicionado a *C* através de uma chamada ao método *addMediaContentGroupToComposite*. Ambos os métodos são também definidos na classe *NclComponentsParser*.

A compilação das composições *C1* e *C2* ocorrerá semelhantemente à compilação de *C*. Os objetos de mídia *audio1*, *title1*, *lyr11* e *lyr12* serão compilados e os resultados das compilações adicionados a *C1*, enquanto *audio2* e *C21* serão compilados e os objetos retornados adicionados a *C2*.

Como pode ser observado, a estrutura genérica do *framework* de compiladores NCL responsabiliza-se pelo caminhamento através da árvore de elementos XML do documento. Às instâncias de compiladores, cabe apenas implementar os métodos abstratos definidos pelo *framework* para criação dos objetos no modelo de destino.

Como descritores são referenciados por objetos de mídia através do atributo *descriptor*, um compilador pode necessitar de uma tabela (indexada pelo *id* do descritor) com todos os descritores retornados pelo método *createDescriptor*. Nesse caso, pode ser usado o vetor de tabelas oferecido pela classe *CFDocumentParser*, superclasse de *NclDocumentCompiler* (como mencionado na Seção 5.1). Assim, o método *createDescriptor* pode adicionar o objeto por ele retornado a uma tabela "descritores" de *NclDocumentCompiler*. Para recuperar uma referência ao descritor apontado por um objeto de mídia, o método *createMediaContentGroup* pode consultar a tabela "descritores" de *NclDocumentCompiler* e obter o descritor a partir de seu *id*.

O vetor de tabelas oferecido por *CFDocumentParser* também pode ser utilizado para criação de uma tabela chamada "retorno". Essa tabela pode conter

alguns dos objetos compilados, de forma a permitir à aplicação que utiliza um compilador obter os objetos retornados pelo compilador. No caso de NCL, essa tabela de retorno pode conter, por exemplo, a composição representada pelo elemento *body*, uma base de descritores, um objeto *layout*, entre outros.

Outro aspecto do *framework* que o exemplo permite destacar é a dependência entre as classes (uma classe chama métodos de outras classes). Como descrito anteriormente, é papel da classe *NclDocumentCompiler* estabelecer as dependências entre as classes do *framework*. Por exemplo, a classe *NclDocumentCompiler* atribui uma referência da classe *NclComponentsParser* para a classe *NclPresentationSpecificationParser*. O código para atribuição das referências entre classes é gerado automaticamente pelo *gerador de frameworks de compiladores*.

5.2.2. Compiladores de documentos NCL

A partir do *framework* de compiladores NCL, foram implementadas quatro instâncias de compiladores, ilustradas na Figura 5:6. Cada instância foi criada pela implementação de uma subclasse para cada uma das classes (abstratas) definidas na Figura 5:6.

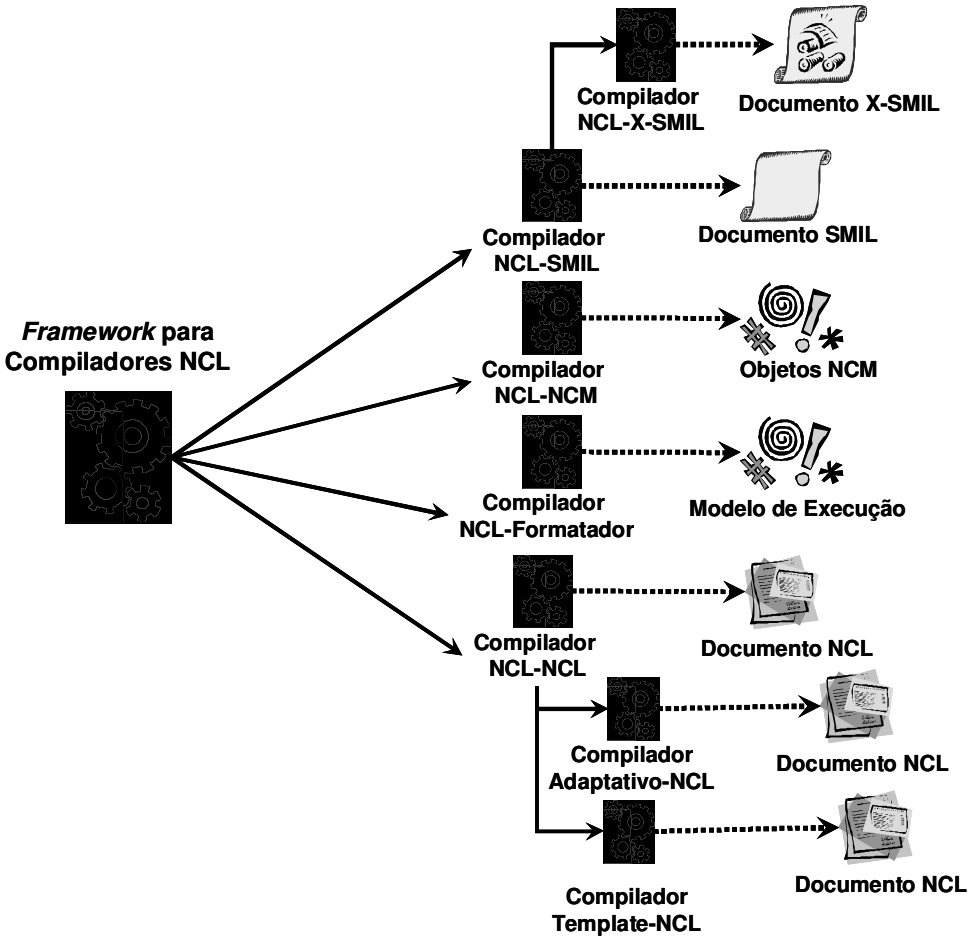


Figura 5:6 - Compiladores NCL.

A primeira instância é o compilador NCL-NCM Java, que gera, como indica o nome, objetos em uma implementação Java do modelo NCM. A estrutura de dados gerada pode ser usada nas ferramentas de autoria do sistema HyperProp. Como NCL é baseada no modelo conceitual NCM, a implementação dos métodos abstratos dessa instância do *framework* foi trivial.

A segunda instância é o compilador NCL-Formatter. Esse compilador gera objetos segundo a estrutura de dados utilizada pelo formatador HyperProp. Os métodos abstratos foram implementados com base nas funções de pré-compilação do formatador, que constroem uma estrutura de dados orientada ao controle da apresentação de um documento. Antes dessa implementação estar disponível, para exibir um documento NCL no formatador HyperProp, esse documento tinha de ser importado para as ferramentas de autoria do sistema HyperProp, através do compilador NCL-NCM, e somente depois convertido para o modelo do formatador. A eliminação do passo intermediário de compilação para objetos

NCM trouxe, como principais benefícios, a redução das necessidades de memória e armazenamento do formatador, além da redução do tempo de conversão. O intuito é favorecer a implementação do conversor/formatador em dispositivos de exibição com recursos limitados, como telefones móveis e *set-top boxes* de TV.

A terceira instância do *framework* é o compilador NCL-SMIL, que gera documentos SMIL 2.0 a partir de documentos NCL. Os elementos pertencentes a módulos que foram trazidos diretamente de SMIL para NCL (como *BasicMedia*), ou ligeiramente modificados (como *Layout*), foram implementados de forma simples. Composições NCL foram mapeadas em elementos *par* e a sincronização definida pelos elos NCL foi descrita através dos eventos de SMIL, como será visto a seguir. Os principais problemas ocorreram com as entidades que só estão presentes em NCL.

A maior dificuldade no mapeamento diz respeito ao uso de conectores. Para possibilitar a transformação, uma classe foi criada com o intuito de simular o comportamento de um conector em SMIL: *SmilConnector*. Ao compilar um conector, uma instância de *SmilConnector* é criada, cujo objetivo é inferir quais eventos SMIL podem ser utilizados para representar o conector NCL. No algoritmo implementado, nem todas as semânticas que podem ser expressadas por conectores NCL podem ser expressas em documentos SMIL.

A Figura 5:7 ilustra uma composição SMIL gerada pelo compilador NCL-SMIL, ao compilar uma composição NCL que referencia o *template audioComLegendas* - utilizado como exemplo nos Capítulos 2 e 3. Note que foi criada uma composição *par* referente à composição NCL; e que foi definido o sincronismo entre o áudio, o *logo* e as legendas por meio de eventos - nos atributos *begin* e *end* dos objetos de mídia - referentes a cada elo NCL. Como somente dois conectores foram utilizados pelos elos da composição NCL, somente duas instâncias da classe *SmilConnector* foram criadas e utilizadas na compilação de todos esses elos.

```
01  <par>
02   <audio id="samba" region="" src="">
03    <area begin="8.4s" end="18s" id="part1"/>
04    <area begin="18.5s" end="28s" id="part2"/>
05    <area begin="29s" end="39s" id="part3"/>
06    <area begin="39.5s" end="50s" id="part4"/>
07    <area begin="50.5s" end="71.4s" id="part5"/>
08    <area begin="72s" end="94s" id="part6"/>
```

```
09   </audio>
10   <img begin="samba.begin" end="samba.end" id="logo" region="" src=""/>
11   <text begin="part1.begin" end="part1.end" id="lyrics1" region="" src=""/>
12   <text begin="part2.begin" end="part2.end" id="lyrics2" region="" src=""/>
13   <text begin="part3.begin" end="part3.end" id="lyrics3" region="" src=""/>
14   <text begin="part4.begin" end="part4.end" id="lyrics4" region="" src=""/>
15   <text begin="part5.begin" end="part5.end" id="lyrics5" region="" src=""/>
16   <text begin="part6.begin" end="part6.end" id="lyrics6" region="" src=""/>
17  </par>
```

Figura 5:7 - Composição SMIL gerada a partir do compilador NCL-SMIL.

O compilador NCL-X-SMIL é uma especialização do compilador NCL-SMIL. Entretanto, como X-SMIL possui o conceito de bases de elos (e elos referenciando conectores) e o elemento *composite*, a compilação de NCL para X-SMIL é mais simples do que para SMIL. Assim, composições (elemento *composite*) NCL são mapeadas para o elemento homônimo em X-SMIL e não é necessária a compilação de conectores (bases de elos NCL são transformadas em bases de elo X-SMIL). As demais funcionalidades do compilador NCL-X-SMIL são herdadas do compilador NCL-SMIL.

À exceção do compilador NCL-X-SMIL, todos os compiladores NCL descritos anteriormente utilizam o compilador XConnector-NCM, instância do *framework* para compiladores XConnector, para processar os conectores que são referenciados por elos. No caso do compilador NCL-NCM, esse uso é obvio. Nos compiladores NCL-Formatador e NCL-SMIL esse compilador de XConnector foi o utilizado por permitir um acesso mais fácil e eficiente às definições de conectores do que seria possível analisando diretamente a estrutura XML dessas definições. Outro ponto importante, nesses compiladores, é o uso do processador de *templates*, que analisa o elemento *body* do documento NCL, processando todas as composições que referenciam *templates* antes que seja iniciado a compilação do elemento *body* e de seu conteúdo. O compilador de XConnector e o processador de *templates* serão discutidos na Seção 5.5.

O último compilador implementado é o NCL-NCL. Esse compilador (gerado automaticamente, conforme mencionado na Seção 5.1) implementa todos os métodos abstratos do *framework* simplesmente gerando uma cópia do documento NCL de entrada. O compilador NCL-NCL foi projetado para se tornar a base para implementação de compiladores NCL adaptativos. Como exemplo, uma especialização desse compilador foi implementada para computar as alternativas de elementos (conteúdo de *switches*) que podem ser resolvidas e

selecionadas antes de iniciar uma apresentação. No novo documento NCL gerado, o elemento *switch* é substituído pelo elemento escolhido, ou por um elemento *switch* com menos alternativas (são retiradas aquelas impossíveis de serem selecionadas). Outro compilador NCL baseado no compilador NCL-NCL é o que processa todos os *templates* de um documento NCL e gera um outro documento NCL processado (ou seja, sem referências a *templates*). Evidentemente, esse compilador também utiliza o processador de *templates* que será descrito na Seção 5.5.

5.3. *Framework* para Compiladores de Documentos SMIL

O *framework* para compiladores SMIL foi a segunda instância gerada automaticamente a partir do *meta-framework*. O diagrama UML de classes dessa instância é ilustrado na Figura 5:8, sendo constituído da classe gerenciadora *SmilDocumentCompiler* e das demais classes representando as áreas funcionais de SMIL. Como as principais características do processo de instanciação de um *framework* para compiladores já foram abordadas na seção anterior, esta seção limita-se a descrever os compiladores SMIL implementados: SMIL-SMIL, SMIL-NCL, SMIL-NCM e SMIL-Formatador.

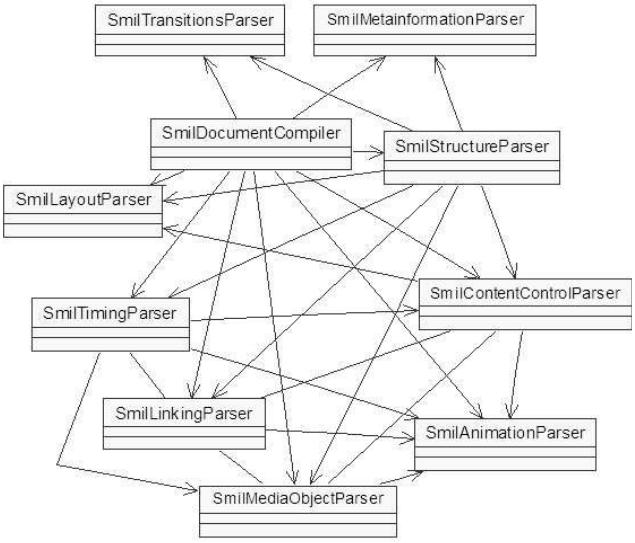


Figura 5:8 - Diagrama de classes do Framework para Compiladores SMIL.

O compilador SMIL-SMIL (também gerado automaticamente) implementa todos os métodos abstratos do *framework* para a geração de uma cópia do documento SMIL de entrada. Similar ao compilador NCL-NCL, esse compilador

é a base para implementação de compiladores adaptativos SMIL, e teve uma especialização implementada para avaliar os elementos *switch* de um documento SMIL.

O compilador SMIL-NCL, como o próprio nome sugere, efetua processamento inverso ao compilador NCL-SMIL. Como nesse último compilador, elementos similares ou comuns nas duas linguagens (como *layout* e o grupo *BasicMedia*) foram trivialmente convertidos. As características de apresentação de cada elemento SMIL (como a região de *layout*) foram agrupadas em descritores NCL (criados pelo compilador). Além disso, cada elemento SMIL, quando compilado para NCL, teve um atributo *descriptor* adicionado para referenciar o descritor criado.

Contêineres SMIL (*par*, *seq*, ou *excl*) foram convertidos em composições NCL. Quando um elemento é adicionado a uma composição NCL, cria-se um elo NCL relacionando esse elemento com os outros elementos já presentes na composição (caso esse não seja o primeiro elemento sendo inserido). Dependendo do contêiner SMIL que deu origem à composição, o elo criado referenciará um determinado conector (já criado pelo compilador): *SmilPar* (contêiner *par*), *SmilSeq* (contêiner *seq*) e *SmilExcl* (contêiner *excl*). Os elos NCL criados oferecem a mesma semântica temporal atribuída pelo contêiner SMIL[46].

Elos em SMIL podem ser definidos a partir de eventos, como os de início e término de apresentação (eventos *begin* e *end*) e de acionamento de mouse (evento *click*), ou a partir do uso de elementos do tipo *a*. Para cada um dos tipos de elos que podem ser definidos em SMIL, foi pré-definido um conector NCL para representar a sua semântica e gerar um ou mais elos NCL correspondentes[47].

Como NCL é baseada no modelo NCM, o compilador SMIL-NCM adota a mesma abordagem utilizada para a compilação de documentos SMIL em documentos NCL e, portanto, não será detalhado neste trabalho. Maiores detalhes do mapeamento entre os modelos podem ser encontrados em [Rodr02].

---

[46] Em alguns casos, mais de um conector é necessário para relacionar elementos de uma composição NCL refletindo uma composição SMIL. Por limitação de espaço, esses casos e as soluções adotadas nos compiladores foram omitidos no texto.

[47] Quando um elemento define o atributo *begin*, o *link* adicionado automaticamente para iniciar a apresentação desse elemento é retirado, para que o início da apresentação do elemento seja dado pelo evento definido no atributo *begin*, como determina a semântica de SMIL.

Por fim, o compilador SMIL-Formatador foi baseado na concatenação dos compiladores SMIL-NCL e NCL-Formatador; e o compilador SMIL-X-SMIL é o próprio compilador SMIL-SMIL, já que todo documento SMIL é um documento válido (e com mesma semântica) em X-SMIL.

5.4. *Framework* para Compiladores de Documentos X-SMIL

Diferente dos *frameworks* para compiladores SMIL e NCL, o *framework* para compiladores X-SMIL não foi gerado automaticamente, mas implementado como uma instância do *framework* para compiladores SMIL. Para tanto, o *framework* para compiladores X-SMIL implementa a classe *XSmilDocumentCompiler*, *XSmilStructureParser* e *XSmilTimingParser* que herdam, respectivamente de *SmilDocumentCompiler*, *SmilStructureParser* e *SmilTimingParser* do *framework* para compiladores SMIL. Utilizando os métodos do tipo *handleElementIn*, esse *framework* adiciona o tratamento dos elementos *linkBase* e *composite* de documentos X-SMIL. Para prover suporte a esse tratamento, o *framework* X-SMIL também aproveita a classe *NclLinkingParser* do *framework* NCL. As demais classes do *framework* X-SMIL são as mesmas do *framework* SMIL.

Por essa estruturação é possível observar que os compiladores X-SMIL são adaptações dos compiladores SMIL combinados com os compiladores NCL já descritos. Assim como os compiladores NCL, compiladores X-SMIL também necessitam processar os *templates* antes da compilação do elemento *body*. Os elementos *linkBase* e *composite* são tratados de forma semelhante ao tratamento dos compiladores NCL, enquanto os demais elementos são compilados da mesma maneira que em SMIL. Herdando partes dos compiladores SMIL-SMIL e NCL-NCL, também foi implementado um compilador X-SMIL-X-SMIL. Uma especialização desse compilador permite gerar documentos X-SMIL com todos os seus *templates* processados.

Os processadores de *templates*, assim como os compiladores de conectores, são assuntos da próxima seção.

5.5. *Framework* para Compiladores de Documentos XConnector e XTemplate

Apesar das linguagens XConnector e XTemplate serem parte integrante da definição da linguagem NCL, cada uma dessas linguagens possui sua própria especificação em XML Schema. Assim, foi gerado o *framework* de compiladores XConnector a partir do gerador de *frameworks*[48], e também uma instância desse *framework*: o compilador XConnector-NCM Java. Uma vez que o modelo NCM serviu como base para a especificação da linguagem XConnector, a implementação desse compilador foi trivial.

Seguindo a mesma abordagem, foi gerado automaticamente um *framework* para compiladores XTemplate 2.1. Uma instância desse *framework* foi implementada como parte integrante do processador de *templates* XTemplate 2.1, que permite a atribuição de semântica às composições NCL, XT-SMIL e X-SMIL.

Um método implementado no processador de *templates* recebe um elemento XML como parâmetro e processa todos os *templates* referenciados pelos elementos filhos desse elemento. Esse método é utilizado por compiladores NCL e X-SMIL para pré-processar o elemento *body* antes de sua compilação (utiliza-se a função *preCompileBody* dos *frameworks* de compiladores).

A Figura 5:9 ilustra graficamente o processamento de um *template*, onde pode-se observar as seguintes entradas: um elemento representando uma composição que referencia um *template* e o documento XTemplate com a especificação do *template* referenciado.

[48] Mesmo a linguagem XConnector não sendo especificada de forma modular, o gerador de *framework* de compiladores pôde ser usado, ao considerar XConnector como uma linguagem com um único módulo.

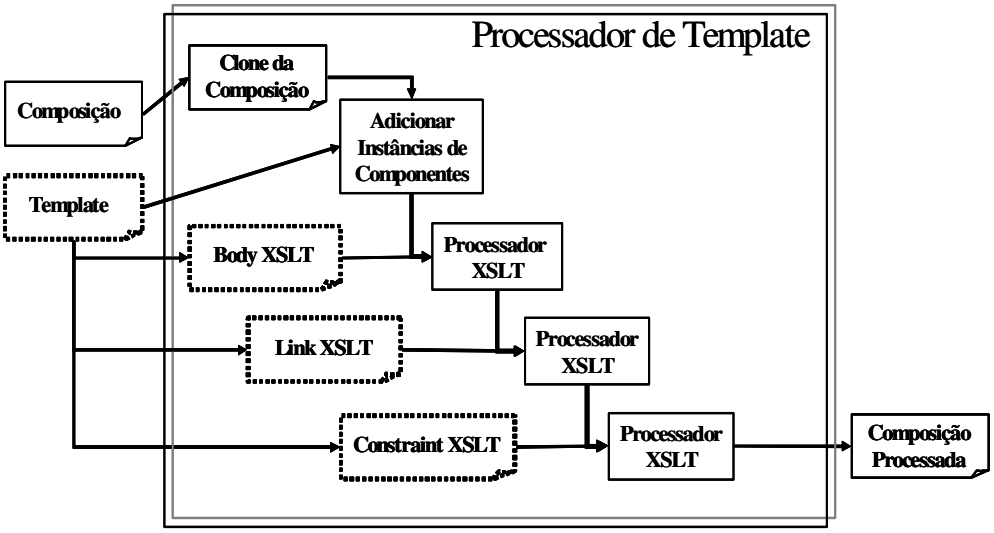


Figura 5:9 - Processador de *Template* XTemplate 2.1.

O processador de *template*, inicialmente, gera uma cópia da composição original. Essa cópia é inserida como único elemento filho do elemento raiz (nomeado *root*) de um documento XML temporário. Considerando, como exemplo, o processamento de uma composição *C1* que referencie o *template audioWithSubtitlesEnPt21*, descrito no Capítulo 3 (Figura 3:14), o documento XML temporário, contendo uma cópia dessa composição, está ilustrado na Figura 5:10.

Em seguida, analisando os recursos declarados no cabeçalho do *template*, instâncias de componentes (recursos) definidas por elementos *resource* são adicionadas a essa cópia. Como *audioWithSubtitlesEnPt21* não define recursos no cabeçalho, nenhum elemento é adicionado nesse passo. No outro exemplo de *template* do Capítulo 3 (Figura 3:12), o recurso *logoJPG* seria adicionado, nesse momento, como elemento filho da cópia da composição.

```
01  <root>
02  <composite id="C1" xtemplate=" audioWithSubtitlesEnPt21.xml">
03    <audio id="samba" src="coisadepeleBeth.wav" type="song">
04      <area begin="8.4s" end="18.5s" id="part1" type="track"/>
05      <area begin="18.5s" end="29s" id="part2" type="track"/>
06      <area begin="29s" end="39.5s" id="part3" type="track"/>
07    </audio>
08    <text id="lyrics-part1a" src="versosEn.html#versos01" type="subtitleEn"/>
09    <text id="lyrics-part1b" src="versosPt.html#versos01" type="subtitlePt"/>
10    <text id="lyrics-part2a" src="versosEn.html#versos02" type="subtitleEn"/>
11    <text id="lyrics-part2b" src="versosPt.html#versos02" type="subtitlePt"/>
12    <text id="lyrics-part3a" src="versosEn.html#versos03" type="subtitleEn"/>
13    <text id="lyrics-part3b" src="versosPt.html#versos03" type="subtitlePt"/>
14  </composite>
```

```
15  </root>
```

Figura 5:10 - Documento XML com a cópia de uma composição sendo processada.

Analisando em seguida o corpo do *template*, são geradas três transformadas XSLT para serem aplicadas, em seqüência, ao documento da Figura 5:10, gerando a composição processada da Figura 5:11. Essas transformadas seguem o padrão XSLT (W3C, 1999d) e podem ser aplicadas ao documento utilizando qualquer processador dessa linguagem.

```
01    <composite id="C1">
02      <audio id="samba" region="audioRegion1" src="coisadepeleBeth.wav">
03        <area begin="8.4s" end="18.5s" id="part1"/>
04        <area begin="18.5s" end="29s" id="part2"/>
05        <area begin="29s" end="39.5s" id="part3"/>
06      </audio>
07      <switch id="Switch1">
08        <text id="lyrics-part1a" src="versosEn.html#versos01"/>
09        <bindRule component="lyrics-part1a" rule="RuleEnUS"/>
10        <text id="lyrics-part1b" src="versosPt.html#versos01"/>
11        <bindRule component="lyrics-part1b" rule="RuleEnPT"/>
12      </switch>
13      <switch id="Switch2">
14        <text id="lyrics-part2a" src="versosEn.html#versos02"/>
15        <bindRule component="lyrics-part2a" rule="RuleEnUS"/>
16        <text id="lyrics-part2b" src="versosPt.html#versos02"/>
17        <bindRule component="lyrics-part2b" rule="RuleEnPT"/>
18      </switch>
19      <switch id="Switch3">
20        <text id="lyrics-part3a" src="versosEn.html#versos03"/>
21        <bindRule component="lyrics-part3a" rule="RuleEnUS"/>
22        <text id="lyrics-part3b" src="versosPt.html#versos03"/>
23        <bindRule component="lyrics-part3b" rule="RuleEnPT"/>
24      </switch>
25      <linkBase>
26        <link xconnector="starts.xml">
27          <bind component="samba" port="part1" role="on_x_presentation_begin"/>
28          <bind component="Switch1" role="start_y"/>
29        </link>
30        <link xconnector="finishes.xml">
31          <bind component="samba" port="part1" role="on_x_presentation_end"/>
32          <bind component="Switch1" role="stop_y"/>
33        </link>
34        <link xconnector="starts.xml">
35          <bind component="samba" port="part2" role="on_x_presentation_begin"/>
36          <bind component="Switch2" role="start_y"/>
37        </link>
38        <link xconnector="finishes.xml">
39          <bind component="samba" port="part2" role="on_x_presentation_end"/>
40          <bind component="Switch2" role="stop_y"/>
41        </link>
```

```
42        <link xconnector="starts.xml">
43          <bind component="samba" port="part3" role="on_x_presentation_begin"/>
44          <bind component="Switch3" role="start_y"/>
45        </link>
46        <link xconnector="finishes.xml">
47          <bind component="samba" port="part3" role="on_x_presentation_end"/>
48          <bind component="Switch3" role="stop_y"/>
49        </link>
50      </linkBase>
51    </composite>
```

Figura 5:11 - Composição gerada pelo processador de *templates*.

A primeira transformada, chamada *body XSLT*, é gerada a partir dos elementos *xsl:stylesheet* definidos como filhos diretos do elemento *body* do *template*, ou seja, ela especifica relacionamentos de inclusão. A transformada do exemplo está ilustrada na Figura 5:12, onde algumas partes são constantes para todas as transformadas desse tipo: as linhas 2-6 indicam a cópia do elemento raiz (*root*); as linhas 7-9 definem a cópia de todos os elementos filhos da composição; e as linhas 10-20 definem a cópia do elemento representando a composição (retirando seu atributo *xtemplate*), cujo conteúdo é formado a partir de outros *xsl:template* da transformada (pelo comando da linha 17) e pelo *xsl:template* nomeado *mainTemplate* (chamado na linha 18). O restante da transformada é obtido a partir da especificação do *template*, sendo todos os elementos do tipo *xsl:resource* substituídos por um elemento *xsl:element*, para geração de elementos referentes aos tipos declarados no vocabulário do *template*. Por exemplo, na linha 24 foi definido o *xsl:element* referente ao recurso do tipo (*type*) *switch*, cujo nome (*name*) é o tipo de conteúdo (*ctype*) desse recurso: *switch*. Um atributo *type* para esse novo elemento é gerado automaticamente, a partir de seu tipo (linha 25). Após a substituição de todos os *xsl:resource* pelos *xsl:element* correspondentes, obtém-se o *xsl:template mainTemplate* (linhas 21-43) que contém todos os elementos definidos na especificação do *template* que não são do tipo *xsl:template*. Os elementos do tipo *xsl:template* são adicionados diretamente ao elemento *xsl:stylesheet* da transformada (linhas 44-74). Após o processamento da transformada *body XSLT* a composição gerada é similar à composição da Figura 5:11 excluindo o elemento *linkBase*.

```
01  <xsl:stylesheet ...>
02    <xsl:template match="/">
03      <xsl:copy>
04        <xsl:apply-templates/>
05      </xsl:copy>
```

```
06    </xsl:template>
07    <xsl:template match="/*/*/*">
08      <xsl:copy-of select="."/>
09    </xsl:template>
10    <xsl:template match="/*/*">
11      <xsl:copy>
12        <xsl:for-each select="@*">
13          <xsl:if test="not(name(.) = 'xtemplate')">
14            <xsl:copy/>
15          </xsl:if>
16        </xsl:for-each>
17        <xsl:apply-templates/>
18        <xsl:call-template name="mainTemplate"/>
19      </xsl:copy>
20    </xsl:template>
21    <xsl:template name="mainTemplate">
22      <xsl:for-each select="//*[@type='subtitleEn']">
23        <xsl:variable name="i" select="position()"/>
24        <xsl:element name="switch">
25          <xsl:attribute name="type">switch</xsl:attribute>
26          <xsl:attribute name="id">Switch
27            <xsl:value-of select="$i"/>
28          </xsl:attribute>
29          <xsl:copy>
30            <xsl:for-each select="text()|@*">
31              <xsl:copy/>
32            </xsl:for-each>
33          </xsl:copy>
34          <xsl:call-template name="bindRuleElement">
35            <xsl:with-param name="component" select="./@id"/>
36            <xsl:with-param name="rule">RuleEnUS</xsl:with-param>
37          </xsl:call-template>
38          <xsl:call-template name="ptSwitchElement">
39            <xsl:with-param name="i" select="$i"/>
40          </xsl:call-template>
41        </xsl:element>
42      </xsl:for-each>
43    </xsl:template>
44    <xsl:template match="/*/*/*">
45      <xsl:if test="not(./@type='subtitleEn') and not(./@type='subtitlePt')">
46        <xsl:copy-of select="."/>
47      </xsl:if>
48    </xsl:template>
49    <xsl:template name="ptSwitchElement">
50      <xsl:param name="i"/>
51      <xsl:for-each select="/*/*/child::*[@type='subtitlePt'][$i]">
52        <xsl:copy>
53          <xsl:for-each select="text()|@*">
54            <xsl:copy/>
55          </xsl:for-each>
56        </xsl:copy>
```

```
57        <xsl:call-template name="bindRuleElement">
58          <xsl:with-param name="component" select="./@id"/>
59          <xsl:with-param name="rule">RuleEnPT</xsl:with-param>
60        </xsl:call-template>
61      </xsl:for-each>
62    </xsl:template>
63    <xsl:template name="bindRuleElement">
64      <xsl:param name="component"/>
65      <xsl:param name="rule"/>
66      <xsl:element name="bindRule">
67        <xsl:attribute name="component">
68          <xsl:value-of select="$component"/>
69        </xsl:attribute>
70        <xsl:attribute name="rule">
71          <xsl:value-of select="$rule"/>
72        </xsl:attribute>
73      </xsl:element>
74    </xsl:template>
75  </xsl:stylesheet>
```

Figura 5:12 - Exemplo de transformada *body XSLT*.

A segunda transforma a ser aplicada é chamada *link XSLT*. Essa transformada, gerada a partir da especificação do elemento *linkBase* do corpo do *template*, corresponde à inserção dos relacionamentos baseados em conectores definidos pelo *template*. Na Figura 5:13, a primeira parte da transformada (linhas 01-20) é similar à transformada da Figura 5:12, sendo que a chamada da linha 18 passa a apontar para o *xsl:template linkBaseTemplate*. Esse *xsl:template* (linhas 21-25) adiciona um elemento *linkBase* à composição (linhas 22-24), cujo conteúdo (*links*) é definido pelo *xsl:template linkTemplate*, que é gerado a partir do *template*, como se segue. Todo conteúdo da transformada especificada pelo elemento *linkBase* no cabeçalho do *template* é copiado para *linkTemplate*, sendo que:

- todo elemento *xsl:link* é transformado em um elemento *xsl:element* (como nas linha 29-72) para geração de um *link* na base de elos da composição, sendo adicionados automaticamente os atributos *xconnector* (linha 30) e *type* (linha 31), obtidos a partir do vocabulário do *template*;
- para cada elemento *xsl:bind* de *xsl:link* é gerado um elemento *xsl:for-each* (como nas linhas 32-51 e 52-71), cujo atributo *select* (linhas 32 e 52) corresponde ao atributo *select* do *xsl:bind*. Para cada *xsl:for-each* define-se um elemento com nome *bind* (linhas 33-50 e

53-70), com um atributo *role* igual ao atributo *role* de *xsl:bind* (linhas 54 e 34). Além do atributo *role*, o elemento *bind* terá os atributos *component* e *port*, caso o elemento referenciado por esse *bind* seja do tipo *port* ou *area* (linhas 36-43 e 56-63); ou terá somente o atributo *component* caso contrário (linhas 44-48 e 64-68). O atributo *component* (ou a combinação dos atributos *component* e *port*), como visto na Seção 2.4, mapeiam um componente do documento ao papel do conector (atributo *role*) definido pelo *bind*.

Na Figura 5:13 temos, portando, a definição de um *loop* do tipo *xsl:for-each* (linhas 27-117) - especificado pelo arquivo XTemplate - que define dois tipos de elos, referenciando dois conectores do vocabulário do *template*, o primeiro nas linhas 29-72 e o segundo nas linhas 73-116.

```
01  <xsl:stylesheet ...>
02    <xsl:template match="/">
03      <xsl:copy>
04        <xsl:apply-templates/>
05      </xsl:copy>
06    </xsl:template>
07    <xsl:template match="/*/*/*">
08      <xsl:copy-of select="."/>
09    </xsl:template>
10    <xsl:template match="/*/*">
11      <xsl:copy>
12        <xsl:for-each select="@*">
13          <xsl:if test="not(name(.) = 'xtemplate')">
14            <xsl:copy/>
15          </xsl:if>
16        </xsl:for-each>
17        <xsl:apply-templates/>
18        <xsl:call-template name="linkBaseTemplate"/>
19      </xsl:copy>
20    </xsl:template>
21    <xsl:template name="linkBaseTemplate">
22      <xsl:element name="linkBase">
23        <xsl:call-template name="linkTemplate"/>
24      </xsl:element>
25    </xsl:template>
26    <xsl:template name="linkTemplate">
27      <xsl:for-each select="child::*[@type='song']/child::*[@type='track']">
28        <xsl:variable name="i" select="position()"/>
29        <xsl:element name="link">
30          <xsl:attribute name="xconnector">starts.xml</xsl:attribute>
31          <xsl:attribute name="type">L</xsl:attribute>
32          <xsl:for-each select="current()">
33            <xsl:element name="bind">
```

```
34              <xsl:attribute name="role">on_x_presentation_begin</xsl:attribute>
35              <xsl:choose>
36                <xsl:when test="name(.)='port' or name(.)='area'">
37                  <xsl:attribute name="component">
38                    <xsl:value-of select="../@id"/>
39                  </xsl:attribute>
40                  <xsl:attribute name="port">
41                    <xsl:value-of select="@id"/>
42                  </xsl:attribute>
43                </xsl:when>
44                <xsl:otherwise>
45                  <xsl:attribute name="component">
46                    <xsl:value-of select="@id"/>
47                  </xsl:attribute>
48                </xsl:otherwise>
49              </xsl:choose>
50            </xsl:element>
51          </xsl:for-each>
52          <xsl:for-each select="//*[@type='switch'][$i]">
53            <xsl:element name="bind">
54              <xsl:attribute name="role">start_y</xsl:attribute>
55              <xsl:choose>
56                <xsl:when test="name(.)='port' or name(.)='area'">
57                  <xsl:attribute name="component">
58                    <xsl:value-of select="../@id"/>
59                  </xsl:attribute>
60                  <xsl:attribute name="port">
61                    <xsl:value-of select="@id"/>
62                  </xsl:attribute>
63                </xsl:when>
64                <xsl:otherwise>
65                  <xsl:attribute name="component">
66                    <xsl:value-of select="@id"/>
67                  </xsl:attribute>
68                </xsl:otherwise>
69              </xsl:choose>
70            </xsl:element>
71          </xsl:for-each>
72        </xsl:element>
73        <xsl:element name="link">
74          <xsl:attribute name="xconnector">finishes.xml</xsl:attribute>
75          <xsl:attribute name="type">P</xsl:attribute>
76          <xsl:for-each select="current()">
77            <xsl:element name="bind">
78              <xsl:attribute name="role">on_x_presentation_end</xsl:attribute>
79              <xsl:choose>
80                <xsl:when test="name(.)='port' or name(.)='area'">
81                  <xsl:attribute name="component">
82                    <xsl:value-of select="../@id"/>
83                  </xsl:attribute>
84                  <xsl:attribute name="port">
```

```
85                    <xsl:value-of select="@id"/>
86                  </xsl:attribute>
87                </xsl:when>
88                <xsl:otherwise>
89                  <xsl:attribute name="component">
90                    <xsl:value-of select="@id"/>
91                  </xsl:attribute>
92                </xsl:otherwise>
93              </xsl:choose>
94            </xsl:element>
95          </xsl:for-each>
96          <xsl:for-each select="//*[@type='switch'][$i]">
97            <xsl:element name="bind">
98              <xsl:attribute name="role">stop_y</xsl:attribute>
99              <xsl:choose>
100               <xsl:when test="name(.)='port' or name(.)='area'">
101                 <xsl:attribute name="component">
102                   <xsl:value-of select="../@id"/>
103                 </xsl:attribute>
104                 <xsl:attribute name="port">
105                   <xsl:value-of select="@id"/>
106                 </xsl:attribute>
107               </xsl:when>
108               <xsl:otherwise>
109                 <xsl:attribute name="component">
110                   <xsl:value-of select="@id"/>
111                 </xsl:attribute>
112               </xsl:otherwise>
113             </xsl:choose>
114           </xsl:element>
115         </xsl:for-each>
116       </xsl:element>
117     </xsl:for-each>
118   </xsl:template>
119 </xsl:stylesheet>
```

Figura 5:13 - Exemplo de transformada *link XSLT*.

Terminado o processamento da segunda transformada, a composição já possui a semântica definida pelos relacionamentos herdados do *template*. Entretanto, mais um passo é necessário: checar as restrições. Para isso a transformada *constraint XSLT* é gerada a partir das restrições do *template*, especificadas pela cardinalidade e aninhamento dos elementos do vocabulário, e pelas restrições definidas por elementos *constraint*. A Figura 5:14 ilustra uma transformada desse tipo, onde as linhas 2-11 definem a cópia da composição de entrada (eliminando-se os atributos *type* e *label*) e as linhas 12-29 especificam as restrições que devem ser verificadas no *template*. Caso essas restrições não sejam válidas, o processador de *templates* gera uma mensagem de erro. No exemplo,

apenas restrições sobre cardinalidade e aninhamento são validadas (a partir de testes e mensagens de erro geradas automaticamente pelo processador de *templates*), já que o *template* não define restrições adicionais.

```
01  <xsl:stylesheet ...>
02    <xsl:template match="/*//*">
03      <xsl:copy>
04        <xsl:for-each select="@*">
05          <xsl:if test="not(name(.) = 'type' or name(.) = 'label')">
06            <xsl:copy/>
07          </xsl:if>
08        </xsl:for-each>
09        <xsl:apply-templates/>
10      </xsl:copy>
11    </xsl:template>
12    <xsl:template match="/">
13      <xsl:apply-templates/>
14      <xsl:if test="count(//*[@type='song']) &gt; 1">
15        <xsl:message terminate="no">Max number of elements of type song is
1</xsl:message>
16      </xsl:if>
17      <xsl:if test="1 &gt; count(//*[@type='song'])">
18        <xsl:message terminate="no">Min number of elements of type song is
1</xsl:message>
19      </xsl:if>
20      <xsl:if      test="not(count(//*[@type='song']/*[@type='track'][1])      =
count(//*[@type='song']))">
21        <xsl:message terminate="no">Min number of elements of type /song/track
is 1</xsl:message>
22      </xsl:if>
23      <xsl:if  test="not((count(//*[@type='switch']/*[@type='subtitleEn'][1])  =
count(//*[@type='switch']))                                                  and
(not(//*[@type='switch']/*[@type='subtitleEn'][2])))">
24        <xsl:message  terminate="no">Min/Max  number  of  elements  of  type
/switch/subtitleEn is 1/1</xsl:message>
25      </xsl:if>
26      <xsl:if  test="not((count(//*[@type='switch']/*[@type='subtitlePt'][1])  =
count(//*[@type='switch']))                                                  and
(not(//*[@type='switch']/*[@type='subtitlePt'][2])))">
27        <xsl:message  terminate="no">Min/Max  number  of  elements  of  type
/switch/subtitlePt is 1/1</xsl:message>
28      </xsl:if>
29    </xsl:template>
30  </xsl:stylesheet>
```

Figura 5:14 - Exemplo de transformada *constraint XSLT*.

Finalmente, aplicada a transformada *constraint XSLT*, obtém-se a composição processada. Essa composição está ilustrada na Figura 5:11, onde é possível observar os relacionamentos de inclusão e os relacionamentos por meio de conectores obtidos pelas transformadas da Figura 5:12 e Figura 5:13,

respectivamente. Essa composição pode, então, substituir a composição original do documento.

6
Trabalhos Relacionados

Este capítulo apresenta trabalhos relacionados com os principais temas abordados por esta dissertação. São descritos trabalhos relativos a: *templates* de composição, processamento de documentos XML, *framework* para compiladores, compiladores XML, extensões à SMIL e conversão entre modelos de documento hipermídia.

6.1. *Template* de Composição

Celentano e Gaggi (Celentano & Gaggi, 2003) descrevem a geração de apresentações multimídia baseada em *templates*. Essas apresentações são construídas automaticamente por uma aplicação que realiza consultas a um banco de dados. *Templates* definem o comportamento da apresentação e a sincronização entre seus objetos. Dessa forma, *templates* são especificados uma única vez e cada nova apresentações é gerada a partir dos dados (instâncias de objetos) obtidos de um repositório.

A Figura 6:1 ilustra um documento que representa um noticiário, composto por três artigos, no ambiente de autoria gráfica do sistema, chamado LAMP (*LAboratory for Multimedia presentations Prototyping*). Uma música de fundo (*soundtrack*) é apresentada continuamente durante a exibição em seqüência dos artigos, sendo cada artigo composto por uma narração ($news_i$), um vídeo ($vídeo_i$) e uma legenda ($caption_i$). Ao término da exibição do último artigo, a música de fundo é substituída por um *jingle* e os créditos (*credits*) são apresentados. Os relacionamentos entre os componentes do documento são definidos por arestas referenciando um dos cinco tipos de relacionamentos oferecidos pelo sistema (Celentano & Gaggi, 2003).

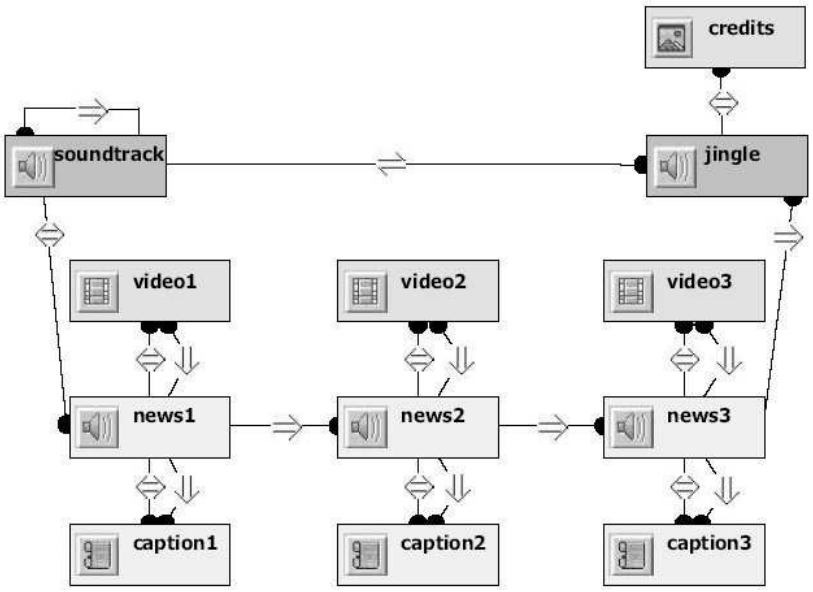


Figura 6:1 - Exemplo de um documento representando um noticiário no sistema LAMP.

Analisando a Figura 6:1, pode-se observar uma estrutura recorrente na definição dos artigos, possibilitando, dessa forma, a especificação de um *template*. O *template* para definição de artigos é ilustrado na Figura 6:2, também visualizado no ambiente gráfico de autoria do LAMP. Na definição de *templates*, uma composição pode ser do tipo *stencil*, que determina um *template* de composição[49] para geração de composições de um mesmo tipo (a partir de uma consulta a um banco de dados). Na Figura 6:2, a composição *article* é do tipo *stencil*, sendo permitido que seus relacionamentos definam, além de um tipo, um atributo referenciando qual tupla retornada pela consulta ao banco de dados deve ser relacionada. Esse atributo pode ter os seguinte valores: *first* (primeira tupla), *last* (última tupla), *next* (próxima tupla) ou *all* (todas tuplas) - valor padrão.

[49] O termo *template de composição* não é o utilizado pelos autores do artigo, sendo utilizado neste texto para facilitar a comparação com as definições desta dissertação.

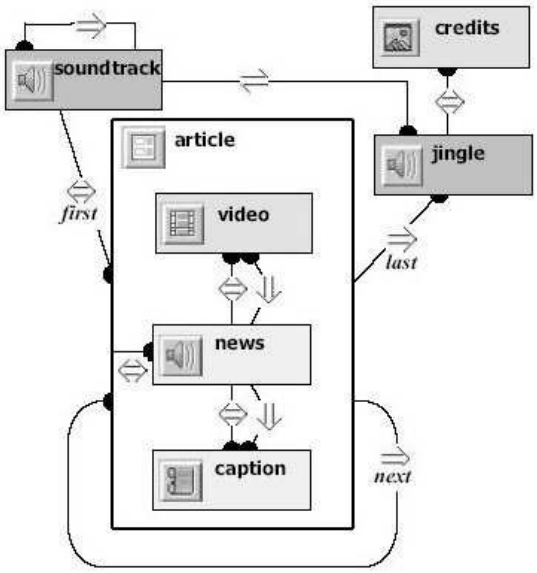


Figura 6:2 - *Template* para artigos de um noticiário.

Baseada no mesmo modelo hipermídia do ambiente de autoria gráfica, foi desenvolvida uma linguagem no padrão XML para especificação de documentos (apresentações) e *templates*. Foi possível a definição de uma única linguagem para a autoria tanto de documentos quanto de *templates*, com a seguinte restrição: elementos do tipo *stencil* somente podem ser declarados na especificação de *templates*. Esse elemento é utilizado para a geração das composições de uma apresentação (ou documento) após uma consulta a um banco de dados. O "processamento de *templates*" consulta um banco de dados e, a partir da resposta, gera composições referentes a elementos *stencil* e, também, os relacionamentos definidos no *template* que envolvem esses elementos.

A proposta apresentada é, em vários aspectos, similar a de *templates* de composição descrita neste trabalho. Suas principais contribuições são: a consulta a bancos de dados para geração de documentos; o oferecimento de um ambiente gráfico para autoria de *templates*; e o uso da mesma linguagem para definição tanto de documentos quanto de *templates*. Entretanto, quando comparado ao NCM, o modelo hipermídia no qual o LAMP se baseia é pouco expressivo (entre outras limitações, apenas cinco tipos de relacionamentos são permitidos), o que facilitou a definição da linguagem para autoria de documentos e *templates*. Além disso, o processador de *templates* do LAMP é baseado em um algoritmo específico que não faz uso de padrões relacionados à XML, enquanto NCL

estende o padrão XSLT para definição de seus *templates* e utiliza processadores de XSLT no processamento de *templates*.

6.2. Processamento de documentos XML

Existem duas principais abordagens para o processamento de documentos XML. A primeira se baseia no caminhamento por uma estrutura de dados em árvore, padronizada em *Document Object Model* - DOM (W3C, 2000a). A segunda abordagem é baseada em eventos (disparados durante a leitura de documentos e tratados por processadores), seguindo o padrão *de facto* SAX (SAX, 2002). Esses dois padrões, que são independentes de linguagem de programação, possuem uma API definida para a linguagem Java: *Java API for XML Processing* - JAXP (Sun, 2002). A Figura 6:3 e a Figura 6:4 ilustram, respectivamente, a definição da API Java para DOM e para SAX[50]. JAXP foi a API utilizada nos compiladores desenvolvidos neste trabalho, por meio das implementações Xerces2 (Apache XML, 2002b) e Xalan (Apache XML, 2002a) para o processamento de arquivos XML e o processamento de folhas de estilo XSLT, respectivamente.

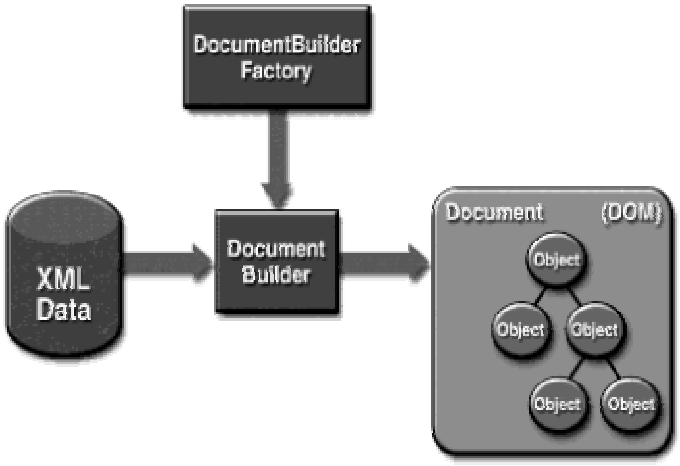


Figura 6:3 - API DOM de JAXP.

A API para DOM define a classe *DocumentBuilderFactory*, a ser utilizada para obter uma instância de *DocumentBuilder*. Essa instância gera, a partir de

[50] As figuras foram extraídas do tutorial "*The J2EE 1.4 Tutorial*", disponível em http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html.

dados XML, um objeto (instância da classe *Document*) em conformidade com a especificação DOM.

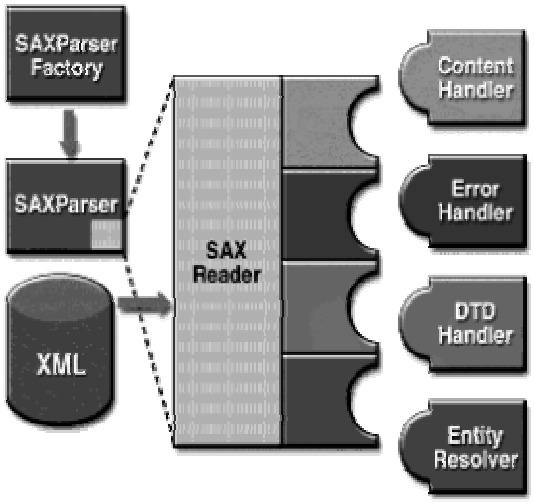


Figura 6:4 - API SAX de JAXP.

O processamento de um documento seguindo a API SAX se inicia pela classe *SAXParserFactory*, que gera uma instância da classe *SAXParser*. Essa classe *parser* possui um leitor SAX (objeto da classe *SAXReader*), que lê um documento XML e invoca métodos (*callback methods*) implementados na aplicação usuária da API (processador XML). Esses métodos são definidos pelas interfaces *ContentHandler*, *ErrorHandler*, *DTDHandler* e *EntityResolver*. Um objeto *DefaultHandler* (não ilustrado na Figura 6:4) implementa todas essas interfaces com métodos "nulos". Aplicações devem herdar dessa classe, implementando somente os métodos de interesse. A interface *ContentHandler* define os principais métodos para processamento de documentos, como: *startDocument* e *endDocument*, chamados no início e término de processamento de um documento, e *startElement* e *endElement*, que são chamados a cada início e término de processamento de um elemento XML. Essa interface também define métodos para tratamento do texto presente como conteúdo de elementos XML. A interface *ErrorHandler* define os métodos *error*, *fatalError* e *warning*, para tratamento dos diversos tipos de erros que podem ocorrer na análise de um documento XML. Os métodos das interfaces *DTDHandler* e *EntityResolver* raramente precisam ser implementados, sendo chamados para tratar especificidades de DTDs e para computar a localização de objetos referenciados pelo documento.

A abordagem DOM possui a vantagem de oferecer acesso a todo documento XML ("*random access*"), já que a árvore que o representa é construída e pode tanto ser consultada quanto alterada. Dessa forma, essa abordagem é recomendada quando o conteúdo do documento como um todo deve ser analisado (de forma não linear), ou quando se deseja alterar o conteúdo de um documento (de forma interativa). Por outro lado, por construir uma representação do documento XML, tipicamente uma aplicação que utilize DOM requer mais recursos de memória.

Quando o acesso serial ("*serial access*") ao conteúdo do documento é suficiente, ou quando somente uma parte desse documento necessita ser analisada, SAX torna-se a opção recomendada. Como nenhuma estrutura intermediária é construída em um processador SAX, quando comparado com processadores DOM, normalmente tem-se um uso mais eficiente de memória e uma execução mais rápida. Entretanto, SAX é um padrão que não apresenta a noção de estado ("*stateless*"), já que os métodos são chamados linearmente durante a leitura do documento, independente do que já foi processado.

Para ser genérica, a estratégia adotada pelo *meta-framework* de compiladores é baseada no padrão DOM. Dessa forma, é permitido um acesso não linear aos elementos quando um documento é compilado. Por exemplo, um compilador pode consultar dados a respeito do pai do elemento sendo computado ou obter a quantidade de filhos de um determinado tipo que esse elemento possui. Isso, a princípio, não é possível utilizando SAX. Outra razão para escolha dessa estratégia é possibilitar o uso de processadores XSLT, já que a representação de um elemento como um nó de uma árvore DOM é necessária para se aplicar transformadas.

6.3. *Framework* e Compiladores

Diversos estudos abordam a geração automática de parte do código para compiladores de linguagens de programação. Ferramentas como Lex/Flex e Yacc/Bison (Levine, 1995) automatizam parte das tarefas de construção de compiladores, utilizando conceitos precisos para expressões regulares e para a gramática de linguagens. *Java Compiler Compiler* - JavaCC (JavaCC, 2003) é uma ferramenta que segue esse princípio, ou seja, lê uma especificação gramatical

de uma linguagem de programação e a converte em um programa Java, que reconhece padrões nessa gramática. Dessa forma, é gerado automaticamente um *parser* em Java para essa linguagem e são oferecidos mecanismos para construção de árvores sintáticas e tratamento de erros, possibilitando o desenvolvimento de compiladores.

Uma abordagem semelhante é proposta em *ANother Tool for Language Recognition* - ANTLR (ANTLR, 2005): uma ferramenta baseada em um *framework* para construção de compiladores e tradutores. Diferente de JavaCC, é oferecido suporte para geração de código em outras linguagens além de Java: C#, Python e C++. ANTLR oferece mecanismos para construção e análise de árvores sintáticas abstratas (*Abstract Syntax Tree* - AST); permite tratamento de erros; e, segundo os desenvolvedores da ferramenta, gera um código de fácil compreensão por programadores.

Essas propostas para geração automática de código para compiladores não estão relacionadas com XML. Assim, estendendo ANTLR, são apresentadas, em (Widemann et al., 2003), duas ferramentas baseadas em XML para geração automática de compiladores: XANTLR e TDOM. A Figura 6:5 ilustra, de forma simplificada, essas ferramentas - implementadas na linguagem Java. A ferramenta XANTLR, definida como um "ANTLR com um pré-processador", recebe como entrada a especificação da gramática de uma linguagem e gera, automaticamente, duas saídas: (1) um código para *parse* de documentos nessa linguagem, que estende o código gerado por ANTLR, ao definir uma representação em XML para a árvore sintática e um conjunto de eventos SAX para o processamento dessa árvore; e (2) uma DTD que reflete exatamente a sintaxe abstrata da linguagem. TDOM, por sua vez, recebe como entrada a DTD gerada por XANTLR e constrói, automaticamente, um conjunto de classes Java representando a sintaxe da linguagem. Uma classe é gerada para cada *elemento* definido pela DTD, contendo implementações de funções para *parse*. Essas funções são chamadas pelos eventos SAX gerados pela árvore XANTLR para construção da árvore TDOM, e podem ser re-implementados por compiladores de documentos na linguagem especificada.

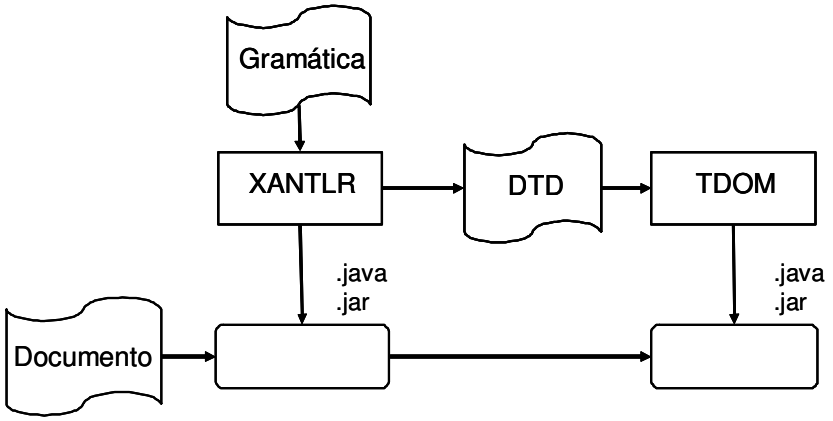


Figura 6:5 - Ferramentas XANTLR e TDOM.

A estrutura de dados em árvore de TDOM é formada por objetos das classes geradas pelos *elementos* definidos na DTD. Dessa forma, ao contrário de árvores DOM e AST, essa árvore possui objetos com tipos definidos. Essa abordagem, segundo os autores da proposta, é mais expressiva, pois conceitos como herança podem ser aplicados aos nós da árvore. A estrutura TDOM não implementa a interface definida pelo padrão DOM, porém, métodos para geração de árvores DOM a partir de TDOM, e vice-versa, são oferecidos.

Os conhecimentos adquiridos na área de compiladores de linguagens de programação devem, sempre que possível, ser aproveitados. Entretanto, as estruturas geradas por compiladores ou *parsers* tradicionais podem não ser as mais adequadas para outras aplicações. Por exemplo, a representação do código de um programa em árvores AST é adequada para compiladores. Porém, como essa estrutura geralmente não reflete diretamente os construtores presentes no nível de programação, possuindo aspectos da gramática da linguagem (que são muitas vezes complexos), a sua utilização em aplicações que apenas analisam o código de uma linguagem nem sempre é trivial (Badros, 2000). Algumas dessas aplicações optam, então, por fazer uma simples análise léxica do código ao invés de reusar um *parser* e analisar uma árvore AST. Assim, cada uma dessas aplicações apresenta uma implementação proprietária para o processamento do código fonte, impedindo, por exemplo, o reuso e o intercâmbio.

Para solucionar esse problema e permitir que ferramentas de software possam facilmente analisar e manipular códigos fonte, é necessário um formato "universal" para representar a estrutura de um programa. O padrão XML, por possuir os requisitos necessários para a definição dessa estrutura, foi utilizado na

proposta de uma linguagem baseada em XML para descrever o código (ou para representar a estrutura) de programas Java: JavaML (Badros, 2000).

Ao utilizar XML, a representação de um código fonte deixa de ser simplesmente baseada em caracteres e pode refletir mais diretamente a estrutura do programa. Uma abordagem para definição de JavaML poderia utilizar um mapeamento direto de AST para XML, mas, como comentado, a estrutura AST apresenta diversos detalhes da gramática da linguagem que iriam gerar uma representação XML muito complexa. Outra abordagem, mais simples, seria apenas adicionar marcações XML ao código original. Nesse caso, tem-se uma agregação de valor ao código, que pode ser útil em diversas aplicações[51]. JavaML foi definida estendendo essa última abordagem. Buscando aproveitar as características principais de XML, a definição da linguagem JavaML procurou modelar os construtores de uma linguagem de programação independente de sua sintaxe, utilizando da melhor forma possível os elementos e atributos XML para representação de *conceitos* do código. A Figura 6:6 ilustra um código para uma classe em Java, cuja representação em JavaML encontra-se na Figura 6:7.

```
01  import java.applet.*;
02  import java.awt.*;
03  public class FirstApplet extends Applet {
04    public void paint(Graphics g) {
05      g.drawString("FirstApplet", 25, 50);
06    }
07  }
```

Figura 6:6 - Especificação da classe *FirstApplet* em Java.

```
01   <java-source-program name="FirstApplet.java">
02    <import module="java.applet.*"/>
03    <import module="java.awt.*"/>
04    <class name="FirstApplet" visibility="public">
05     <superclass class="Applet"/>
06     <method name="paint" visibility="public" id="meth-15">
07      <type name="void" primitive="true"/>
08      <formal-arguments>
09       <formal-argument name="g" id="frmarg-13">
10         <type name="Graphics"/></formal-argument>
11      </formal-arguments>
12      <block>
13       <send message="drawString">
14        <target><var-ref name="g" idref="frmarg-13"/></target>
```

[51] Exemplos de documentos XML nessas duas abordagens, assim como uma análise de suas vantagens e desvantagens, podem ser encontrados em (Badros, 2000).

```
15        <arguments>
16         <literal-string value="FirstApplet"/>
17         <literal-number kind="integer" value="25"/>
18         <literal-number kind="integer" value="50"/>
19        </arguments>
20       </send>
21      </block>
22     </method>
23    </class>
24   </java-source-program>
```

Figura 6:7 - Especificação de *FirstApplet* em JavaML.

Como ilustrado pelo exemplo, arquivos em JavaML, ao mesmo tempo que podem ser utilizados pelas mais diversas ferramentas de análise, mantêm o código legível para programadores. A conversão entre classes Java e arquivos JavaML foi implementada a partir do compilador Jikes (IBM, 1998) para Java; uma folha de estilo XSLT permite a conversão reversa.

A grande vantagem de JavaML é permitir que as diversas aplicações que a utilizem usufruam das ferramentas e padrões XML existentes. Transformadas XSLT podem ser aplicadas a documentos JavaML para, por exemplo, alterar nomes de métodos de forma precisa (editores de texto simples não conseguem distinguir entre um nome que representa um método e uma ocorrência desse nome em outro contexto), e padrões como XSL (W3C, 2001a) podem ser utilizados para a apresentação dos dados XML. A abordagem utilizada em JavaML pode ser seguida para descrever, em XML, códigos fonte de outras linguagens. Assim, transformadas nesses arquivos XML podem ser suficientes para permitir o intercâmbio de códigos fonte entre linguagens de programação.

A definição de JavaML mostra que, para se obter as vantagens de XML, nem sempre existe um mapeamento direto entre linguagens de programação e suas representações em XML. O código fonte ou árvores AST não podem ser utilizados diretamente para gerar essa representação, indicando que as tecnologias utilizadas pelos *frameworks* para compiladores de linguagens de programação ainda precisam ser melhor examinadas para sua aplicação no domínio XML. Dessa forma, ainda é necessário analisar como essas tecnologias, geralmente complexas, podem ser utilizadas para refinar o *meta-framework*, mantendo uma de suas principais características: a simplicidade.

6.4. Compiladores XML

Para facilitar o processamento de dados em aplicações baseadas em XML, em (Li et al., 2003) é proposta uma arquitetura extensível e integrada chamada *XML Virtual Machine* (XVM). Essa arquitetura é modular e baseada em componentes, oferecendo um *framework* para desenvolvimento de aplicações. XVM considera elementos XML como objetos, cujo comportamento (*behavior*) é determinado por um componente de software a ele mapeado. A Figura 6:8 ilustra a abordagem proposta. O nome *XML Virtual Machine* foi utilizado por considerar dados XML como "instruções para processamento", e componentes como "interpretadores dessas instruções".

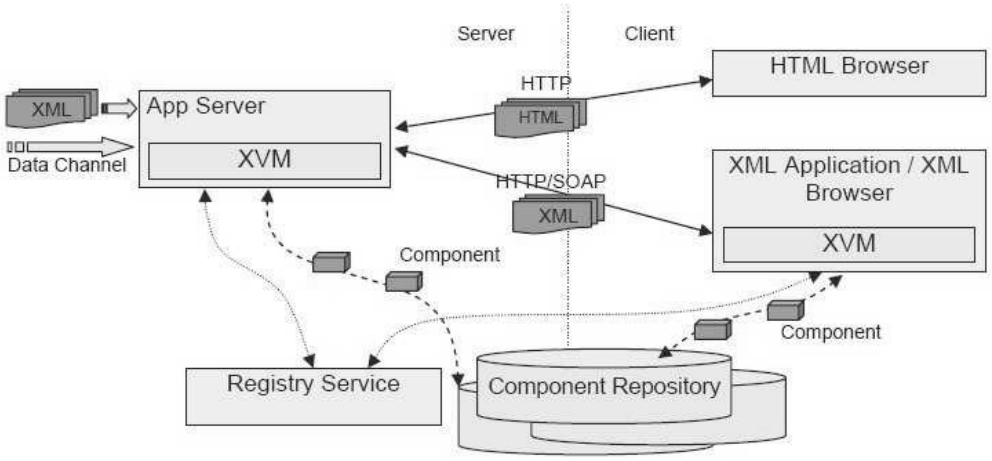


Figura 6:8 - *Framework* XVM para o desenvolvimento de aplicações com XML.

Aplicações XVM obtêm componentes dinamicamente a partir de um repositório (*Component Repository*). Um serviço de registro (*Registry Service*) é oferecido para localização de componentes, mapeando elementos XML em componentes. Utilizando esse registro, XVM associa automaticamente os elementos de uma árvore DOM a componentes, gerando uma estrutura, também em árvore, desses componentes, que determinam a semântica do documento.

Para facilitar o desenvolvimento de aplicações em diversos contextos, XVM permite que o processamento de elementos XML pelos componentes ocorra tanto no cliente (*XML Browser*) quanto no servidor (*App server*), e que um único componente seja associado a uma sub-árvore da árvore DOM (estendendo o mapeamento *um para um* entre elementos e componentes). Outra facilidade de XVM é um mecanismo para que diferentes componentes sejam associados ao mesmo elemento, dependendo do *modo de processamento*. Assim, aplicações

distintas podem obter componentes distintos ao processar o mesmo documento XML. A arquitetura simplifica a adição de novos componentes (oferecendo um mecanismo simples para atualização das aplicações) e permite que o mesmo componente seja reutilizado por diferentes aplicações.

XVM é uma arquitetura para o tratamento de documentos XML. A principal semelhança com o *meta-framework* é delegar o tratamento de elementos XML para outras entidades da "arquitetura". No *meta-framework*, essas entidades são representadas pelas classes dos compiladores, enquanto que em XVM, são representadas por componentes. Embora XVM, diferente do *meta-framework*, não tenha como foco o processamento de documentos XML como um todo, sua abordagem oferece um ponto de partida[52] para que *frameworks* de compiladores sejam utilizados na compilação de documentos XML em um ambiente distribuído, possivelmente explorando recursos de paralelismo.

6.5. Extensões à SMIL

Em (King et al., 2004) é descrita uma extensão à SMIL para permitir a definição de relacionamentos envolvendo entradas que variam constantemente com o tempo, ou seja, que possuem comportamento ("*behavior*") dinâmico em tempo-real. Relacionamentos podem ser especificados utilizando *eventos dinâmicos* ou *dependência contínua de tempo-real*. Um *comportamento* dinâmico pode gerar uma seqüência de *eventos dinâmicos*. Esses eventos são discretos e permitem disparar alguma ação ao se detectar que, por exemplo, um valor ultrapassa um certo limite. *Dependência contínua de tempo-real* considera uma propriedade *p*, de um item *A*, que depende do valor de propriedades dinâmicas de outros itens. Assim, *p* deve ser continuamente re-avaliada refletindo na exibição de *A*. Por exemplo, um objeto deve se movimentar em direção a um alvo, sendo que esse alvo também está em movimento.

Para permitir que relacionamentos como os descritos acima possam ser especificados, são necessárias duas extensões à SMIL: a habilidade de *avaliação de expressões* e um mecanismo de *definição de eventos*. Como o foco da extensão

[52] O ponto de partida se baseia na analogia entre os *componentes* XVM e as *classes* dos compiladores.

é o módulo de animação de SMIL, essas expressões podem ser aplicadas aos atributos utilizados para definição de animações - como *from*, *to*, *by*, *values* e *path* (W3C, 2001b) - e avaliadas por meio do prefixo *calc*.

Em (Goose et al., 2002) é proposta uma outra extensão para a linguagem SMIL que permite a definição e apresentação de áudios em três dimensões. A definição para o *layout* do documento é estendida, passando a ser especificada em um espaço de três dimensões (3D) ao invés de duas dimensões (2D), o que permite a definição de posições e trajetórias 3D. Baseado no elemento *regPoint* de SMIL (W3C, 2001b), foi definido o elemento *regPoint3d*, cujos atributos identificam a posição no espaço de coordenadas 3D onde objetos de áudio podem estar localizados. A posição de um objeto no espaço 3D também pode ser determinada matematicamente, pelo uso do elemento *trajectory*. Esse elemento, adicionado à SMIL, oferece um mecanismo genérico para especificar uma equação parametrizada para cada dimensão. Os atributos do elemento *trajectory* permitem que variáveis (*variables*) sejam aplicadas a uma expressão (*expression*), calculada utilizando os valores (*values*) correspondentes.

Para animação de áudios 3D, foi estendido o elemento *animate* de SMIL, que deu origem ao elemento *animateLoc*. Esse elemento especifica a animação de um objeto da posição referenciada pelo atributo *from* para a posição referenciada pelo atributo *to*. Para permitir a definição de áudios sintetizados em SMIL, foi definido um novo elemento: *tts*. Esse elemento é similar ao elemento *audio* de SMIL e possui outros atributos, como *voice* (que especifica qual padrão de voz a ser utilizado pelo sintetizador). A proposta foi validada a partir de uma arquitetura cliente/servidor na qual o servidor adapta um documento SMIL 3D gerando um documento SMIL. Os áudios 3D são convertidos em áudios estéreos, que simulam as especificações 3D do documento SMIL 3D.

Ambas as propostas apresentadas para extensão à SMIL abordam casos particulares que requerem alguma funcionalidade não oferecida pela linguagem. A linguagem X-SMIL, descrita nesta dissertação, estende SMIL em sua principal característica: a sincronização de objetos multimídia. Conectores hipermídia podem ser usados na especificação dos diversos relacionamentos descritos em (King et al., 2004), desde que ferramentas de exibição (Rodrigues, 2003) sinalizem as mudanças dos atributos e eventos referenciados pelos conectores.

É importante destacar, também, que a especificação de *layout* 2D em NCL, por exemplo, pode ser facilmente alterada para uma representação 3D. Finalmente, como foi exemplificado no Capítulo 3 (Seção 3.3), o elemento *tts*, introduzido pela proposta (Goose et al., 2002), torna-se desnecessário ao se utilizar o conceito de descritores.

6.6. Conversão entre Modelos

Yang e Yang (Yang & Yang, 2003) propõem uma ferramenta gráfica para autoria de documentos hipermídia baseada em SMIL: SMILAuthor. O processo de autoria é dividido em três partes. Primeiramente um documento SMIL é importado para o ambiente de autoria. Em seguida, são oferecidas funções de edição (como copiar, recortar e colar). Finalmente, pode-se salvar o resultado da edição em um documento SMIL. A principal diferença entre essa ferramenta e outras baseadas em SMIL (Coelho, 2004) é a utilização de um modelo em *timeline* para edição, ao invés de utilizar a própria estrutura de SMIL (composições com semântica temporal etc.). Dessa forma, no primeiro passo é aplicado um algoritmo para conversão de um documento SMIL em um modelo *timeline*, no qual todas as funções de edição podem ser aplicadas.

Segundo os autores da proposta, funções como copiar e colar são mais facilmente realizadas em uma visão baseada em *timeline*. A Figura 6:9, por exemplo, ilustra como um objeto (*object* - item *a* na figura) ou um grupo de objetos (*time zone* - item *b* da figura), que foram previamente recortados, são inseridos (colados) na linha de tempo (*timeline*).

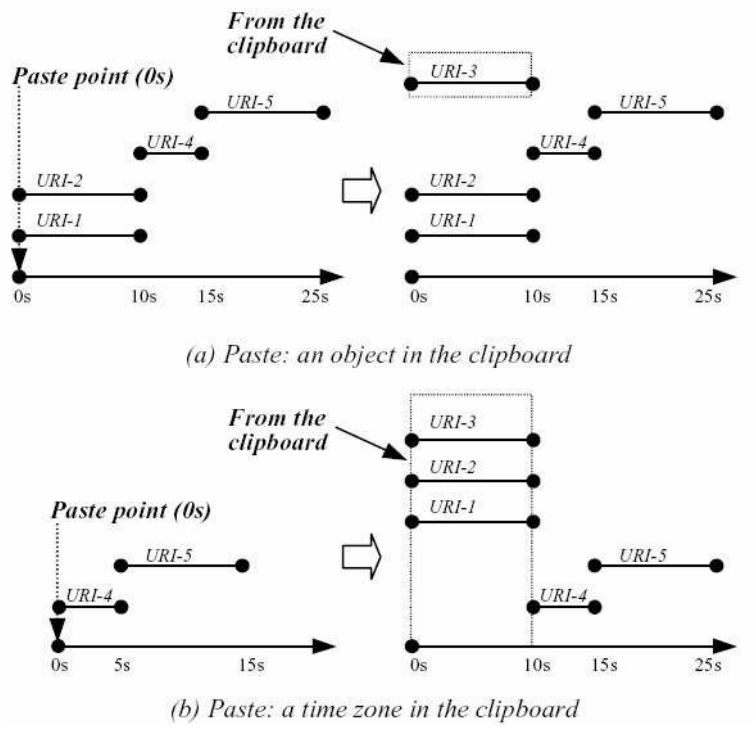


Figura 6:9 - Edição de um documento em *timeline*.

Após a edição, a estrutura em *timeline* deve ser convertida para um documento SMIL. A solução trivial para essa conversão é agrupar todos os objetos em um contêiner *par* e utilizar atributos *begin* e *dur* para refletir o comportamento temporal desses objetos em SMIL. Entretanto, esse algoritmo é apontado como problemático, já que *players* SMIL devem tratar simultaneamente todos os elementos filhos de contêineres *par*. Por isso, buscou-se maximizar o número de contêineres do tipo *seq* no documento SMIL. Assim, é utilizado um algoritmo que percorre a lista de objetos, gerando grupos de objetos com apresentação disjunta. Esses grupos de objetos geram, então, contêineres *seq*, filhos de um contêiner *par*. Esse algoritmo considera que objetos do mesmo tipo de mídia têm maior probabilidade de não serem apresentados em paralelo, sendo o mesmo verdade para objetos que referenciam a mesma região do *layout*.

Os problemas e possíveis soluções descritos pela proposta na geração de documentos SMIL, a partir de outro modelo, são sua principal contribuição. A principal limitação da proposta é a abordagem adotada para a sincronização temporal entre objetos de mídia (sincronização inter-mídia), que é baseada em *timeline*: a sincronização é realizada através do posicionamento dos componentes de uma apresentação em relação a um eixo do tempo. Essa abordagem apresenta

uma série de problemas, como a impossibilidade de modularizar uma apresentação e a dificuldade de se especificar requisitos temporais entre eventos cuja duração é variável, ou desconhecida, até o momento da execução.

Essas limitações não existem em NCL ou SMIL, que adotam um paradigma baseado em restrição e causalidade (*constraint-and-causal-based*) para sincronização. Como visto, é possível passar de um paradigma de sincronização para outro, mas, às vezes, há perda de expressividade. Isso se deve ao fato dos diversos paradigmas terem poderes de expressividade diferentes. Além de não possuir as limitações de *timeline*, o paradigma de restrição e causalidade tem maior poder de expressão (FLEXTV, 2004a).

O compilador NCL-SMIL apresenta alguns dos problemas descritos na conversão entre *timeline* e SMIL, como o número de composições *par* que são geradas. Porém, as limitações do paradigma *timeline* implicam em uma perda de informações na seqüência de conversão SMIL-*timeline* e *timeline*-SMIL. Essas informações (por exemplo: o término do objeto A gera o término do objeto B) são preservadas na conversão NCL-SMIL. Além disso, o poder de expressão dos modelos baseados em restrição e causalidade se reflete na maior complexidade do compilador NCL-SMIL.

7
Conclusão e Trabalhos Futuros

Esta dissertação apresenta a linguagem X-SMIL para autoria declarativa de documentos hipermídia. Nessa direção, o primeiro passo do trabalho foi definir NCL 2.1, refinando alguns dos módulos de NCL 2.0 e introduzindo dois novos módulos, um para a especificação de funções de custo e outro para regras de apresentação. Esses módulos aumentam tanto o reuso quanto a expressividade de NCL. Foi também refinada a linguagem XConnector, com a especificação de XConnector 2.1; e estendida a linguagem XTemplate (XTemplate 2.1), permitindo que *templates* definam relacionamentos não somente por meio de conectores, mas também de inclusão. Além disso, XTemplate 2.1 foi especificada de forma modular, permitindo que outras linguagens definam o perfil de XTemplate que utilizam. XTemplate 2.1 se torna, assim, uma das principais contribuições deste trabalho.

A partir de NCL 2.1, foi possível a especificação de X-SMIL e de seus perfis: XT-SMIL e XC-SMIL. O perfil XT-SMIL, apesar de não aumentar a expressividade de SMIL, adiciona novas funcionalidades para o reuso, flexibilizando os tipos de composição oferecidos pela linguagem. Assim, o autor não fica obrigado a definir uma hierarquia de composições dos tipos básicos (*par*, *seq* e *excl*) e um conjunto de eventos para representar relacionamentos complexos entre componentes. A definição de XT-SMIL utiliza um perfil de XTemplate que contempla relacionamento de inclusão, que somente pôde ser especificado devido à abordagem modular e às novas funcionalidades de XTemplate 2.1. O perfil XC-SMIL aumenta o reuso e expressividade de SMIL ao introduzir o conceito de conectores hipermídia nessa linguagem. Finalmente, X-SMIL combina as facilidades dos perfis XT-SMIL e XC-SMIL, permitindo, por exemplo, a definição de *templates* com relações de inclusão e por meio de conectores. Por aumentar tanto o reuso quanto a expressividade de SMIL, X-SMIL está também entre as principais contribuições desta dissertação.

Para facilitar o processamento de documentos especificados em SMIL, NCL e X-SMIL, foi definido um *framework* genérico (*meta-framework*) para o processamento de documentos XML. Baseados nesse *framework*, diversos *frameworks* de compiladores e suas instâncias (compiladores) foram implementados, permitindo, por exemplo, a conversão entre documentos NCL, SMIL e X-SMIL; e a edição e exibição de documentos especificados nessas linguagens no sistema HyperProp. A especificação do *framework* genérico de processamento e de suas instâncias são outra importante contribuição desta dissertação.

Cabe destacar, ainda, o processador de *templates*, um subproduto deste trabalho que realiza o processamento de todos os *templates* referenciados em um documento XML e, ao contrário do processador de XTemplate 2.0 (Muchaluat-Saade, 2003), oferece suporte a todas as facilidades de XSLT. O processador de *templates* implementado foi utilizado, por exemplo, na conversão de documentos XT-SMIL em documentos SMIL, demonstrando a compatibilidade entre essas linguagens.

7.1. Templates

Com o padrão MPEG-4 (Koenen, 2002), novas linguagens declarativas baseadas em XML para especificação de documentos hipermídia (ou cenas) foram definidas: XMT-A e XMT-O (ISO, 2001). Um trabalho concluído (Costa, 2005) é a definição de XT-XMT-O, uma extensão de XMT-O que introduz o conceito de *templates* nessa linguagem.

Como o conceito de *templates* de composição não é restrito à hipermídia, perfis de XTemplate podem ser definidos para outras linguagens de autoria, como linguagens de *workflow* (Thatte, 2003). Em (Silva, 2003), é apresentada uma comparação entre conceitos de hipermídia e de *workflow*, apontando como conectores hipermídia podem ser utilizados na especificação de padrões de *workflow* (Aalst, 2003) e como *templates* podem ser definidos e reutilizados no processo de autoria de documentos nessas linguagens.

Um trabalho em andamento constitui a definição de *templates* que utilizam outros *templates*. Ou seja, composições definidas como recursos em *templates*

poderão referenciar outros *templates*, reusando-os e facilitando o processo de autoria. O processador de *templates* está sendo modificado para que, após o processamento de cada *template*, seja verificado se algum componente contido na composição processada referencia outro *template*. Se houver, deve-se processar novamente esses componentes, e assim sucessivamente. Para garantir que esse algoritmo seja correto, *templates* não podem, direta ou indiretamente, se auto referenciar.

A linguagem XTemplate também está sendo refinada para permitir a definição de parâmetros. Elementos do tipo *resourceParam*, que contém os atributos *name* e *value*, poderão ser declarados como filhos de elementos *resource*, determinando que um atributo com o nome *name* e o valor *value* seja adicionado a um recurso. Os atributos *name* e *value*, quando iniciados com o caractere "$", referenciam parâmetros do *template*. Por exemplo, para adicionar, a um recurso, um atributo com o nome *region* e com valor definido pela variável "*r1*" do *template*, deve-se especificar: <*resourceParam name="region" value="$r1"*>. Parâmetros também poderão ser referenciados por elementos *xsl:templateParam*, utilizados de modo semelhante ao elemento *xsl:value-of* em transformadas XSLT. Durante o processamento de *templates*, elementos *xsl:templateParam* serão substituídos pelo valor do parâmetro do *template* indicado pelo atributo *value*. Por exemplo, para definição do atributo *systemLanguage*[53] de forma parametrizada (pela variável *langX*), pode-se especificar: <*xsl:attribute name = "systemLanguage"*> <*xsl:templateParam value="langX" /*> <*/xsl:attribute*>[54]. Como *templates* são referenciados, em documentos XML, pelo valor do atributo *xtemplate*, será utilizado o padrão de *query strings* (W3C, 2001c) para passagem de parâmetros. Por exemplo, para referenciar o *template* "*áudioComLegendasXY*" (similar ao *audioComLegendasEnPt* do Capítulo 4, mas com a escolha entre as alternativas de legendas sendo baseada nas línguas parametrizadas pelas variáveis *langX* e *langY*), deve-se declarar: *xtemplate = "audioComLegendasXY.xml ? langX=en &*

[53] Ver exemplo da Figura 4:3 do Capítulo 4.

[54] Um valor padrão para os atributos *name* (de *resourceParam*) e *value* (de *resourceParam* e de *xsl:templateParam*) podem ser especificados, respectivamente, pelos atributos *defaultName* e *defaultValue*.

langY=pt" (que define o valor *en* para a variável *langX* e *pt* para a variável *langY* do *template*).

Em adição a esses refinamentos a XTemplate, um trabalho futuro é a definição de uma linguagem de domínio especifico para a autoria de *templates*. O uso de XSLT em XTemplate traz grande expressividade à linguagem e permite utilizar processadores XSLT padronizados para o processamento de *templates*. Entretanto, o uso de XSLT adiciona uma grande complexidade na especificação de *templates*, sendo importante estudar funcionalidades que simplifiquem sua autoria. Essas novas funcionalidades podem seguir a abordagem adotada neste trabalho, que estende XSLT, ou pode-se definir uma outra linguagem. Em ambos os casos, é desejável um mapeamento para folhas de estilo XSL (W3C, 1999d).

Um outro trabalho futuro consiste em explorar o conceito de *templates* e adotá-lo tanto no ambiente de autoria quanto no formatador do sistema HyperProp. Uma abordagem para introduzir *templates* no ambiente de autoria gráfica é a definição de uma *interface* Java (por exemplo, "*Template.java*") declarando um método "*process*", que recebe como parâmetro uma composição hipermídia. Classes que implementem essa interface (ou seja, que definam *templates*) devem implementar esse método para processar o *objeto composição* recebido como parâmetro para atribuir a semântica do *template* (incluindo elos na composição, por exemplo). Na interface gráfica, deve-se permitir ao usuário escolher um *template* (uma classe que especifique um *template*) para processar composições sendo editadas. Adicionalmente, a interface "*Template.java*" pode definir os métodos "*checkConstraints*" e "*drawTimeView*", responsáveis, respectivamente, por verificar restrições em uma composição e definir como ela deve ser representada na visão temporal do ambiente de autoria (Coelho, 2004). Esse último método pode ser útil para adicionar o conceito de composições na visão temporal do sistema.

O mecanismo de *plug-ins* do formatador HyperProp (Rodrigues, 2003) pode ser explorado para introduzir o conceito de *templates* no formatador. Dessa forma, cada *template* deve ser implementado como um *plug-in*, sendo responsável pelo controle da exibição de uma composição que referencia esse *template*. Por exemplo, pode-se definir *plug-ins* para composições com semântica temporal paralela e seqüencial. O uso desse conceito pode facilitar a implementação de um formatador modular e distribuído.

Se o processo de incorporação de *templates* nos componentes do sistema HyperProp for bem sucedido, deve-se estabelecer mecanismos para o mapeamento, se possível automático, entre *templates* especificados em NCL, no ambiente de autoria, e no formatador. Paralelamente a esses estudos, a edição gráfica de *templates*, como apresentada na Seção 6.1, deve ser analisada para *templates* NCL.

7.2. Conversões entre Formatos

Em (Joung & Kim, 2002) é definida uma API para a conversão entre XMT-O e XMT-A, além da compilação de XMT-A em BIFS. Existem também ferramentas proprietárias para a compilação de XMT em BIFS ou em arquivos MPEG-4 (.mp4) (IBM, 2004) (GPAC, 2004). Porém, essas implementações não foram desenvolvidas visando a reutilização de código por parte de outros compiladores. Ao contrário, o trabalho descrito em (Costa, 2005) consiste em utilizar o *meta-framework* para definir *frameworks* de compiladores XMT-A e XMT-O; implementar instâncias desses *frameworks* (compiladores) para conversão de especificações XMT-O / XMT-A em BIFS e em NCL; e a construção de compiladores NCL-(XMT-A) e NCL-(XMT-O) - usando o *framework* de compiladores NCL. Como XMT-O é baseada no padrão SMIL, e como existe um mapeamento entre XMT-O e XMT-A, esses compiladores procuram aproveitar os esforços dedicados à construção dos compiladores NCL-SMIL e SMIL-NCL.

Outro trabalho em andamento utiliza a estrutura de dados (voltada para apresentação) definida pelo formatador HyperProp para geração de um documento SMIL com um menor número de composições *par*. Como visto na Seção 6.6, essa abordagem simplifica a exibição de documentos pelos *players* SMIL. No estágio atual, foi implementado (como parte dos trabalhos desenvolvidos nesta dissertação) um compilador NCL-SMIL modificado, que utiliza o compilador NCL-Formatador para determinar objetos com tempo de apresentação disjunto, visando estruturar o documento SMIL. Deve-se estudar outros algoritmos para essa estruturação, testando diferentes abordagens.

A geração de código pelo gerador automático de *framework* de compiladores ainda pode ser melhor explorada. Por exemplo, ao gerar esqueletos de códigos para compiladores específicos, a ferramenta gera novas classes independente se houve alteração do código previamente gerado. Isso dificulta a atualização de compiladores quando existe mudança na linguagem de origem. Um mecanismo de análise de código, para atualizar somente o que foi previamente gerado, deve ser estudado.

Um trabalho futuro interessante é utilizar o *meta-framework* para linguagens de outros domínios, que não o de hipermídia. Por exemplo, uma aplicação para teste foi implementada como parte dos trabalhos desenvolvidos nesta dissertação: um compilador para documentos em XML Schema foi criado utilizando o *gerador automático de frameworks* (usando um arquivo em XML Schema que define a própria linguagem XML Schema). A partir desse compilador, foi implementada uma outra versão para o próprio *gerador automático*. A nova versão do *gerador automático*, gerada a partir do *gerador automático* anterior, foi, então, testada satisfatoriamente para geração de *frameworks* de compiladores. Essa aplicação de teste demonstra como linguagens baseadas em XML, de diferentes domínios, podem aproveitar as facilidades apresentadas neste trabalho.

É importante destacar, também, que o uso do *gerador automático* tem uma outra vantagem: o auxilio na validação tanto de compiladores quanto de *schemas*. Por exemplo, foi possível realizar uma depuração nos arquivos XML Schema de NCL e X-SMIL a partir de *erros* (como ausência de métodos) encontrados nos compiladores gerados automaticamente. Além disso, durante a implementação dos compiladores gerados pelo *gerador automático*, erros de programação puderam ser corrigidos nos compiladores baseados em implementações anteriores.

Ainda um outro trabalho futuro é analisar o uso da abordagem SAX (Seção 6.2) no *framework* de compiladores. Alguns compiladores podem ser implementados sem o uso de árvores DOM, como, aparentemente, é o caso dos compiladores que geram cópias dos arquivos XML de entrada. A utilização da abordagem SAX pode aumentar a eficiência desses compiladores ao eliminar o passo intermediário de construção da árvore DOM. Essa abordagem também pode permitir que um documento seja compilado, sob demanda, enquanto é recebido em um dispositivo de exibição (via *streaming*, por exemplo).

Outra opção para melhorar o desempenho dos compiladores desenvolvidos nesta dissertação é utilizar implementações alternativas para o padrão DOM, como JDOM (JDOM, 2004) e DOM4J (DOM4J, 2004). Essas implementações, apesar de apresentarem algumas diferenças em relação ao padrão, são mais eficientes que as implementações padronizadas, segundo seus desenvolvedores, e permitem uma fácil integração com outras APIs baseadas em DOM. Implementações mais eficientes são possíveis ao se utilizar características particulares de Java, já que o padrão DOM foi definido de forma independente de linguagem de programação (JDOM, 2004).

8
Referências Bibliográficas

AALST, W. M. P. et al. **Workflow Patterns**. Distributed and Parallel Databases, 14(3), p.5-51, Julho de 2003. Disponível em: <http://tmitwww.tm.tue.nl/research/patterns/>. Acesso em: Março de 2004. (Aalst, 2003)

ANTLR. **ANother Tool for Language Recognition 2.7.5**. 28 de Janeiro de 2005. Disponível em <http://www.antlr.org/>. Acesso em Fevereiro de 2005. (ANTLR, 2005)

ANTONACCI, M. J. **NCL: uma Linguagem Declarativa para Especificação de Documentos Hipermídia com Sincronização Temporal e Espacial**. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, Abril de 2000. (Antonacci, 2000)

ANTONACCI, M. J.; MUCHALUAT-SAADE, D. C.; RODRIGUES, R. F.; SOARES, L. F. G. **NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia na Web**. VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia - SBMídia2000, Natal, Rio Grande do Norte, Junho 2000. (Antonacci et al., 2000a)

ANTONACCI, M.J.; MUCHALUAT-SAADE, D.C.; RODRIGUES, R.F.; SOARES, L.F.G. **Improving the expressiveness of XML-based Hypermedia Authoring Languages**. Proceedings of the Multimedia Modeling Conference'2000, Nagano, Japan, Novembro 2000. (Antonacci et al., 2000b)

APACHE - Apache Software Foundation, Apache XML Project. **Apache Xalan-Java XSLT Processor**. 1 de Novembro de 2002. Disponível em <http://xml.apache.org/xalan-j/>. Acesso em Fevereiro de 2005. (Apache XML, 2002a)

APACHE - Apache Software Foundation, Apache XML Project. **Apache Xerces2 XML Parsers**. 6 de Maio de 2003. Disponível em <http://xml.apache.org/xerces2-j/>. Acesso em Fevereiro de 2005. (Apache XML, 2002b)

BACHELET B. et al. **Elastic Time Computation for Hyper-media Documents**. VI Brazilian Symposium on Multimedia and Hypermedia Systems - SBMídia'2000, Natal, Brazil, (2000) 47-62. (Bachelet et al., 2000)

BADROS, G. J. **JavaML: A Markup Language for Java Source Code**. The International Journal of Computer and Telecommunications Networking, Volume 33, Issue 1-6, Junho de 2000, Amsterdam, Holanda, p.159-177, North-Holland Publishing Co., 2000. (Badros, 2000)

BERNERS-LEE, T.J. et al. The World-Wide Web. Communications of the ACM, v. 37, n. 8, Agosto de 1994, p. 76-82. (Berners-Lee, 1994)

BOLL, S.; KLAS, W. **Z_YX - A Semantic Model for Multimedia Documents and Presentations**. VIII IFIP Conference on Data Semantics, 1999. (Boll & Klas, 1999)

BRUSILOVSKY, P. **Methods and Techniques of Adaptive Hypermedia**. Journal of User Modelling and User-Adaptive Interaction, v. 6, n. 2-3, 1996, p. 87-129. (Brusilovsky, 1996)

CASANOVA, M.A. et al. **The Nested Context Model for Hyperdocuments**. Hypertext'91, San Antonio, EUA, Dezembro de 1991, p. 193-201. (Casanova et al., 1991)

CELENTANO A.; GAGGI O. **Template-Based Generation of Multimedia Presentations**. International Journal of Software Engineering and Knowledge Engineering, Vol. 13, No. 4, 2003, p.419-445. World Scientific Publishing Company. (Celentano & Gaggi, 2003)

COELHO, R.M. **Integração de Ferramentas Gráficas e Declarativas na Autoria de Arquiteturas Modeladas através de Grafos Compostos**. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, Agosto de 2004. (Coelho, 2004)

COSTA, R.M.R. **Integração e Interoperabilidade de Documentos MPEG-4 e NCL**. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, previsão de defesa: Agosto de 2004. (Costa, 2005)

DEY, A.K.; SALBER, D.; ABOWD, G.D. **A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications**. Human-Computer Interaction (HCI) Journal - special issue on Context-Aware Computing, v. 16, n. 2-4, 2001, p. 97-166. (Dey et al., 2001)

DOM4J. **DOM4J 1.5**. 15 de Novembro de 2004. Disponível em <http://dom4j.org/>. Acesso em Fevereiro de 2005. (DOM4J, 2004)

FLEXTV. TV Digital: **Análise das Alternativas Tecnológicas**. Relatório em atendimento ao Requisito 4.1.3 - RFP04/2004 - MCT/FINEP, produto 1A, dezembro, 2004. (FLEXTV, 2004a)

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley, 1995. (Gamma et al., 1995)

GOOSE S. et al. **Streaming Speech^3: A Framework for Generating and Streaming 3D Text-To-Speech and Audio Presentations to Wireless PDAs as Specified Using Extensions to SMIL**. International World Wide Web Conference - WWW'02, Maio de 2002, Honolulu, Hawaii, E.U.A., p.37-44, ISBN: 1-58113-449-5. (Goose et al., 2002)

GPAC. **Project on Advanced Content**. Disponível em <http://gpac.sourceforge.net/>. (GPAC, 2004)

IBM. **IBM Toolkit for MPEG-4**. Disponível em <http://www.alphaworks.ibm.com/tech/tk4mpeg4>. (IBM, 2004)

IBM. **Jikes Java Compiler**. Dezembro de 1998. Disponível em <http://www.alphaworks.ibm.com/tech/Jikes>. Acesso em Fevereiro de 2005. (IBM, 1998)

ISO - International Organization for Standardization. Information processing - Text and office systems - **Standard Generalized Markup Language (SGML)** 8879, 1986. (ISO, 1986)

ISO - International Organization for Standardization. ISO/IEC – 14496-1:2001. **Coding of Audio-Visual Objects – Part 1: Systems**. Second Edition, 2001. (ISO, 2001)

JavaCC. **Java Compiler Compiler (JavaCC) version 3.2**. 12 de Agosto de 2003. Disponível em <https://javacc.dev.java.net/ >. Acesso em Fevereiro de 2005. (JavaCC, 2003)

JDOM. **JDOM 1.0**. 09 de Setembro de 2004. Disponível em <http://www.jdom.org>. Acesso em Fevereiro de 2005. (JDOM, 2004)

JOUNG Y., KIM K. **An XMT API for generation of the MPEG-4 scene description**. Proceedings of the tenth ACM international conference on Multimedia. Juan-les-Pins, França, 2002, pp. 307-310. (Joung & Kim, 2002)

KING P.; SCHMITZ P.; THOMPSON S.. **Behavioral Reactivity and Real Time Programming in XML Functional Programming meets SMIL Animation**. ACM Symposium on Document Engineering - DocEng'04, Outubro de 2004, Milwaukee, E.U.A., p.188-197, ISBN: 1-58113-938-1. (King et al., 2004)

KOENEN, R. **MPEG-4 Overview - (V21 - Jeju Version)**, Março 2002. Disponível em <http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm>. Acesso em Março de 2005. (Koenen, 2002)

LEVINE, J. R. **Lex & Yacc**. O'Reilly & Associates, Inc., Sebastopol, California, 2nd edition, 1992. (Levine, 1995)

LI, Q. et al. **XVM: A Bridge between XML Data and Its Behavior**. International World Wide Web Conference 2004, New York, E.U.A., p.155-163, ISBN: 1-58113-844-X. ACM Press, 2004. (Li et al., 2003)

LUCENA, P.S. **Expressive Talking Heads: uma Ferramenta com Fala e Expressão Facial Sincronizadas para o Desenvolvimento de Aplicações Interativas**. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, Setembro de 2002. (Lucena, 2002)

MUCHALUAT-SAADE, D. C.; RODRIGUES, R. F.; SOARES L. F. G. **XConnector: Extending XLink to Provide Multimedia Synchronization**. ACM Symposium on Document Engineering - DocEng'02, Virginia, USA, Novembro de 2002. (Muchaluat-Saade et al., 2002)

MUCHALUAT-SAADE, D.C. **Relações em Linguagens de Autoria Hipermídia: Aumentando Reuso e Expressividade**. Tese de Doutorado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, Março de 2003. (Muchaluat-Saade, 2003)

MUCHALUAT-SAADE, D.C.; SILVA, H.V.O.; SOARES, L.F.G. **Linguagem NCL versão 2.0 para Autoria Declarativa de Documentos Hipermídia**. WEBMídia 2003, Salvador, Novembro de 2003. (Muchaluat-Saade et al., 2003)

RODRIGUES, R.F. **Formatação e Controle de Apresentações Hipermídia com Mecanismos de Adaptabilidade**, Tese de Doutorado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Março 2003. (Rodrigues, 2003)

RODRIGUES, R.F.; SOARES, L. F. G. **Inter and Intra Media-Object QoS Provisioning in Adaptive Formatters**. ACM Symposium on Document Engineering - DocEng'03, Grenoble, França, Novembro 2003. (Rodrigues et al., 2003)

RODRIGUES, R.F. et al. **Cross-Media and Elastic Time Adaptive Presentations: the Integration of a Talking Head Tool into a Hypermedia Formatter**. Adaptive Hypermedia and Adaptive Web-Based Systems, 3., Lecture Notes in Computer Science (LNCS 3137), Agosto de 2004, Eindhoven, Holanda, p.215-224, ISBN 3-540-22895-0. (Rodrigues et al., 2004)

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language: Reference Manual**. Addison-Wesley, 1999. (Rumbaugh et al., 1999)

SAX. **Simple API for XML (SAX)**. 29 de Janeiro de 2002. Disponível em <http://sax.sourceforge.net/>. Acesso em Fevereiro de 2005. (SAX, 2002)

SCHILIT, B.; ADAMS, N.; WANT, R. **Context-aware computing applications**. IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, EUA, Dezembro de 1994, p. 85-90. (Schilit et al., 1994)

SILVA, H.V.O. et al. **NCL 2.0: Integrating New Concepts to XML Modular Languages**. ACM Symposium on Document Engineering - DocEng'04, Outubro de 2004, Milwaukee, E.U.A., p.188-197, ISBN: 1-58113-938-1. (Silva et al., 2004a)

SILVA, H.V.O. **Workflow & Multimídia: Um Estudo Comparativo**. Relatório Técnico, Laboratório TeleMídia, PUC-Rio, Dezembro de 2003. Disponível em <http://www.telemidia.puc-rio.br>. Acesso em Janeiro de 2005. (Silva, 2003)

SILVA, H.V.O.; RODRIGUES, R.F.; SOARES, L.F.G. **SMIL+XTemplate**. X Simpósio Brasileiro de Sistemas Multimídia e Web, Ribeirão Preto, São Paulo. Anais do WebMedia & LA-Web, 2004. p. 79-86. (Silva et al., 2004b)

SOARES, L. F. G., CASANOVA, M. A.; RODRIGUEZ, N. R. **Nested Composite Nodes and Version Control in an Open Hypermedia System**. International Journal on Information Systems; Special issue on Multimedia Information Systems, 20(6):501-520, Elsevier Science Ltd. England, Setembro 1995. (Soares et al., 1995)

SOARES, L. F. G.; RODRIGUES, R. F.; MUCHALUAT-SAADE, D. C. **Modeling, Authoring and Formatting Hypermedia Documents in the HyperProp System**. ACM Multimedia Systems Journal, v. 8, n. 2, Springer-Verlag, Março de 2000, p. 118-134. (Soares et al., 2000)

SOARES, L.F.G.; RODRIGUES, R.F.; MUCHALUAT-SAADE, D.C. **Modelo de Contextos Aninhados - versão 3.0**. Relatório Técnico, Laboratório TeleMídia, PUC-Rio, Março 2003. Disponível em <http://www.telemidia.puc-rio.br>. Acesso em Janeiro de 2005. (Soares et al., 2003)

Sun Microsystems. **Java API for XML Processing (JAXP)**. 23 de Agosto de 2002. Disponível em <http://java.sun.com/xml/jaxp/>. Acesso em Fevereiro de 2005. (Sun, 2002)

THATTE, S. et al. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1. 5 de Maio de 2003. Disponível em:

<http://dev2dev.bea.com/techtrack/BPEL4WS.jsp>. Acesso em Março de 2004. (Thatte, 2003)

VAN ROSSUM, G.; JABSEB, J.; MULLENDER, K. S.; BULTERMAN, D. **CMIFed: A Presentation Environment for Portable Hypermedia Documents**. Proceedings of ACM Multimedia'93, California, 1993. pp. 183-188. (van-Rossum et al., 1993)

VILLARD, L.; ROISIN, C.; LAYAÏDA, N. VILLARD, L.; ROISIN, C.; LAYAÏDA, N. **An XML-based multimedia document processing model for content adaptation**. VIII International Conference on Digital Documents and Electronic Publishing, Setembro de 2000. VIII International Conference on Digital Documents and Electronic Publishing, Setembro de 2000. (Villard et al., 2000)

W3C - World-Wide Web Consortium. **Cascading Style Sheets, level 2 (CSS2 Specification)**. W3C Recommendation, 12 de maio de 1998. Disponível em < http://www.w3.org/TR/REC-CSS2>. Acesso em Janeiro de 2005. (W3C, 1998a)

W3C - World-Wide Web Consortium. **Document Object Model (DOM) Level 2 Core Specification Version 1.0**. W3C Recommendation, 13 de Novembro de 2000. Disponível em <http://www.w3.org/TR/DOM-Level-2-Core>. Acesso em Janeiro de 2005. (W3C, 2000a)

W3C - World-Wide Web Consortium. **Extensible Markup Language (XML) 1.0 (Second Edition).** W3C Recommendation, Outubro de 2000. Disponível em <http://www.w3.org/TR/REC-xml>. Acesso em: Janeiro de 2004. (W3C, 2000b)

W3C - World-Wide Web Consortium. **Extensible Markup Language (XML) 1.1**. W3C Recommendation, fevereiro de 2004. Disponível em < http://www.w3.org/TR/xml11>. Acesso em Janeiro de 2005. (W3C, 2004)

W3C - World-Wide Web Consortium. **Extensible Stylesheet Language (XSL) Version 1.0**. W3C Recommendation, 15 de outubro de 2001. Disponível em <http://www.w3.org/TR/xsl>. Acesso em janeiro de 2005. (W3C, 2001a)

W3C - World-Wide Web Consortium. **HyperText Markup Language (HTML)**. W3C Recommendation, Dezembro de 1999. Disponível em <http://www.w3.org/TR/html4/>. Acesso em: Janeiro de 2004. (W3C, 1999a)

W3C - World-Wide Web Consortium. **Mathematical Markup Language (MathML) Version 2.0 (Second Edition)**. W3C Recommendation, 21 de outubro de 2003. Disponível em < http://www.w3.org/TR/MathML2>. Acesso em Janeiro de 2005. (W3C, 2003)

W3C - World-Wide Web Consortium. **Namespaces in XML**. W3C Recommendation, 14 de janeiro de 1999. Disponível em < http://www.w3.org/TR/REC-xml-names>. Acesso em Janeiro de 2005. (W3C, 1999b)

W3C - World-Wide Web Consortium. **Synchronized Multimedia Integration Language (SMIL 2.0)**. W3C Recommendation, Agosto de 2001. Disponível em <http://www.w3.org/TR/smil20>. Acesso em: Janeiro de 2005. (W3C, 2001b)

W3C - World-Wide Web Consortium. **Synchronized Multimedia Integration Language (SMIL) 1.0 Specification**. W3C Recommendation, 15 junho de 1998.

Disponível em <http://www.w3.org/TR/REC-smil/>. Acesso em: Janeiro de 2005. (W3C, 1998b)

W3C - World-Wide Web Consortium. **Timed Interactive Multimedia Extensions for HTML (HTML+TIME)**. W3C Submission Request, 18 de setembro de 1998. Disponível em <http://www.w3.org/TR/NOTE-HTMLplusTIME>. Acesso em Janeiro de 2005. (W3C, 1998c)

W3C - World-Wide Web Consortium. **URIs, URLs, and URNs: Clarifications and Recommendations 1.0**. Report from the joint W3C/IETF URI Planning Interest Group, 21 de setembro de 2001. Disponível em < http://www.w3.org/TR/uri-clarification/>. Acesso em Março de 2005. (W3C, 2001c)

W3C - World-Wide Web Consortium. **XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)**. W3C Recommendation, 26 de Janeiro de 2000. Disponível em <http://www.w3.org/TR/xhtml1/>. Acesso em Janeiro de 2005. (W3C, 2000c)

W3C - World-Wide Web Consortium. **XML Path Language (XPath)**. W3C Recommendation, 16 de novembro de 1999. Disponível em <http://www.w3.org/TR/xpath>. Acesso em Janeiro de 2005. (W3C, 1999c)

W3C - World-Wide Web Consortium. **XML Schema Part 0: Primer**. W3C Recommendation, 2 de maio de 2001. Disponível em < http://www.w3.org/TR/xmlschema-0>. Acesso em Janeiro de 2005. (W3C, 2001d)

W3C - World-Wide Web Consortium. **XSL Transformations (XSLT) Version 1.0**. W3C Recommendation, 16 de novembro de 1999. Disponível em < http://www.w3.org/TR/xslt>. Acesso em Janeiro de 2005. (W3C, 1999d)

WIDEMANN, B. T.; LEPPER M.; WIELAND J. **Automatic Construction of XML-Based Tools Seen as Meta-Programming**. Automated Software Engineering, Vol. 10, Issue 1, p.23-38, Janeiro de 2003, ISSN:0928-8910. Kluwer Academic Publishers. (Widemann et al., 2003)

YANG C.; YANG Y. **SMILAuthor: An Authoring System for SMIL-Based Multimedia Presentations**. Multimedia Tools and Applications, Dezembro de 2003, Hingham, MA, E.U.A., Volume 21, Issue 3, p. 243-260, Kluwer Academic Publishers. (Yang & Yang, 2003)

9
Apêndice A

Este apêndice descreve os elementos e atributos das áreas funcionais de NCL 2.0 e NCL 2.1. Os símbolos nas tabelas possuem os seguintes significados: "?" opcional, "|" ou, "*" zero ou mais ocorrências e "+" uma ou mais ocorrências.

9.1.
NCL 2.0

Área Funcional	Elementos	Atributos	Conteúdo
Structure	ncl	id	(head?, body)
	head		(Metainformation*, layout?, descriptorBase?)
	body		(BasicMedia|composite|switch||linkBase)*
Components	animation, audio, img, text, textstream, vídeo, ref	id, src, type, descriptor, label	area*, attribute*
	composite	id, descriptor, xtemplate, label	(areaComposite*, port*, attribute*, componentPresentation*, (BasicMedia|composite||linkBase| switch)*)
Interfaces	Area	id, coords, begin, end, dur, text, position, first, last, label	Vazio
	areaComposite	id, label, componentList	Vazio
	port	id, component, port, label	Vazio
	attribute	id, name	Vazio
	portSwitch	id, label	port+
Linking	bind	role, component,	Vazio

		port	
	param	name, value	Vazio
	link	id, xconnector	(param*, bind+)
	lref	id, src	Vazio
	linkBase	id	(link|lref)+
Connectors	xconnector	id	(param*, role+, glue)
	param	name, type	Vazio
	connectorBase	id	xconnector+
	compositeConnector	id	(role+, glue)
Composite Templates	xtemplate	id	(vocabulary, constraints?)
	templateBase	id	xtemplate+
Layout	layout	src, type	topLayout*
	topLayout	id, title, left, top, height, width, backgroundColor, zIndex, scroll, open, close, movable, resizable, visible	region*
	region	id, title, left, top, height, width, backgroundColor, zIndex, scroll, open, close, fit, visible	region*
Presentation Specification	descriptor	id, player, dur, min, max, region, enableTimeBar, style, soundLevel, balanceLevel, trebleLevel, bassLevel,	Vazio
	descriptorBase		(descriptor|descriptorSwitch)*
	componentPresentation	component, descriptor	Vazio
Presentation Control	switch	id,	(portSwitch*, componentPresentation*, (BasicMedia|composite|switch)+)
	descriptorSwitch	id	descriptor+

9.2. NCL 2.1

Área Funcional	Elementos	Atributos	Conteúdo
Structure	Ncl	id	(head?, body)
	Head		(Metainformation*, layout?, descriptorBase?)
	Body		(BasicMedia|composite|switch||linkBase)*
Components	Animation, audio, img, text, textstream, vídeo	id, src, ref, type, descriptor, label, implicitDur	area*, attribute*
	Composite	id, ref, descriptor, xtemplate, label	(areaComposite*, port*, attribute*, bindDescriptor*, (BasicMedia|composite|linkBase| switch)*)
Interfaces	Area	id, coords, begin, end, dur, text, position, first, last, label	Vazio
	areaComposite	id, label, componentList	Vazio
	Port	id, component, port, label	Vazio
	Attribute	id, name	Vazio
	portSwitch	id, label	port+
Linking	Bind	role, component, port	Vazio
	Param	name, value	Vazio
	Link	id, xconnector	(param*, bind+)
	Lref	id, src	Vazio
	linkBase	id	(link||lref)+
Connectors	xconnector	id	(param*, role+, glue)
	Param	name, type	Vazio
	connectorBase	id	xconnector+
	compositeConnector	id	(role+, glue)
Composite Templates	Xtemplate	id	(vocabulary, constraints?)
	templateBase	id	xtemplate+
Layout	Layout	src, type, ref	topLayout*

	topLayout	id, title, left, top, height, width, backgroundColor, zIndex, scroll, open, close, movable, resizable, visible	region*
	Region	id, title, left, top, height, width, backgroundColor, zIndex, scroll, open, close, fit, visible	region*
Presentation Specification	Descriptor	id, player, explicitDur, min, max, ref, region, enableTimeBar, style, soundLevel, balanceLevel, trebleLevel, bassLevel, nodeRule	descriptorParam*
	descriptorParam	name, value	
	descriptorBase	ref	(descriptor|descriptorSwitch)*
	bindDescriptor	component, descriptor	Vazio
Presentation Control	Switch	id, ref	(portSwitch*, bindDescriptor*, bindRule*,(BasicMedia|composite|switch)+)
	descriptorSwitch	id	descriptor+, bindRule*
	presentationRuleBase	ref	(presentationRule | compositePresentationRule)+
	presentationRule	ref	Vazio
	compositePresentationRule	ref	(presentationRule | compositePresentationRule)+
	bindRule		Vazio
Timing	costFunctionBase	id, ref	costFunction+
	costFunction	id, type, ref, mathMLRef	costFunctionParam*
	costFunctionParam	name, value	Vazio

9.3. XConnector 2.1

Elementos	Atributos	Conteúdo
connectorBase	id, name, description	(causalConnector, constraintConnector)*
causalConnector	id, name, description	param, (conditionRole, propertyRole)*, actionRole*, causalGlue
constraintConnector	id, name, description	param, propertyRole*, constraintGlue
param	name, type	
actionRole	id, eventType, attributeName, min, max, actionType, show, delay, repeat, delayBetweenRep, initialValue, finalValue, duration	
conditionRole	id, eventType, attributeName, min, max	(eventStateCondition, eventTransitionCondition, eventAttributeCondition, compoundCondition)
eventStateCondition	isNegated, state, comparator	
eventTransitionCondition	isNegated, transition	
eventAttributeCondition	isNegated, comparator, attributeType, value, attributeName	
compoundCondition	isNegated, operator	(eventStateCondition, eventTransitionCondition, eventAttributeCondition, compoundCondition)*
propertyRole	id, eventType, attributeName, min, max	(eventStateProperty, eventTransitionProperty, eventAttributeProperty)
eventStateProperty	state	
eventTransitionProperty	transition, offset	
eventAttributeProperty	attributeType, offset	
causalGlue		(simpleTriggerExpression | compoundTriggerExpression), (simpleActionExpression | compoundActionExpression)
constraintGlue		(propertyToPropertyExpression | attributeToValueExpression |

		eventStateToValueExpression | compoundPropertyExpression)
simpleTriggerExpression	isNegated, minDelay, maxDelay, conditionRole, qualifier	
compoundTriggerExpression	isNegated, minDelay, maxDelay, operator	(simpleTriggerExpression| compoundTriggerExpression)* | ((simpleTriggerExpression| compoundTriggerExpression), (propertyToPropertyExpression | attributeToValueExpression | eventStateToValueExpression | compoundPropertyExpression))*
propertyToPropertyExpression	comparator, firstPropertyRole, firstQualifier, secondPropertyRole, secondQualifier	
attributeToValueExpression	comparator, propertyRole, qualifier, value	
eventStateToValueExpression	comparator, propertyRole, qualifier, state	
compoundPropertyExpression	operator	(propertyToPropertyExpression | attributeToValueExpression | eventStateToValueExpression | compoundPropertyExpression)*
simpleActionExpression	delay, actionRole, qualifier	
compoundActionExpression	delay, operator	(simpleActionExpression | compoundActionExpression)*

10
Apêndice B

Especificação da linguagem NCL 2.1.

10.1.
NCL21.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- include the schema files for the building block types -->
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
struct.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
interface.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
link.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
component.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
timing.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
layout.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
presentation.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
control.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
connector.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
compositeTemplate.xsd"/>

    <!-- import the NCL 2.0 language namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- import the definitions in the modules namespaces -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Structure"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-Structure.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/BasicMedia"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-BasicMedia.xsd"/>
```

```
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicComposite"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-BasicComposite.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/MediaInterface"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-MediaInterface.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeInterface"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
CompositeInterface.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/AttributeInterface"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
AttributeInterface.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/SwitchInterface"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-SwitchInterface.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Linking"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-Linking.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XConnector"
schemaLocation="file:../mediaContent/data/schemas/XConnector.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeConnector"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
CompositeConnector.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate"
schemaLocation="file:../mediaContent/data/schemas/XTemplate.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/TemplateUse"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-XTemplateUse.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/BasicTiming"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-BasicTiming.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/BasicLayout"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-BasicLayout.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicDescriptor"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-BasicDescriptor.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeDescriptor"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
CompositeDescriptor.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/TestAttributes"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-TestAttributes.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/ContentControl"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-ContentControl.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/DescriptorControl"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-
DescriptorControl.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/CostFunction"
```

```
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-CostFunction.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/TestRules"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-TestRules.xsd"/>
</schema>
```

10.2. NCL-AttributeInterface.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:AttributeInterface="http://www.telemidia.puc-
rio.br/specs/xml/ncl/AttributeInterface"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/AttributeInterface"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="attribute" type="ncllang:attributeInterfaceType"
substitutionGroup="ncllang:attribute"/>

    <!-- declare global attributes in this module -->
</schema>
```

10.3. NCL-BasicComposite.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:BasicComposite="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicComposite"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicComposite"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
```

```
    <element name="composition" type="ncllang:compositionType"
substitutionGroup="ncllang:composition"/>
</schema>
```

10.4. NCL-BasicDescriptor.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:BasicDescriptor="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicDescriptor"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicDescriptor"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global attributes in this module -->
    <attribute name="descriptor" type="IDREF"/>

    <!-- declare global elements in this module -->
    <element name="descriptor" type="ncllang:descriptorType"
substitutionGroup="ncllang:descriptor"/>
    <element name="descriptorBase" type="ncllang:descriptorBaseType"
substitutionGroup="ncllang:descriptorBase"/>
    <element name="descriptorParam" type="ncllang:descriptorParamType"
substitutionGroup="ncllang:descriptorParam"/>
</schema>
```

10.5. NCL-BasicLayout.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:BasicLayout="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicLayout"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicLayout"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
```

```
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="layout" type="ncllang:layoutType"
substitutionGroup="ncllang:layout"/>
    <element name="topLayout" type="ncllang:topLayoutType"
substitutionGroup="ncllang:topLayout"/>
    <element name="region" type="ncllang:regionType"
substitutionGroup="ncllang:region"/>

    <!-- declare global attributes in this module -->
    <attribute name="region" type="IDREF"/>
</schema>
```

10.6. NCL-BasicMedia.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:BasicMedia="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicMedia"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/BasicMedia"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="text" type="ncllang:mediaType"
substitutionGroup="ncllang:text"/>
    <element name="img" type="ncllang:mediaType" substitutionGroup="ncllang:img"/>
    <element name="audio" type="ncllang:mediaType"
substitutionGroup="ncllang:audio"/>
    <element name="animation" type="ncllang:mediaType"
substitutionGroup="ncllang:animation"/>
    <element name="video" type="ncllang:mediaType"
substitutionGroup="ncllang:video"/>
    <element name="textstream" type="ncllang:mediaType"
substitutionGroup="ncllang:textstream"/>

    <!-- declare global attributes in this module -->
    <attribute name="type" type="string"/>
    <attribute name="src" type="anyURI"/>
</schema>
```

10.7. NCL-BasicTiming.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:BasicTiming="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicTiming"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicTiming"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>

    <!-- declare global attributes in this module -->
    <attribute name="dur" type="string"/>
    <attribute name="min" type="string"/>
    <attribute name="max" type="string"/>
    <attribute name="implicitDur" type="string"/>
</schema>
```

10.8. NCL-component.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- define the media src attributes-->
    <attribute name="type" type="string"/>
    <attribute name="src" type="anyURI"/>
    <!-- define the media src attribute group -->
    <attributeGroup name="mediaSrcAttrs">
        <attribute name="type" type="string" use="optional"/>
        <attribute name="src" type="anyURI" use="optional"/>
        <attribute name="ref" type="string" use="optional"/>
    </attributeGroup>

    <complexType name="mediaPrototype">
        <attributeGroup ref="ncl:mediaSrcAttrs" />
    </complexType>

    <!--    define the composition element prototype    -->
```

```
    <complexType name="compositionPrototype">
    </complexType>

    <!-- define the global media elements -->
    <element name="text" type="ncllang:mediaType"
substitutionGroup="ncllang:text"/>
    <element name="img" type="ncllang:mediaType" substitutionGroup="ncllang:img"/>
    <element name="audio" type="ncllang:mediaType"
substitutionGroup="ncllang:audio"/>
    <element name="animation" type="ncllang:mediaType"
substitutionGroup="ncllang:animation"/>
    <element name="video" type="ncllang:mediaType"
substitutionGroup="ncllang:video"/>
    <element name="textstream" type="ncllang:mediaType"
substitutionGroup="ncllang:textstream"/>
    <element name="ref" type="ncllang:mediaType" substitutionGroup="ncllang:ref"/>

    <!-- define the global composition element -->
    <element name="composition" type="ncllang:compositionType"
substitutionGroup="ncllang:composition"/>
</schema>
```

10.9. NCL-CompositeConnector.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:CompositeConnector="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeConnector"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeConnector"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="compositeConnector" type="ncllang:compositeConnectorType"
substitutionGroup="ncllang:compositeConnector"/>
    <!--
    <element name="partialLinkBase" type="ncllang:partialLinkBaseType"
substitutionGroup="ncllang:partialLinkBase"/>
    -->
    <element name="partialLinkBase" substitutionGroup="ncllang:partialLinkBase"/>
</schema>
```

10.10. NCL-CompositeDescriptor.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:CompositeDescriptor="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeDescriptor"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeDescriptor"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="componentPresentation" type="ncllang:componentPresentationType"
substitutionGroup="ncllang:componentPresentation"/>
    <element name="bindDescriptor" type="ncllang:bindDescriptorType"
substitutionGroup="ncllang:bindDescriptor"/>
</schema>
```

10.11. NCL-CompositeInterface.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:CompositeInterface="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeInterface"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CompositeInterface"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="compositeArea" type="ncllang:compositeAnchorType"
substitutionGroup="ncllang:compositeArea"/>
    <element name="port" type="ncllang:compositePortType"
substitutionGroup="ncllang:port"/>
</schema>
```

10.12. NCL-compositeTemplate.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- import the definitions in the XTemplate namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate"
schemaLocation="file:../mediaContent/data/schemas/XTemplate.xsd"/>

    <!-- declare global attributes in this module -->
    <attribute name="xtemplate" type="anyURI"/>
    <attribute name="label" type="string"/>

    <!-- define the CompositeTemplateUseAttrs attribute group -->
    <attributeGroup name="CompositeTemplateUseAttrs">
        <attribute name="xtemplate" type="anyURI"/>
        <attribute name="label" type="string"/>
    </attributeGroup>

    <!-- define the TemplateComponentAttrs attribute group -->
    <attributeGroup name="TemplateComponentAttrs">
        <attribute name="label" type="string"/>
    </attributeGroup>
</schema>
```

10.13. NCL-connector.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- import the definitions in the XConnector namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XConnector"
schemaLocation="file:../mediaContent/data/schemas/XConnector.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!--    define the composite connector element prototype    -->

    <complexType name="compositeConnectorPrototype">
    </complexType>

    <complexType name="compositeRolePrototype">
```

```
        <attribute name="id" type="string" use="required"/>
        <attribute name="partialLink" type="IDREF" use="required"/>
        <attribute name="role" type="string" use="required"/>
    </complexType>

    <complexType name="compositeGluePrototype">
    </complexType>

    <complexType name="partialLinkBasePrototype">
    </complexType>

    <!-- define the global composite element -->
    <element name="compositeConnector" type="ncllang:compositeConnectorType"
substitutionGroup="ncllang:compositeConnector"/>

    <element name="partialLinkBase" type="ncllang:partialLinkBaseType" />
    <element name="role" type="ncllang:compositeRoleType"
substitutionGroup="ncllang:role"/>
    <element name="glue" type="ncllang:compositeGlueType"
substitutionGroup="ncllang:glue"/>
</schema>
```

10.14. NCL-ContentControl.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ContentControl="http://www.telemidia.puc-
rio.br/specs/xml/ncl/ContentControl"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/ContentControl"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- import the definitions in the NCL namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="switch" type="ncllang:switchType"
substitutionGroup="ncllang:switch"/>
    <element name="bindRule" type="ncllang:bindRuleType"
substitutionGroup="ncllang:bindRule"/>
</schema>
```

10.15. NCL-control.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- define the switch element prototype -->
    <complexType name="switchPrototype">
    </complexType>
    <complexType name="switchPrototype2">
    </complexType>

    <!-- define the descriptor switch element prototype -->
    <complexType name="descriptorSwitchPrototype">
    </complexType>

    <!-- define the bindRule element prototype -->
    <complexType name="bindRulePrototype">
              <attribute name="rule" type="IDREF" use="required"/>
              <attribute name="component" type="IDREF" use="required"/>
    </complexType>

    <complexType name="presentationRuleBasePrototype">
    </complexType>

    <complexType name="presentationRulePrototype">
        <attribute name="var" type="string" use="required"/>
        <attribute name="value" type="string" use="required"/>
        <attribute name="op" use="required">
          <simpleType>
             <restriction base="string">
                <enumeration value="eq"/>
                <enumeration value="ne"/>
                <enumeration value="gt"/>
                <enumeration value="ge"/>
                <enumeration value="lt"/>
                <enumeration value="le"/>
            </restriction>
          </simpleType>
        </attribute>
    </complexType>

    <complexType name="compositePresentationRulePrototype">
        <attribute name="op" use="required">
          <simpleType>
```

```
            <restriction base="string">
               <enumeration value="and"/>
               <enumeration value="or"/>
            </restriction>
         </simpleType>
        </attribute>
    </complexType>

    <!-- define the global content control elements -->
    <element name="switch" type="ncllang:switchType"
substitutionGroup="ncllang:switch"/>
    <element name="descriptorSwitch" type="ncllang:descriptorSwitchType"
substitutionGroup="ncllang:descriptorSwitch"/>
    <element name="bindRule" type="ncllang:bindRuleType"
substitutionGroup="ncllang:bindRule"/>

    <element name="presentationRule" type="ncllang:presentationRuleType"
substitutionGroup="ncllang:presentationRule"/>
    <element name="compositePresentationRule"
type="ncllang:compositePresentationRuleType"
substitutionGroup="ncllang:compositePresentationRule"/>
    <element name="presentationRuleBase" type="ncllang:presentationRuleBaseType"
substitutionGroup="ncllang:presentationRuleBase"/>
</schema>
```

10.16. NCL-CostFunction.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:BasicDescriptor="http://www.telemidia.puc-
rio.br/specs/xml/ncl/BasicDescriptor"
        xmlns:CostFunction="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CostFunction"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/CostFunction"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="costFunction" type="ncllang:costFunctionType"
substitutionGroup="ncllang:costFunction"/>
    <element name="costFunctionBase" type="ncllang:costFunctionBaseType"
substitutionGroup="ncllang:costFunctionBase"/>
```

```
    <element name="costFunctionParam" type="ncllang:costFunctionParamType"
substitutionGroup="ncllang:costFunctionParam"/>
</schema>
```

10.17. NCL-DescriptorControl.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:DescriptorControl="http://www.telemidia.puc-
rio.br/specs/xml/ncl/DescriptorControl"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/DescriptorControl"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- import the definitions in the NCL namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="descriptorSwitch" type="ncllang:descriptorSwitchType"
substitutionGroup="ncllang:descriptorSwitch"/>
</schema>
```

10.18. NCL-interface.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- define the coords attribute -->
    <attribute name="coords" type="string"/>

    <!-- define the begin, end, dur attributes -->
    <attribute name="begin" type="string"/>
    <attribute name="end" type="string"/>
    <!-- <attribute name="dur" type="unsignedLong"/> -->

    <attribute name="anchorLabel" type="string"/>
```

```
<!-- define the text, position attributes -->
<attribute name="text" type="string"/>
<attribute name="position" type="unsignedLong"/>

<!-- define the first, last attributes -->
<attribute name="first" type="unsignedLong"/>
<attribute name="last" type="unsignedLong"/>

<!-- define the component, anchor attributes -->
<attribute name="component" type="string"/>
<attribute name="anchor" type="string"/>

<!-- define the component list attribute -->
<attribute name="componentList" type="IDREFS"/>

<!-- define the attribute name attribute -->
<!-- <attribute name="name" type="string"/> -->

<!-- define the TemporalAnchorAttrs attribute group -->
<attributeGroup name="TemporalAnchorAttrs">
    <attribute name="begin" type="string"/>
    <attribute name="end" type="string"/>
    <attribute name="dur" type="string"/>
    <attribute name="implicitDur" type="string"/>
</attributeGroup>

<!-- define the TextAnchorAttrs attribute group -->
<attributeGroup name="TextAnchorAttrs">
    <attribute name="text" type="string"/>
    <attribute name="position" type="unsignedLong"/>
</attributeGroup>

<!-- define the SampleAnchorAttrs attribute group -->
<attributeGroup name="SampleAnchorAttrs">
    <attribute name="first" type="unsignedLong"/>
    <attribute name="last" type="unsignedLong"/>
</attributeGroup>

<!-- define the CompositePortAttrs attribute group -->
<attributeGroup name="CompositePortAttrs">
    <attribute name="component" type="IDREF"/>
    <attribute name="port" type="string"/>
</attributeGroup>

<complexType name="anchorPrototype">
</complexType>

<complexType name="imgAnchorPrototype">
    <complexContent>
        <extension base="ncl:anchorPrototype">
            <attribute name="anchorLabel" type="string"/>
```

```
            <attribute name="coords" type="string"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="audioAnchorPrototype">
    <complexContent>
        <extension base="ncl:anchorPrototype">
            <attribute name="anchorLabel" type="string"/>
            <attributeGroup ref="ncl:TemporalAnchorAttrs" />
            <attributeGroup ref="ncl:SampleAnchorAttrs" />
        </extension>
    </complexContent>
</complexType>

<complexType name="videoAnchorPrototype">
    <complexContent>
        <extension base="ncl:anchorPrototype">
            <attribute name="anchorLabel" type="string"/>
            <attributeGroup ref="ncl:TemporalAnchorAttrs" />
            <attribute name="coords" type="string"/>
            <attributeGroup ref="ncl:SampleAnchorAttrs" />
        </extension>
    </complexContent>
</complexType>

<complexType name="textAnchorPrototype">
    <complexContent>
        <extension base="ncl:anchorPrototype">
            <attribute name="anchorLabel" type="string"/>
            <attributeGroup ref="ncl:TextAnchorAttrs" />
        </extension>
    </complexContent>
</complexType>

<complexType name="compositeAnchorPrototype">
    <complexContent>
        <extension base="ncl:anchorPrototype">
            <attribute name="componentList" type="string"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="compositePortPrototype">
    <complexContent>
        <extension base="ncl:anchorPrototype">
            <attribute name="anchorLabel" type="string"/>
            <attributeGroup ref="ncl:CompositePortAttrs" />
        </extension>
    </complexContent>
</complexType>
```

```
    <complexType name="componentAnchorPrototype">
        <complexContent>
            <extension base="ncl:anchorPrototype">
                <attribute name="coords" type="string"/>
                <attributeGroup ref="ncl:TemporalAnchorAttrs" />
                <attributeGroup ref="ncl:TextAnchorAttrs" />
                <attributeGroup ref="ncl:SampleAnchorAttrs" />
                <attribute name="anchorLabel" type="string"/>
            </extension>
        </complexContent>
    </complexType>

    <complexType name="attributeInterfacePrototype">
        <attribute name="name" type="string"/>
    </complexType>

    <complexType name="portSwitchPrototype">
    </complexType>

    <!-- declare global elements in this module -->
    <element name="area" type="ncllang:anchorType"
substitutionGroup="ncllang:area"/>

    <element name="compositeArea" type="ncllang:compositeAnchorType"
substitutionGroup="ncllang:compositeArea"/>
    <element name="port" type="ncllang:compositePortType"
substitutionGroup="ncllang:port"/>

    <element name="attribute" type="ncllang:attributeInterfaceType"
substitutionGroup="ncllang:attribute"/>

    <element name="portSwitch" type="ncllang:portSwitchType"
substitutionGroup="ncllang:portSwitch"/>

</schema>
```

10.19. NCL-Language.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
           xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        elementFormDefault="qualified">

    <!-- import the ncl namespaces -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
```

```
    <!-- ================================================================== -->
    <!-- CoreAttrs attribute group used on all NCL profile elements         -->
    <!-- ================================================================== -->
    <attributeGroup name="CoreAttrs">
        <attribute name="id" type="ID" />
    </attributeGroup>


    <!-- ================================================================== -->
    <!-- Structure Functionality                                            -->
    <!-- ================================================================== -->

    <!-- ================================================================== -->
    <!-- define the top down structure of an NCL language document.          -->
    <!-- ================================================================== -->

    <!-- top level ncl element and content model -->
    <element name="ncl" type="ncllang:nclType"/>
    <complexType name="nclType">
        <complexContent>
            <extension base="ncl:nclPrototype">
                <sequence>
                    <element ref="ncllang:head" minOccurs="0" maxOccurs="1"/>
                    <element ref="ncllang:body" minOccurs="1" maxOccurs="1"/>
                </sequence>
                <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- head element and content model -->
    <element name="head" type="ncllang:headType"/>
    <complexType name="headType">
        <complexContent>
            <extension base="ncl:headPrototype">
               <!--
               <sequence>
                  <element ref="ncllang:layout" minOccurs="0" maxOccurs="1"/>
                  <element ref="ncllang:descriptorBase" minOccurs="0"
maxOccurs="1"/>
                  <element ref="ncllang:costFunctionBase" minOccurs="0"
maxOccurs="1"/>
                  <element ref="ncllang:presentationRuleBase" minOccurs="0"
maxOccurs="1"/>
               </sequence>
               -->
               <choice minOccurs="0" maxOccurs="unbounded">
                    <group ref="ncllang:presentationRuleBaseGroup"/>
                    <group ref="ncllang:descriptorBaseGroup"/>
                    <group ref="ncllang:layoutGroup"/>
```

```
                <group ref="ncllang:costFunctionBaseGroup"/>
            </choice>
        </extension>
    </complexContent>
</complexType>

<!-- body element and content model -->
<element name="body" type="ncllang:bodyType"/>
<complexType name="bodyType">
    <complexContent>
        <extension base="ncl:bodyPrototype">
            <choice minOccurs="0" maxOccurs="unbounded">
                <group ref="ncllang:compositeInterfaceElementGroup"/>
                <group ref="ncllang:mediaContentGroup"/>
                <group ref="ncllang:compositionContentGroup"/>
                <group ref="ncllang:linkBaseElementGroup"/>
                <group ref="ncllang:contentControlGroup"/>
            </choice>
        </extension>
    </complexContent>
</complexType>

<group name="contentControlGroup">
    <choice>
        <element ref="ncllang:switch"/>
    </choice>
</group>

<!-- ================================================================== -->
<!-- Layout Functionality                                               -->
<!-- ================================================================== -->

<!-- layout element and content model -->
<element name="layout" type="ncllang:layoutType"/>
<complexType name="layoutType">
    <complexContent>
        <extension base="ncl:layoutPrototype">
            <choice maxOccurs="unbounded">
                <group ref="ncllang:topLayoutGroup"/>
            </choice>
        </extension>
    </complexContent>
</complexType>

<group name="layoutGroup">
    <choice>
        <element ref="ncllang:layout"/>
    </choice>
</group>

<!-- topLayout element and content model -->
```

```
    <element name="topLayout" type="ncllang:topLayoutType"/>
    <complexType name="topLayoutType">
        <complexContent>
           <extension base="ncl:topLayoutPrototype">
                <choice maxOccurs="unbounded">
                    <group ref="ncllang:regionGroup"/>
                </choice>
                <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- toplayout groups -->
    <group name="topLayoutGroup">
        <choice>
            <element ref="ncllang:topLayout"/>
        </choice>
    </group>


    <!-- region element and content model -->
    <element name="region" type="ncllang:regionType"/>

    <complexType name="regionType">
        <complexContent>
            <extension base="ncl:regionPrototype">
               <!--
                <sequence>
                    <element name="region" type="ncllang:regionType" minOccurs="0"
maxOccurs="unbounded"/>
                </sequence>
                -->
                <choice minOccurs="0" maxOccurs="unbounded">
                    <group ref="ncllang:regionGroup"/>
                </choice>
                <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- region groups -->
    <group name="regionGroup">
        <choice>
            <element ref="ncllang:region"/>
        </choice>
    </group>

    <!-- ================================================================= -->
    <!-- Presentation Specification Functionality                          -->
    <!-- ================================================================= -->
```

```
    <!-- descriptor element and content model -->
    <element name="descriptor" type="ncllang:descriptorType"/>
    <complexType name="descriptorType">
        <complexContent>
            <extension base="ncl:descriptorPrototype">
                <sequence>
                    <element ref="ncllang:descriptorParam" minOccurs="0"
maxOccurs="unbounded" />
                </sequence>
                <attributeGroup ref="ncllang:CoreAttrs"/>
                <attributeGroup ref="ncl:regionAttrs"/>
                <attributeGroup ref="ncl:TimingAttrs"/>
                <attributeGroup ref="ncl:systemTestAttrs"/>
                <attributeGroup ref="ncl:costFunctionAttrs"/>
                <attributeGroup ref="ncl:explicitDurAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <element name="descriptorParam" type="ncllang:descriptorParamType"/>
    <complexType name="descriptorParamType">
        <complexContent>
            <extension base="ncl:descriptorParamPrototype">
            </extension>
        </complexContent>
    </complexType>

    <!-- descriptor groups -->
    <group name="descriptorGroup">
        <choice>
            <element ref="ncllang:descriptor"/>
        </choice>
    </group>

    <!-- descriptorBase element and content model -->
    <element name="descriptorBase" type="ncllang:descriptorBaseType"/>
    <complexType name="descriptorBaseType">
        <complexContent>
            <extension base="ncl:descriptorBasePrototype">
                <choice maxOccurs="unbounded">
                    <group ref="ncllang:descriptorGroup"/>
                    <group ref="ncllang:descriptorSwitchGroup"/>
                </choice>
            </extension>
        </complexContent>
    </complexType>

    <!-- descriptorBase groups -->
    <group name="descriptorBaseGroup">
        <choice>
            <element ref="ncllang:descriptorBase"/>
```

```
        </choice>
    </group>

    <!-- componentPresentation element and content model -->
    <element name="componentPresentation"
type="ncllang:componentPresentationType"/>
    <complexType name="componentPresentationType">
        <complexContent>
            <extension base="ncl:componentPresentationPrototype">
            </extension>
        </complexContent>
    </complexType>

    <!-- componentPresentation groups -->
    <group name="componentPresentationGroup">
        <choice>
            <element ref="ncllang:componentPresentation"/>
        </choice>
    </group>

    <!-- bindDescriptor element and content model -->
    <element name="bindDescriptor" type="ncllang:bindDescriptorType"/>
    <complexType name="bindDescriptorType">
        <complexContent>
            <extension base="ncl:bindDescriptorPrototype">
            </extension>
        </complexContent>
    </complexType>

    <!-- bindDescriptor groups -->
    <group name="bindDescriptorGroup">
        <choice>
            <element ref="ncllang:bindDescriptor"/>
        </choice>
    </group>
    <!-- ================================================================= -->
    <!-- Component Functionality                                           -->
    <!-- ================================================================= -->

    <!-- media elements and content model -->
    <element name="text" type="ncllang:mediaType"/>
    <element name="img" type="ncllang:mediaType"/>
    <element name="audio" type="ncllang:mediaType"/>
    <element name="animation" type="ncllang:mediaType"/>
    <element name="video" type="ncllang:mediaType"/>
    <element name="textstream" type="ncllang:mediaType"/>
    <element name="ref" type="ncllang:mediaType"/>

    <complexType name="mediaType">
        <complexContent>
            <extension base="ncl:mediaPrototype">
```

```
                    <choice minOccurs="0" maxOccurs="unbounded">
                        <group ref="ncllang:mediaInterfaceElementGroup"/>
                </choice>
                <attributeGroup ref="ncllang:CoreAttrs"/>
                <attributeGroup ref="ncl:descriptorAttrs"/>
                <attributeGroup ref="ncl:CompositeTemplateUseAttrs"/>
                <attributeGroup ref="ncl:systemTestAttrs"/>
                <attributeGroup ref="ncl:implicitDurAttrs"/>
            </extension>
        </complexContent>
    </complexType>


    <!-- media groups -->
    <group name="mediaContentGroup">
        <choice>
            <element ref="ncllang:text"/>
            <element ref="ncllang:img"/>
            <element ref="ncllang:audio"/>
            <element ref="ncllang:animation"/>
            <element ref="ncllang:video"/>
            <element ref="ncllang:textstream"/>
            <element ref="ncllang:ref"/>
        </choice>
    </group>

    <!-- ================================================================= -->
    <!-- composition element                                               -->
    <!-- ================================================================= -->

    <!-- composition elements and content model -->
    <element name="composition" type="ncllang:compositionType"/>

    <complexType name="compositionType">
        <complexContent>
            <extension base="ncl:compositionPrototype">
                <choice minOccurs="0" maxOccurs="unbounded">
                    <group ref="ncllang:compositeInterfaceElementGroup"/>
                    <group ref="ncllang:componentPresentationGroup"/>
                    <group ref="ncllang:bindDescriptorGroup"/>
                    <group ref="ncllang:mediaContentGroup"/>
                    <group ref="ncllang:compositionContentGroup"/>
                    <group ref="ncllang:linkBaseElementGroup"/>
                    <group ref="ncllang:contentControlGroup"/>
                </choice>
                <attributeGroup ref="ncllang:CoreAttrs"/>
                <attributeGroup ref="ncl:descriptorAttrs"/>
                <attributeGroup ref="ncl:systemTestAttrs"/>
                <attributeGroup ref="ncl:CompositeTemplateUseAttrs"/>
            </extension>
        </complexContent>
```

```
    </complexType>

    <!-- composition component groups -->
    <group name="compositionContentGroup">
        <choice>
            <element ref="ncllang:composition"/>
        </choice>
    </group>

    <!-- ================================================================= -->
    <!-- Interface Functionality                                           -->
    <!-- ================================================================= -->

    <!-- area elements and content model -->
    <element name="area" type="ncllang:anchorType"/>

    <complexType name="anchorType">
        <complexContent>
            <extension base="ncl:componentAnchorPrototype">
                <attributeGroup ref="ncllang:CoreAttrs"/>
                <attributeGroup ref="ncl:CompositeTemplateUseAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- attribute elements and content model -->
    <element name="attribute" type="ncllang:attributeInterfaceType"/>

    <complexType name="attributeInterfaceType">
        <complexContent>
            <extension base="ncl:attributeInterfacePrototype">
                <attributeGroup ref="ncllang:CoreAttrs"/>
                <attributeGroup ref="ncl:CompositeTemplateUseAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- media interface element groups -->
    <group name="mediaInterfaceElementGroup">
        <choice>
            <element ref="ncllang:area"/>
            <element ref="ncllang:attribute"/>
        </choice>
    </group>

    <!-- compositeArea elements and content model -->
    <element name="compositeArea" type="ncllang:compositeAnchorType"/>

    <complexType name="compositeAnchorType">
        <complexContent>
            <extension base="ncl:compositeAnchorPrototype">
```

```
            <attributeGroup ref="ncllang:CoreAttrs"/>
            <attributeGroup ref="ncl:CompositeTemplateUseAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- port elements and content model -->
<element name="port" type="ncllang:compositePortType"/>

<complexType name="compositePortType">
    <complexContent>
        <extension base="ncl:compositePortPrototype">
            <attributeGroup ref="ncllang:CoreAttrs"/>
            <attributeGroup ref="ncl:CompositeTemplateUseAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- composite interface element groups -->
<group name="compositeInterfaceElementGroup">
    <choice>
        <element ref="ncllang:compositeArea"/>
        <element ref="ncllang:port"/>
    </choice>
</group>

<!-- portSwitch elements and content model -->
<element name="portSwitch" type="ncllang:portSwitchType"/>

<complexType name="portSwitchType">
    <complexContent>
        <extension base="ncl:portSwitchPrototype">
            <attributeGroup ref="ncllang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- switch interface element groups -->
<group name="switchInterfaceElementGroup">
    <choice>
        <element ref="ncllang:port"/>
        <element ref="ncllang:portSwitch"/>
    </choice>
</group>

<!-- ================================================================= -->
<!-- Linking Functionality                                             -->
<!-- ================================================================= -->

<!-- bind element and content model -->
<element name="bind" type="ncllang:bindType"/>
```

```
<complexType name="bindType">
    <complexContent>
        <extension base="ncl:bindPrototype">
            <attributeGroup ref="ncl:descriptorAttrs"/>
        </extension>
    </complexContent>
</complexType>

<group name="bindGroup">
    <choice>
        <element ref="ncllang:bind"/>
    </choice>
</group>

<!-- linkbase element and content model -->
<element name="linkBase" type="ncllang:linkBaseType"/>
<complexType name="linkBaseType">
    <complexContent>
        <extension base="ncl:linkBasePrototype">
            <choice minOccurs="1" maxOccurs="unbounded">
                <group ref="ncllang:lrefGroup"/>
                <group ref="ncllang:linkGroup"/>
            </choice>
            <attributeGroup ref="ncllang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- lref element and content model -->
<element name="lref" type="ncllang:lrefType"/>
<complexType name="lrefType">
    <complexContent>
        <extension base="ncl:lrefPrototype">
            <attributeGroup ref="ncllang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<group name="lrefGroup">
    <choice>
        <element ref="ncllang:lref"/>
    </choice>
</group>

<group name="linkGroup">
    <choice>
        <element ref="ncllang:link"/>
        <!-- <element ref="ncllang:lref"/> -->
    </choice>
</group>
```

```
    <!-- linkbase element groups -->
    <group name="linkBaseElementGroup">
        <choice>
            <element ref="ncllang:linkBase"/>
        </choice>
    </group>

    <!-- param element and content model -->
    <element name="param" type="ncllang:paramType"/>
    <complexType name="paramType">
        <complexContent>
            <extension base="ncl:paramPrototype">
                <attributeGroup ref="ncl:descriptorAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <group name="paramGroup">
        <choice>
            <element ref="ncllang:param"/>
        </choice>
    </group>

    <!-- link element and content model -->
    <element name="link" type="ncllang:linkType"/>
    <complexType name="linkType">
        <complexContent>
            <extension base="ncl:linkPrototype">
                <choice maxOccurs="unbounded" minOccurs="0">
                  <group ref="ncllang:bindGroup"/>
                  <group ref="ncllang:paramGroup"/>
                </choice>
                      <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- ================================================================= -->
    <!-- Connector Functionality                                           -->
    <!-- ================================================================= -->

           <element name="compositeConnector"
type="ncllang:compositeConnectorType"/>
           <complexType name="compositeConnectorType">
               <complexContent>
                   <extension base="ncl:compositeConnectorPrototype">
                       <sequence>
                           <element ref="ncllang:role" minOccurs="2"
maxOccurs="unbounded" />
                           <element ref="ncllang:glue" maxOccurs="1" />
                       </sequence>
```

```
                    <attributeGroup ref="ncllang:CoreAttrs"/>
                </extension>
            </complexContent>
        </complexType>

<!-- role element and content model (for composite connectors) -->
<element name="role" type="ncllang:compositeRoleType"/>
<complexType name="compositeRoleType">
    <complexContent>
        <extension base="ncl:compositeRolePrototype">
        </extension>
    </complexContent>
</complexType>

<!-- glue element and content model (for composite connectors) -->
<element name="glue" type="ncllang:compositeGlueType"/>
<complexType name="compositeGlueType">
    <complexContent>
        <extension base="ncl:compositeGluePrototype">
           <choice minOccurs="0" maxOccurs="unbounded">
                <group ref="ncllang:mediaContentGroup"/>
                <group ref="ncllang:compositionContentGroup"/>
                <group ref="ncllang:linkBaseElementGroup"/>
                <element ref="ncllang:partialLinkBase"/>
            </choice>
        </extension>
    </complexContent>
</complexType>

<element name="partialLinkBase" type="ncllang:partialLinkBaseType"/>

<complexType name="partialLinkBaseType">
    <complexContent>
        <extension base="ncl:partialLinkBasePrototype">
            <sequence>
                <element ref="ncllang:link" maxOccurs="unbounded" />
            </sequence>
        </extension>
    </complexContent>
</complexType>

<!-- ================================================================= -->
<!-- Presentation Control Functionality                                -->
<!-- ================================================================= -->

<!-- switch element and content model -->
<element name="switch" type="ncllang:switchType"/>
<complexType name="switchType">
    <complexContent>
        <extension base="ncl:switchPrototype">
            <choice maxOccurs="unbounded">
```

```
                <group ref="ncllang:switchInterfaceElementGroup"/>
                <group ref="ncllang:mediaContentGroup"/>
                <group ref="ncllang:compositionContentGroup"/>
                <group ref="ncllang:contentControlGroup"/>
                <group ref="ncllang:bindRuleGroup"/>
            </choice>
            <attributeGroup ref="ncllang:CoreAttrs"/>
            <attributeGroup ref="ncl:systemTestAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- descriptorSwitch element and content model -->
<element name="descriptorSwitch" type="ncllang:descriptorSwitchType"/>
<complexType name="descriptorSwitchType">
    <complexContent>
        <extension base="ncl:descriptorSwitchPrototype">
            <choice minOccurs="0" maxOccurs="unbounded">
                <group ref="ncllang:descriptorGroup"/>
                <group ref="ncllang:bindRuleGroup"/>
            </choice>
            <attributeGroup ref="ncllang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- descriptorSwitch groups -->
<group name="descriptorSwitchGroup">
    <choice>
        <element ref="ncllang:descriptorSwitch"/>
    </choice>
</group>

<!-- bindRule element and content model -->
<element name="bindRule" type="ncllang:bindRuleType"/>
<complexType name="bindRuleType">
    <complexContent>
        <extension base="ncl:bindRulePrototype">
            <attributeGroup ref="ncllang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<group name="bindRuleGroup">
    <choice>
        <element ref="ncllang:bindRule"/>
    </choice>
</group>


<!-- ================================================================= -->
```

```
    <!-- Presentation Rule Base Functionality                              -->
    <!-- ================================================================= -->

    <!-- presentationRule element and content model -->
    <element name="presentationRule" type="ncllang:presentationRuleType"/>
    <complexType name="presentationRuleType">
        <complexContent>
            <extension base="ncl:presentationRulePrototype">
                <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <element name="compositePresentationRule"
type="ncllang:compositePresentationRuleType"/>
    <complexType name="compositePresentationRuleType">
        <complexContent>
            <extension base="ncl:compositePresentationRulePrototype">
                        <choice maxOccurs="unbounded">
                    <group ref="ncllang:presentationRuleGroup"/>
                    <group ref="ncllang:compositePresentationRuleGroup"/>
                       </choice>
                <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- presentationRuleBase element and content model -->
    <element name="presentationRuleBase" type="ncllang:presentationRuleBaseType"/>
    <complexType name="presentationRuleBaseType">
        <complexContent>
            <extension base="ncl:presentationRuleBasePrototype">
                <choice maxOccurs="unbounded">
                    <group ref="ncllang:presentationRuleGroup"/>
                    <group ref="ncllang:compositePresentationRuleGroup"/>
                </choice>
            </extension>
        </complexContent>
    </complexType>

    <group name="presentationRuleBaseGroup">
        <choice>
            <element ref="ncllang:presentationRuleBase"/>
        </choice>
    </group>

    <!-- presentationRuleGroup groups -->
    <group name="presentationRuleGroup">
        <choice>
            <element ref="ncllang:presentationRule"/>
        </choice>
```

```
    </group>

    <group name="compositePresentationRuleGroup">
        <choice>
            <element ref="ncllang:compositePresentationRule"/>
        </choice>
    </group>

    <!-- ================================================================= -->
    <!-- Cost Function Base Functionality                                  -->
    <!-- ================================================================= -->

    <!-- costFunction element and content model -->
    <element name="costFunction" type="ncllang:costFunctionType"/>
    <complexType name="costFunctionType">
        <complexContent>
            <extension base="ncl:costFunctionPrototype">
                <sequence>
                    <element ref="ncllang:costFunctionParam" minOccurs="0"
maxOccurs="unbounded" />
                </sequence>
                <attributeGroup ref="ncllang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- costFunction groups -->
    <group name="costFunctionGroup">
        <choice>
            <element ref="ncllang:costFunction"/>
        </choice>
    </group>

    <element name="costFunctionParam" type="ncllang:costFunctionParamType"/>
    <complexType name="costFunctionParamType">
        <complexContent>
            <extension base="ncl:costFunctionParamPrototype">
            </extension>
        </complexContent>
    </complexType>

    <!-- costFunctionBase element and content model -->
    <element name="costFunctionBase" type="ncllang:costFunctionBaseType"/>
    <complexType name="costFunctionBaseType">
        <complexContent>
            <extension base="ncl:costFunctionBasePrototype">
                <choice minOccurs="0" maxOccurs="unbounded">
                    <group ref="ncllang:costFunctionGroup"/>
                </choice>
            </extension>
        </complexContent>
```

```
    </complexType>

    <!-- costFunctionBase groups -->
    <group name="costFunctionBaseGroup">
        <choice>
            <element ref="ncllang:costFunctionBase"/>
        </choice>
    </group>
</schema>
```

10.20. NCL-layout.xsd

```
schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- utility type for fit attribute values -->
    <simpleType name="fitAttributeType">
        <restriction base="string">
            <enumeration value="fill"/>
            <enumeration value="hidden"/>
            <enumeration value="meet"/>
            <enumeration value="scroll"/>
            <enumeration value="slice"/>
        </restriction>
    </simpleType>

    <!-- define the layout element prototype -->
    <complexType name="layoutPrototype">
        <attribute name="type" type="string" use="optional"  default="text/ncl-
layout"/>
        <attribute name="src" type="anyURI" use="optional" />
    </complexType>

    <!-- define the Layout Component element prototype -->
    <complexType name="LayoutComponentPrototype">
        <attribute name="title" type="string" use="optional" />
        <attribute name="backgroundColor" type="string" use="optional" />
        <!-- novo nome equivalente para backgroundColor -->
        <attribute name="background" type="string" use="optional" />
        <attribute name="height" type="string" use="optional" />
        <attribute name="left" type="string" use="optional"  />
        <attribute name="top" type="string" use="optional"  />
        <attribute name="width" type="string" use="optional"  />
```

```
        <attribute name="zIndex" type="integer" use="optional" />
        <attribute name="visible" type="boolean" use="optional" />

        <attribute name="close" use="optional"  default="onRequest">
            <simpleType>
                <restriction base="string">
                    <enumeration value="onRequest"/>
                    <enumeration value="whenNotActive"/>
                </restriction>
            </simpleType>
        </attribute>

        <attribute name="open" use="optional"  default="onStart">
            <simpleType>
                <restriction base="string">
                    <enumeration value="onStart"/>
                    <enumeration value="whenActive"/>
                </restriction>
            </simpleType>
        </attribute>

    </complexType>

    <!-- define the topLayout element prototype -->
    <complexType name="topLayoutPrototype">
        <complexContent>
            <extension base="ncl:LayoutComponentPrototype">
                <attribute name="movable" type="boolean" use="optional"  />
                <attribute name="resizable" type="boolean" use="optional"  />
            </extension>
        </complexContent>
    </complexType>

    <!-- define the region element prototype -->
    <complexType name="regionPrototype">
        <complexContent>
            <extension base="ncl:LayoutComponentPrototype">
                <attribute name="fit" use="optional"  default="hidden"
type="ncl:fitAttributeType"/>
                <attribute name="showBackground" use="optional"  default="always">
                   <simpleType>
                      <restriction base="string">
                         <enumeration value="always"/>
                         <enumeration value="whenActive"/>
                      </restriction>
                   </simpleType>
                </attribute>
            </extension>
        </complexContent>
    </complexType>
```

```
    <!-- define the global region attribute -->
    <attribute name="region" type="IDREF"/>

    <!-- define the region attributeGroup -->
    <attributeGroup name="regionAttrs">
        <attribute name="region" type="IDREF" use="optional"/>
    </attributeGroup>

    <!-- define the global layout elements -->
    <element name="layout" type="ncllang:layoutType"
substitutionGroup="ncllang:layout"/>
    <element name="topLayout" type="ncllang:topLayoutType"
substitutionGroup="ncllang:topLayout"/>
    <element name="region" type="ncllang:regionType"
substitutionGroup="ncllang:region"/>

</schema>
```

10.21. NCL-link.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>



    <attribute name="xconnector" type="anyURI"/>
    <attribute name="name" type="string" />
    <attribute name="value" type="anySimpleType"/>
    <attribute name="role" type="string" />
    <!-- <attribute name="component" type="anyURI" /> -->
    <attribute name="port" type="string" />
    <!-- <attribute name="src" type="anyURI" /> -->

    <complexType name="linkPrototype">
        <attribute name="xconnector" type="anyURI" use="required"/>
    </complexType>

    <complexType name="paramPrototype">
         <attribute name="name" type="string" use="required"/>
         <attribute name="value" type="anySimpleType" use="required"/>
    </complexType>

    <complexType name="bindPrototype">
```

```
        <attribute name="role" type="string" use="required"/>
        <attribute name="component" type="anyURI" use="required"/>
        <attribute name="port" type="string" />
        <attribute name="descriptor" type="IDREF" />
    </complexType>

    <complexType name="lrefPrototype">
        <attribute name="src" type="anyURI" use="required"/>
    </complexType>

    <complexType name="linkBasePrototype">
    </complexType>

    <element name="param" type="ncllang:paramType"
substitutionGroup="ncllang:param"/>
    <element name="bind" type="ncllang:bindType"
substitutionGroup="ncllang:bind"/>

    <element name="link" type="ncllang:linkType"
substitutionGroup="ncllang:link"/>
    <element name="linkBase" type="ncllang:linkBaseType"
substitutionGroup="ncllang:linkBase"/>
    <element name="lref" type="ncllang:lrefType"
substitutionGroup="ncllang:lref"/>

</schema>
```

10.22. NCL-Linking.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:Linking="http://www.telemidia.puc-rio.br/specs/xml/ncl/Linking"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Linking"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <element name="param" type="ncllang:paramType"
substitutionGroup="ncllang:param"/>
    <element name="bind" type="ncllang:bindType"
substitutionGroup="ncllang:bind"/>

    <element name="link" type="ncllang:linkType"
```

```
substitutionGroup="ncllang:link"/>
    <element name="linkBase" type="ncllang:linkBaseType"
substitutionGroup="ncllang:linkBase"/>
    <element name="lref" type="ncllang:lrefType"
substitutionGroup="ncllang:lref"/>

    <!-- declare global attributes in this module -->
    <attribute name="xconnector" type="anyURI"/>
    <attribute name="name" type="string" />
    <attribute name="value" type="anySimpleType"/>
    <attribute name="role" type="string" />
    <attribute name="component" type="anyURI" />
    <attribute name="port" type="string" />
    <attribute name="src" type="anyURI" />

</schema>
```

10.23. NCL-MediaInterface.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:MediaInterface="http://www.telemidia.puc-
rio.br/specs/xml/ncl/MediaInterface"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/MediaInterface"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="area" type="ncllang:anchorType"
substitutionGroup="ncllang:area"/>

    <!-- declare global attributes in this module -->
    <attribute name="coords" type="string"/>
    <attribute name="begin" type="string"/>
    <attribute name="end" type="string"/>
    <attribute name="dur" type="unsignedLong"/>
    <attribute name="text" type="string"/>
    <attribute name="position" type="unsignedLong"/>
    <attribute name="first" type="unsignedLong"/>
    <attribute name="last" type="unsignedLong"/>
    <attribute name="implicitDur" type="string"/>
</schema>
```

10.24. NCL-presentation.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- define the descriptor global attribute -->
    <attribute name="descriptor" type="IDREF"/>

    <!-- define the descriptor attributeGroup -->
    <attributeGroup name="descriptorAttrs">
        <attribute name="descriptor" type="IDREF" use="optional"/>
    </attributeGroup>

    <!-- define the audio descriptor attribute -->
    <attribute name="soundLevel" type="string"/>
    <attribute name="balanceLevel" type="string"/>
    <attribute name="trebleLevel" type="string"/>
    <attribute name="bassLevel" type="string"/>

    <!-- define the soundLevel attributeGroup -->
    <attributeGroup name="AudioDescriptorAttrs">
        <attribute name="soundLevel" type="string" use="optional"/>
        <attribute name="balanceLevel" type="string" use="optional"/>
        <attribute name="trebleLevel" type="string" use="optional"/>
        <attribute name="bassLevel" type="string" use="optional"/>
    </attributeGroup>

    <complexType name="descriptorPrototype">
        <attribute name="player" type="string" use="optional"/>
        <attribute name="enableTimeBar" use="optional">
          <simpleType>
             <restriction base="string">
                <enumeration value="on"/>
                <enumeration value="off"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name="style" type="anyURI" use="optional"/>
        <attributeGroup ref="ncl:AudioDescriptorAttrs" />
        <attribute name="nodeRule" type="IDREF" use="optional"/>
    </complexType>
```

```
    <complexType name="descriptorBasePrototype">
    </complexType>

    <complexType name="componentPresentationPrototype">
        <attribute name="component" type="IDREF" use="required"/>
        <attribute name="descriptor" type="IDREF" use="required"/>
    </complexType>

    <complexType name="bindDescriptorPrototype">
        <attribute name="component" type="IDREF" use="required"/>
        <attribute name="descriptor" type="IDREF" use="required"/>
    </complexType>

    <complexType name="descriptorParamPrototype">
         <attribute name="name" type="string" use="required"/>
         <attribute name="value" type="anySimpleType" use="required"/>
    </complexType>

    <!-- declare global elements in this module -->
    <element name="descriptor" type="ncllang:descriptorType"
substitutionGroup="ncllang:descriptor"/>
    <element name="descriptorParam" type="ncllang:descriptorParamType"
substitutionGroup="ncllang:descriptorParam"/>
    <element name="descriptorBase" type="ncllang:descriptorBaseType"
substitutionGroup="ncllang:descriptorBase"/>
    <element name="componentPresentation" type="ncllang:componentPresentationType"
substitutionGroup="ncllang:componentPresentation"/>
    <element name="bindDescriptor" type="ncllang:bindDescriptorType"
substitutionGroup="ncllang:bindDescriptor"/>
</schema>
```

10.25. NCL-struct.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- define the structure module attribute group -->
    <attributeGroup name="structureModuleAttrs">
    </attributeGroup>

    <!-- define the ncl element prototype -->
    <complexType name="nclPrototype">
    </complexType>
```

```
    <!-- define the head element prototype -->
    <complexType name="headPrototype">
    </complexType>

    <!-- define the body element prototype -->
    <complexType name="bodyPrototype">
    </complexType>

    <!-- declare global elements -->
    <element name="ncl" type="ncllang:nclType" substitutionGroup="ncllang:ncl"/>
    <element name="head" type="ncllang:headType"
substitutionGroup="ncllang:head"/>
    <element name="body" type="ncllang:bodyType"
substitutionGroup="ncllang:body"/>

</schema>
```

10.26. NCL-Structure.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:Structure="http://www.telemidia.puc-rio.br/specs/xml/ncl/Structure"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Structure"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="ncl" type="ncllang:nclType" substitutionGroup="ncllang:ncl"/>
    <element name="head" type="ncllang:headType"
substitutionGroup="ncllang:head"/>
    <element name="body" type="ncllang:bodyType"
substitutionGroup="ncllang:body"/>
</schema>
```

10.27. NCL-SwitchInterface.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:SwitchInterface="http://www.telemidia.puc-
rio.br/specs/xml/ncl/SwitchInterface"
```

```
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/SwitchInterface"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="portSwitch" type="ncllang:portSwitchType"
substitutionGroup="ncllang:portSwitch"/>
</schema>
```

10.28. NCL-TestRules.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:TestRules="http://www.telemidia.puc-rio.br/specs/xml/ncl/TestRules"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/TestRules"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="presentationRule" type="ncllang:presentationRuleType"
substitutionGroup="ncllang:presentationRule"/>
    <element name="presentationRuleBase" type="ncllang:presentationRuleBaseType"
substitutionGroup="ncllang:presentationRuleBase"/>
    <element name="compositePresentationRule"
type="ncllang:compositePresentationRuleType"
substitutionGroup="ncllang:compositePresentationRule"/>
</schema>
```

10.29. NCL-timing.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >
```

```
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL-language.xsd"/>

    <!-- define the dur, min, max timing attributes -->
    <attribute name="dur" type="string"/>
    <attribute name="min" type="string"/>
    <attribute name="max" type="string"/>
    <attribute name="implicitDur" type="string"/>

    <!-- define the TimingAttrs attribute group -->
    <attributeGroup name="TimingAttrs">
        <attribute name="dur" type="string" use="optional"/>
        <attribute name="min" type="string" use="optional"/>
        <attribute name="max" type="string" use="optional"/>
        <attribute name="implicitDur" type="string" use="optional"/>
    </attributeGroup>

    <!-- declare global attributes in this module -->
    <attribute name="costFunction" type="IDREF"/>

    <!-- define the costFunctionAttrs attribute group -->
       <attributeGroup name="costFunctionAttrs">
       <attribute name="costFunction" type="IDREF" use="optional"/>
    </attributeGroup>

    <!-- define the explicitDur attribute group -->
       <attributeGroup name="explicitDurAttrs">
       <attribute name="explicitDur" type="string" use="optional"/>
    </attributeGroup>

    <!-- define the implicitDur attribute group -->
       <attributeGroup name="implicitDurAttrs">
       <attribute name="implicitDur" type="string" use="optional"/>
    </attributeGroup>

    <complexType name="costFunctionPrototype">
        <!--<attribute name="type" type="string" use="required"/> -->
        <attribute name="type" type="string" use="optional"/>
        <attribute name="ref" type="string" use="optional"/>
        <attribute name="deltaShrink" type="string" use="optional"/>
        <attribute name="deltaStretch" type="string" use="optional"/>
        <attribute name="minDurCost" type="string" use="optional"/>
        <attribute name="maxDurCost" type="string" use="optional"/>
    </complexType>

    <complexType name="costFunctionBasePrototype">
    </complexType>

    <complexType name="costFunctionParamPrototype">
         <attribute name="name" type="string" use="required"/>
```

```
        <attribute name="value" type="anySimpleType" use="required"/>
    </complexType>

    <!-- declare global elements in this module -->
    <element name="costFunction" type="ncllang:costFunctionType"
substitutionGroup="ncllang:costFunction"/>
    <element name="costFunctionBase" type="ncllang:costFunctionBaseType"
substitutionGroup="ncllang:costFunctionBase"/>
    <element name="costFunctionParam" type="ncllang:costFunctionParamType"
substitutionGroup="ncllang:costFunctionParam"/>
</schema>
```

10.30. NCL-XTemplateUse.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ncl="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
        xmlns:ncllang="http://www.telemidia.puc-rio.br/specs/xml/ncl/Language"
        xmlns:TemplateUse="http://www.telemidia.puc-
rio.br/specs/xml/ncl/TemplateUse"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/ncl/TemplateUse"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/ncl/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/NCL21.xsd"/>

    <!-- declare global attributes in this module -->
    <attribute name="xtemplate" type="anyURI"/>
    <attribute name="label" type="string"/>
</schema>
```

11
Apêndice C

Especificação da linguagem XConnector 2.1.

11.1.
XConnector21.xsd

```
<schema
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XConnector/"
xmlns:xconnector="http://www.telemidia.puc-rio.br/specs/xml/XConnector/"
elementFormDefault="qualified" attributeFormDefault="unqualified">

<complexType name="HypermediaConnector" abstract="true">
   <sequence>
      <element name="param" type="xconnector:ParameterType" minOccurs="0"
maxOccurs="unbounded"/>
   </sequence>
   <attribute name="id" type="ID" use="required"/>
   <attribute name="name" type="string"/>
   <attribute name="description" type="string"/>
</complexType>

<complexType name="ParameterType" >
   <attribute name="name" type="string" use="required"/>
   <attribute name="type" type="string" use="required"/>
</complexType>

<complexType name="CausalHypermediaConnector">
   <complexContent>
      <extension base="xconnector:HypermediaConnector">
         <sequence>
            <choice maxOccurs="unbounded">
               <element name="conditionRole" type="xconnector:ConditionRole"/>
               <element name="propertyRole" type="xconnector:PropertyRole" />
            </choice>
            <element name="actionRole" type="xconnector:ActionRole"
maxOccurs="unbounded"/>
            <element name="causalGlue" type="xconnector:CausalGlue"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

```
<complexType name="ConstraintHypermediaConnector">
   <complexContent>
      <extension base="xconnector:HypermediaConnector">
         <sequence>
            <element name="propertyRole" type="xconnector:PropertyRole"
minOccurs="2" maxOccurs="unbounded"/>
            <element name="constraintGlue" type="xconnector:ConstraintGlue"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<complexType name="Role" abstract="true">
   <attribute name="id" type="string" use="required"/>
   <attribute name="eventType" type="xconnector:EventType" use="required"/>
   <attribute name="attributeName" type="string" />
   <attributeGroup ref="xconnector:Cardinality"/>
</complexType>

<complexType name="ActionRole">
   <complexContent>
      <extension base="xconnector:Role">
         <attribute name="actionType" type="xconnector:SimpleAction"
use="required"/>
         <attribute name="show" type="xconnector:ShowType" default="new"/>
         <attribute name="delay" type="xconnector:unsignedLongParamUnion"
default="0"/>
         <attribute name="repeat" type="xconnector:repeatUnion" default="0"/>
         <attribute name="delayBetweenRep"
type="xconnector:unsignedLongParamUnion" default="0"/>
         <attribute name="initialValue" type="anySimpleType"/>
         <attribute name="finalValue" type="anySimpleType"/>
         <attribute name="duration" type="xconnector:unsignedLongParamUnion"
default="0"/>
      </extension>
   </complexContent>
</complexType>

<simpleType name="ShowType">
   <restriction base="string">
        <enumeration value="embed" />
        <enumeration value="replace" />
        <enumeration value="new" />
   </restriction>
</simpleType>

<attributeGroup name="Cardinality">
   <attribute name="min" type="positiveInteger" default="1"/>
   <attribute name="max" type="xconnector:maxUnion" default="1"/>
</attributeGroup >
```

```
<simpleType name="unboundedString">
   <restriction base="string">
      <pattern value="unbounded"/>
   </restriction>
</simpleType>

<simpleType name="unsignedLongParamUnion">
   <union memberTypes="unsignedLong string"/>
</simpleType>

<simpleType name="maxUnion">
   <union memberTypes="positiveInteger xconnector:unboundedString"/>
</simpleType>

<simpleType name="repeatUnion">
   <union memberTypes="nonNegativeInteger xconnector:unboundedString string"/>
</simpleType>

<simpleType name="EventType">
   <restriction base="string">
        <enumeration value="presentation" />
        <enumeration value="mouseClick" />
        <enumeration value="attribution" />
        <enumeration value="mouseOver" />
        <enumeration value="prefetch" />
        <enumeration value="focus" />
   </restriction>
</simpleType>

<simpleType name="SimpleAction">
   <restriction base="string">
      <enumeration value="start" />
      <enumeration value="stop" />
      <enumeration value="pause" />
      <enumeration value="resume" />
      <enumeration value="abort" />
   </restriction>
</simpleType>

<complexType name="ConditionRole">
   <complexContent>
      <extension base="xconnector:Role">
         <sequence>
               <choice minOccurs="1" maxOccurs="1">
                 <group ref="xconnector:ConditionGroup"/>
               </choice>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

```
<complexType name="Condition" abstract="true">
   <attribute name="isNegated" type="boolean" default="false"/>
</complexType>

<complexType name="SimpleCondition" abstract="true">
   <complexContent>
      <extension base="xconnector:Condition">
      </extension>
   </complexContent>
</complexType>

<complexType name="EventStateConditionType">
   <complexContent>
      <extension base="xconnector:SimpleCondition">
         <attribute name="comparator" type="xconnector:ComparatorType"
use="required"/>
         <attribute name="state" type="xconnector:StateType" use="required"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="EventTransitionConditionType">
   <complexContent>
      <extension base="xconnector:SimpleCondition">
         <attribute name="transition" type="xconnector:TransitionType"
use="required"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="EventAttributeConditionType">
   <complexContent>
      <extension base="xconnector:SimpleCondition">
         <attribute name="comparator" type="xconnector:ComparatorType"
use="required"/>
         <attribute name="attributeType" type="xconnector:AttributeType"
use="required"/>
         <attribute name="value" type="anySimpleType" use="required"/>
         <attribute name="attributeName" type="anySimpleType"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="CompoundConditionType">
   <complexContent>
      <extension base="xconnector:Condition">
         <sequence>
               <choice minOccurs="2" maxOccurs="2">
                 <group ref="xconnector:ConditionGroup"/>
               </choice>
```

```
        </sequence>
        <attribute name="operator" type="xconnector:LogicalOperatorType"
use="required"/>
      </extension>
   </complexContent>
</complexType>

    <group name="ConditionGroup">
        <choice>
            <element ref="xconnector:eventStateCondition"/>
            <element ref="xconnector:eventTransitionCondition"/>
            <element ref="xconnector:eventAttributeCondition"/>
            <element ref="xconnector:compoundCondition"/>
        </choice>
    </group>
    <element name="eventStateCondition"
type="xconnector:EventStateConditionType"/>
    <element name="eventTransitionCondition"
type="xconnector:EventTransitionConditionType"/>
    <element name="eventAttributeCondition"
type="xconnector:EventAttributeConditionType"/>
    <element name="compoundCondition" type="xconnector:CompoundConditionType"/>

<simpleType name="LogicalOperatorType">
   <restriction base="string">
      <enumeration value="and" />
      <enumeration value="or" />
   </restriction>
</simpleType>

<simpleType name="ComparatorType">
   <restriction base="string">
      <enumeration value="eq"/>
      <enumeration value="ne"/>
      <enumeration value="gt"/>
      <enumeration value="ge"/>
      <enumeration value="lt"/>
      <enumeration value="le"/>
   </restriction>
</simpleType>

<simpleType name="StateType">
   <restriction base="string">
      <enumeration value="prepared" />
      <enumeration value="occurring" />
      <enumeration value="paused" />
      <enumeration value="finished" />
   </restriction>
</simpleType>

<simpleType name="TransitionType">
```

```
   <restriction base="string">
      <enumeration value="starts" />
      <enumeration value="stops" />
      <enumeration value="pauses" />
      <enumeration value="resumes" />
      <enumeration value="aborts" />
      <enumeration value="abortsFromPaused" />
      <enumeration value="ends" />
   </restriction>
</simpleType>

<simpleType name="AttributeType">
   <restriction base="string">
      <enumeration value="repeat" />
      <enumeration value="occurences" />
      <enumeration value="attribute" />
   </restriction>
</simpleType>

<complexType name="PropertyRole">
   <complexContent>
      <extension base="xconnector:Role">
         <sequence>
               <choice minOccurs="1" maxOccurs="1">
                 <group ref="xconnector:PropertyGroup"/>
               </choice>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<complexType name="Property" abstract="true">
</complexType>

<complexType name="EventStatePropertyType">
   <complexContent>
      <extension base="xconnector:Property">
      </extension>
   </complexContent>
</complexType>

<complexType name="EventTransitionPropertyType">
   <complexContent>
      <extension base="xconnector:Property">
         <attribute name="transition" type="xconnector:TransitionType"
use="required"/>
         <attribute name="offset" type="xconnector:unsignedLongParamUnion"/>
      </extension>
   </complexContent>
</complexType>
```

```
<complexType name="EventAttributePropertyType">
   <complexContent>
      <extension base="xconnector:Property">
        <attribute name="attributeType" type="xconnector:AttributeType"
use="required"/>
        <attribute name="offset" type="anySimpleType"/>
      </extension>
   </complexContent>
</complexType>

    <group name="PropertyGroup">
        <choice>
            <element ref="xconnector:eventStateProperty"/>
            <element ref="xconnector:eventTransitionProperty"/>
            <element ref="xconnector:eventAttributeProperty"/>
        </choice>
    </group>
    <element name="eventStateProperty" type="xconnector:EventStatePropertyType"/>
    <element name="eventTransitionProperty"
type="xconnector:EventTransitionPropertyType"/>
    <element name="eventAttributeProperty"
type="xconnector:EventAttributePropertyType"/>

<complexType name="CausalGlue">
   <sequence>
     <choice minOccurs="1" maxOccurs="1">
       <group ref="xconnector:TriggerExpressionGroup"/>
      </choice>
      <choice minOccurs="1" maxOccurs="1">
            <group ref="xconnector:ActionExpressionGroup"/>
      </choice>
   </sequence>
</complexType>

<complexType name="ConstraintGlue">
   <sequence>
        <choice minOccurs="1" maxOccurs="1">
            <group ref="xconnector:PropertyExpressionGroup"/>
        </choice>
   </sequence>
</complexType>

<complexType name="TriggerExpression" abstract="true">
   <attribute name="isNegated" type="boolean" default="false"/>
   <attribute name="minDelay" type="xconnector:unsignedLongParamUnion"
default="0"/>
   <attribute name="maxDelay" type="xconnector:unsignedLongParamUnion"
default="0"/>
</complexType>

<complexType name="SimpleTriggerExpressionType">
```

```
   <complexContent>
      <extension base="xconnector:TriggerExpression">
         <attribute name="conditionRole" type="string" use="required"/>
         <attribute name="qualifier" type="xconnector:ConditionQualifierType"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="CompoundTriggerExpressionType" >
   <complexContent>
      <extension base="xconnector:TriggerExpression">
         <choice>
            <sequence>
               <choice minOccurs="2" maxOccurs="2">
                 <group ref="xconnector:TriggerExpressionGroup"/>
               </choice>
            </sequence>
            <sequence>
               <choice minOccurs="1" maxOccurs="1">
                 <group ref="xconnector:TriggerExpressionGroup"/>
               </choice>
               <choice minOccurs="1" maxOccurs="1">
                 <group ref="xconnector:PropertyExpressionGroup"/>
               </choice>
            </sequence>
         </choice>
         <attribute name="operator" type="xconnector:LogicalOperatorType"
use="required"/>
      </extension>
   </complexContent>
</complexType>

    <group name="TriggerExpressionGroup">
        <choice>
            <element ref="xconnector:simpleTriggerExpression"/>
            <element ref="xconnector:compoundTriggerExpression"/>
        </choice>
    </group>
    <element name="simpleTriggerExpression"
type="xconnector:SimpleTriggerExpressionType"/>
    <element name="compoundTriggerExpression"
type="xconnector:CompoundTriggerExpressionType"/>

<simpleType name="ConditionQualifierType">
   <restriction base="string">
      <enumeration value="all" />
      <enumeration value="any" />
   </restriction>
</simpleType>

<complexType name="PropertyExpression" abstract="true">
```

```
</complexType>

<complexType name="SimplePropertyExpression" abstract="true">
   <complexContent>
      <extension base="xconnector:PropertyExpression">
         <attribute name="comparator" type="xconnector:ComparatorType"
use="required"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="PropertyToPropertyExpressionType">
   <complexContent>
      <extension base="xconnector:SimplePropertyExpression">
         <attribute name="firstPropertyRole" type="string" use="required"/>
         <attribute name="firstQualifier"
type="xconnector:ConditionQualifierType"/>
         <attribute name="secondPropertyRole" type="string" use="required"/>
         <attribute name="secondQualifier"
type="xconnector:ConditionQualifierType"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="AttributeToValueExpressionType">
   <complexContent>
      <extension base="xconnector:SimplePropertyExpression">
         <attribute name="propertyRole" type="string" use="required"/>
         <attribute name="qualifier" type="xconnector:ConditionQualifierType"/>
         <attribute name="value" type="anySimpleType" use="required"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="EventStateToValueExpressionType">
   <complexContent>
      <extension base="xconnector:SimplePropertyExpression">
         <attribute name="propertyRole" type="string" use="required"/>
         <attribute name="qualifier" type="xconnector:ConditionQualifierType"/>
         <attribute name="state" type="xconnector:StateType" use="required"/>
      </extension>
   </complexContent>
</complexType>

<complexType name="CompoundPropertyExpressionType" >
   <complexContent>
      <extension base="xconnector:PropertyExpression">
         <sequence>
               <choice minOccurs="2" maxOccurs="2">
                 <group ref="xconnector:PropertyExpressionGroup"/>
               </choice>
```

```
        </sequence>
        <attribute name="operator" type="xconnector:LogicalOperatorType"
use="required"/>
      </extension>
   </complexContent>
</complexType>

    <group name="PropertyExpressionGroup">
        <choice>
            <element ref="xconnector:propertyToPropertyExpression"/>
            <element ref="xconnector:attributeToValueExpression"/>
            <element ref="xconnector:eventStateToValueExpression"/>
            <element ref="xconnector:compoundPropertyExpression"/>
        </choice>
    </group>
    <element name="propertyToPropertyExpression"
type="xconnector:PropertyToPropertyExpressionType"/>
    <element name="attributeToValueExpression"
type="xconnector:AttributeToValueExpressionType"/>
    <element name="eventStateToValueExpression"
type="xconnector:EventStateToValueExpressionType"/>
    <element name="compoundPropertyExpression"
type="xconnector:CompoundPropertyExpressionType"/>

<complexType name="ActionExpression" abstract="true">
   <attribute name="delay" type="xconnector:unsignedLongParamUnion" default="0"/>
</complexType>

<complexType name="SimpleActionExpressionType">
   <complexContent>
      <extension base="xconnector:ActionExpression">
         <attribute name="actionRole" type="string" use="required"/>
         <attribute name="qualifier" type="xconnector:ActionQualifierType"/>
      </extension>
   </complexContent>
</complexType>

<simpleType name="ActionQualifierType">
   <restriction base="string">
      <enumeration value="all" />
      <enumeration value="one" />
   </restriction>
</simpleType>

<complexType name="CompoundActionExpressionType" >
   <complexContent>
      <extension base="xconnector:ActionExpression">
         <sequence>
          <choice minOccurs="2" maxOccurs="2">
             <group ref="xconnector:ActionExpressionGroup"/>
          </choice>
```

```
        </sequence>
        <attribute name="operator" type="xconnector:CompoundActionOperatorType"
use="required"/>
      </extension>
   </complexContent>
</complexType>

<simpleType name="CompoundActionOperatorType">
   <restriction base="string">
      <enumeration value="par" />
      <enumeration value="seq" />
      <enumeration value="excl" />
   </restriction>
</simpleType>

    <group name="ActionExpressionGroup">
        <choice>
            <element ref="xconnector:simpleActionExpression"/>
            <element ref="xconnector:compoundActionExpression"/>
        </choice>
    </group>
    <element name="simpleActionExpression"
type="xconnector:SimpleActionExpressionType"/>
    <element name="compoundActionExpression"
type="xconnector:CompoundActionExpressionType"/>

<element name="connectorBase">
   <complexType>
      <sequence>
         <choice minOccurs="1" maxOccurs="unbounded">
              <group ref="xconnector:xconnectorGroup"/>
         </choice>
      </sequence>
      <attribute name="id" type="ID"/>
      <attribute name="name" type="string"/>
      <attribute name="description" type="string"/>
   </complexType>
</element>

<element name="causalConnector" type="xconnector:CausalHypermediaConnector" >
</element>

<element name="constraintConnector"
type="xconnector:ConstraintHypermediaConnector" >
</element>

<group name="xconnectorGroup">
    <choice>
        <element ref="xconnector:causalConnector"/>
        <element ref="xconnector:constraintConnector"/>
    </choice>
```

```
</group>
</schema>
```

12
Apêndice D

Especificação da linguagem XTemplate 2.1.

12.1.
XT-BasicContraints.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        xmlns:BasicConstraints="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicConstraints"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicConstraints"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="constraints" type="xtemplatelang:constraintsType"
substitutionGroup="xtemplatelang:constraints"/>
    <element name="constraint" type="xtemplatelang:constraintType"
substitutionGroup="xtemplatelang:constraint"/>

</schema>
```

12.2.
XT-BasicLinking.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        xmlns:BasicLinking="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicLinking"
        targetNamespace="http://www.telemidia.puc-
```

```
rio.br/specs/xml/XTemplate/BasicLinking"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="linkBase" type="xtemplatelang:linkBaseType"
substitutionGroup="xtemplatelang:linkBase"/>
    <element name="bind" type="xtemplatelang:bindType"
substitutionGroup="xtemplatelang:bind"/>
    <element name="link" type="xtemplatelang:linkType"
substitutionGroup="xtemplatelang:link"/>
</schema>
```

12.3. XT-BasicResources.xsd

```
<schema
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        xmlns:BasicResources="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicResources"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicResources"
        elementFormDefault="qualified"
>
        <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="resources" type="xtemplatelang:resourcesType"
substitutionGroup="xtemplatelang:resources"/>
    <element name="resource" type="xtemplatelang:resourceType"
substitutionGroup="xtemplatelang:resource"/>
</schema>
```

12.4. XT-BasicVocabulary.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        xmlns:BasicVocabulary="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicVocabulary"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicVocabulary"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="vocabulary" type="xtemplatelang:vocabularyType"
substitutionGroup="xtemplatelang:vocabulary"/>
    <element name="port" type="xtemplatelang:portType"
substitutionGroup="xtemplatelang:port"/>
    <element name="component" type="xtemplatelang:componentType"
substitutionGroup="xtemplatelang:component"/>
</schema>
```

12.5. XT-connector.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- define the vocabulary element prototype -->
    <complexType name="connectorPrototype">
          <attribute name="src" type="anyURI" use="required"/>
          <attribute name="type" type="NMTOKEN" use="required"/>
          <attribute name="minOccurs" type="positiveInteger"/>
          <attribute name="maxOccurs" type="xtemplate:maxUnion"/>
    </complexType>
```

```
    <!-- define the global vocabulary elements -->
    <element name="connector" type="xtemplatelang:connectorType"
substitutionGroup="xtemplatelang:connector"/>
</schema>
```

12.6. XT-ConnectorVocabulary.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        xmlns:ConnectorVocabulary="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/ConnectorVocabulary"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/ConnectorVocabulary"
        elementFormDefault="qualified">

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="connector" type="xtemplatelang:connectorType"
substitutionGroup="xtemplatelang:connector"/>
</schema>
```

12.7. XT-constraints.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- define the constraints element prototype -->
    <complexType name="constraintsPrototype">
    </complexType>
```

```
    <complexType name="constraintPrototype">
       <attribute name="select" type="string" use="required" />
       <attribute name="description" type="string" />
    </complexType>

    <!-- define the global constraints elements -->
    <element name="constraints" type="xtemplatelang:constraintsType"
substitutionGroup="xtemplatelang:constraints"/>
    <element name="constraint" type="xtemplatelang:constraintType"
substitutionGroup="xtemplatelang:constraint"/>
</schema>
```

12.8. XTemplate21.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- include the schema files for the building block types -->
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
struct.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
vocabulary.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
constraints.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
resources.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
connector.xsd"/>
    <include schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
linking.xsd"/>

    <!-- import the NCL 2.0 language namespace -->
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- import the definitions in the modules namespaces -->
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Structure"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-Structure.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicVocabulary"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-BasicVocabulary.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicConstraints"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-BasicConstraints.xsd"/>
    <import namespace="http://www.telemidia.puc-
```

```
rio.br/specs/xml/XTemplate/BasicResources"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-BasicResources.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/ConnectorVocabulary"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-
ConnectorVocabulary.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/BasicLinking"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-BasicLinking.xsd"/>
</schema>
```

12.9. XT-language.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
           xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
              xmlns:xtemplatexslt="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/XTemplateXSLT"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        elementFormDefault="qualified">

    <!-- import the xtemplate namespaces -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemaplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/XTemplateXSLT"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-xslt.xsd"/>

    <!-- ================================================================= -->
    <!-- CoreAttrs attribute group used on all xtemplate profile elements  -->
    <!-- ================================================================= -->
    <attributeGroup name="CoreAttrs">
        <attribute name="id" type="ID" />
    </attributeGroup>

    <!-- ================================================================= -->
    <!-- Structure Functionality                                           -->
    <!-- ================================================================= -->

    <!-- ================================================================= -->
    <!-- define the top down structure of an xtemplate language document.  -->
    <!-- ================================================================= -->

    <!-- top level xtemplate element and content model -->
    <element name="xtemplate" type="xtemplatelang:xtemplateType"/>
    <complexType name="xtemplateType">
        <complexContent>
```

```
            <extension base="xtemplate:xtemplatePrototype">
                <sequence>
                    <element ref="xtemplatelang:head" minOccurs="0"
maxOccurs="1"/>
                    <element ref="xtemplatelang:body" minOccurs="1"
maxOccurs="1"/>
                </sequence>
                <attributeGroup ref="xtemplatelang:CoreAttrs"/>
                         <attribute name="name" type="string"/>
                         <attribute name="description" type="string"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- head element and content model -->
    <element name="head" type="xtemplatelang:headType"/>
    <complexType name="headType">
        <complexContent>
            <extension base="xtemplate:headPrototype">
               <choice minOccurs="0" maxOccurs="unbounded">
                  <group ref="xtemplatelang:vocabularyGroup"/>
                  <group ref="xtemplatelang:constraintsGroup"/>
                  <group ref="xtemplatelang:resourcesGroup"/>
               </choice>
            </extension>
        </complexContent>
    </complexType>

    <!-- body element and content model -->
    <element name="body" type="xtemplatelang:bodyType"/>
    <complexType name="bodyType">
        <complexContent>
            <extension base="xtemplate:bodyPrototype">
                    <sequence>
                        <choice minOccurs="0" maxOccurs="unbounded">
                             <group ref="xtemplatelang:linkBaseGroup"/>
                             <group ref="xtemplatelang:stylesheetGroup"/>
                        </choice>
                    </sequence>
            </extension>
        </complexContent>
    </complexType>

    <group name="stylesheetGroup">
        <choice>
            <element ref="xtemplatexslt:stylesheet"/>
        </choice>
    </group>

    <!-- ================================================================= -->
    <!-- linkBase Functionality                                            -->
```

```
<!-- ================================================================== -->

<!-- linkBase element and content model -->
<element name="linkBase" type="xtemplatelang:linkBaseType"/>
<complexType name="linkBaseType">
    <complexContent>
        <extension base="xtemplate:linkBasePrototype">
                    <sequence>
            <choice minOccurs="0" maxOccurs="unbounded">
                <group ref="xtemplatelang:linkGroup"/>
                <group ref="xtemplatelang:stylesheetGroup"/>
            </choice>
                    </sequence>
        </extension>
    </complexContent>
</complexType>

<group name="linkBaseGroup">
    <choice>
        <element ref="xtemplatelang:linkBase"/>
    </choice>
</group>

<!-- link element and content model -->
<element name="link" type="xtemplatelang:linkType"/>

<complexType name="linkType">
    <complexContent>
        <extension base="xtemplate:linkPrototype">
            <choice minOccurs="0" maxOccurs="unbounded">
                <group ref="xtemplatelang:bindGroup"/>
            </choice>
            <attributeGroup ref="xtemplatelang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- link groups -->
<group name="linkGroup">
    <choice>
        <element ref="xtemplatelang:link"/>
    </choice>
</group>

<!-- bind element and content model -->
<element name="bind" type="xtemplatelang:bindType"/>
<complexType name="bindType">
    <complexContent>
       <extension base="xtemplate:bindPrototype">
            <attributeGroup ref="xtemplatelang:CoreAttrs"/>
    </extension>
```

```
            </complexContent>
    </complexType>

    <!-- bind groups -->
    <group name="bindGroup">
        <choice>
            <element ref="xtemplatelang:bind"/>
        </choice>
    </group>

    <!-- ================================================================= -->
    <!-- vocabulary Functionality                                          -->
    <!-- ================================================================= -->

    <!-- vocabulary element and content model -->
    <element name="vocabulary" type="xtemplatelang:vocabularyType"/>
    <complexType name="vocabularyType">
        <complexContent>
            <extension base="xtemplate:vocabularyPrototype">
                <choice maxOccurs="unbounded">
                    <group ref="xtemplatelang:componentGroup"/>
                    <group ref="xtemplatelang:connectorGroup"/>
                </choice>
            </extension>
        </complexContent>
    </complexType>

    <group name="vocabularyGroup">
        <choice>
            <element ref="xtemplatelang:vocabulary"/>
        </choice>
    </group>

    <!-- connector element and content model -->
    <element name="connector" type="xtemplatelang:connectorType"/>

    <complexType name="connectorType">
        <complexContent>
            <extension base="xtemplate:connectorPrototype">
                <attributeGroup ref="xtemplatelang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- connector groups -->
    <group name="connectorGroup">
        <choice>
            <element ref="xtemplatelang:connector"/>
        </choice>
    </group>
```

```
<!-- component element and content model -->
<element name="component" type="xtemplatelang:componentType"/>

<complexType name="componentType">
    <complexContent>
        <extension base="xtemplate:componentPrototype">
            <choice minOccurs="0" maxOccurs="unbounded">
                <group ref="xtemplatelang:componentGroup"/>
                <group ref="xtemplatelang:portGroup"/>
            </choice>
            <attributeGroup ref="xtemplatelang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- component groups -->
<group name="componentGroup">
    <choice>
        <element ref="xtemplatelang:component"/>
    </choice>
</group>

<!-- port element and content model -->
<element name="port" type="xtemplatelang:portType"/>
<complexType name="portType">
    <complexContent>
       <extension base="xtemplate:portPrototype">
            <attributeGroup ref="xtemplatelang:CoreAttrs"/>
        </extension>
   </complexContent>
</complexType>

<!-- port groups -->
<group name="portGroup">
    <choice>
        <element ref="xtemplatelang:port"/>
    </choice>
</group>

<!-- ==================================================================== -->
<!-- constraints Functionality                                            -->
<!-- ==================================================================== -->

<!-- constraints element and content model -->
<element name="constraints" type="xtemplatelang:constraintsType"/>
<complexType name="constraintsType">
    <complexContent>
        <extension base="xtemplate:constraintsPrototype">
            <choice maxOccurs="unbounded">
                <group ref="xtemplatelang:constraintGroup"/>
            </choice>
```

```
        </extension>
    </complexContent>
</complexType>

<group name="constraintsGroup">
    <choice>
        <element ref="xtemplatelang:constraints"/>
    </choice>
</group>

<!-- constraint element and content model -->
<element name="constraint" type="xtemplatelang:constraintType"/>

<complexType name="constraintType">
    <complexContent>
        <extension base="xtemplate:constraintPrototype">
           <attributeGroup ref="xtemplatelang:CoreAttrs"/>
        </extension>
    </complexContent>
</complexType>

<!-- constraint groups -->
<group name="constraintGroup">
    <choice>
        <element ref="xtemplatelang:constraint"/>
    </choice>
</group>

<!-- ================================================================== -->
<!-- Resources Functionality                                            -->
<!-- ================================================================== -->

<!-- resources element and content model -->
<element name="resources" type="xtemplatelang:resourcesType"/>
<complexType name="resourcesType">
    <complexContent>
        <extension base="xtemplate:resourcesPrototype">
            <choice maxOccurs="unbounded">
                <group ref="xtemplatelang:resourceGroup"/>
            </choice>
        </extension>
    </complexContent>
</complexType>

<group name="resourcesGroup">
    <choice>
        <element ref="xtemplatelang:resources"/>
    </choice>
</group>

<!-- resource element and content model -->
```

```
    <element name="resource" type="xtemplatelang:resourceType"/>

    <complexType name="resourceType">
        <complexContent>
            <extension base="xtemplate:resourcePrototype">
                <attributeGroup ref="xtemplatelang:CoreAttrs"/>
            </extension>
        </complexContent>
    </complexType>

    <!-- resource groups -->
    <group name="resourceGroup">
        <choice>
            <element ref="xtemplatelang:resource"/>
        </choice>
    </group>
</schema>
```

12.10. XT-linking.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- define thlinkBasery element prototype -->
    <complexType name="linkBasePrototype">
    </complexType>

    <!-- define the link element prototype -->
    <complexType name="linkPrototype">
               <attribute name="type" type="NMTOKEN" use="required"/>
    </complexType>

    <!-- define the bind element prototype -->
    <complexType name="bindPrototype">
        <attribute name="role" type="string" use="required"/>
        <attribute name="select" type="string" use="required"/>
    </complexType>

    <!-- define the global linkBase elements -->
    <element name="linkBase" type="xtemplatelang:linkBaseType"
substitutionGroup="xtemplatelang:linkBase"/>
```

```
    <element name="bind" type="xtemplatelang:bindType"
substitutionGroup="xtemplatelang:bind"/>
    <element name="link" type="xtemplatelang:linkType"
substitutionGroup="xtemplatelang:link"/>
</schema>
```

12.11. XT-resources.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- define the constraints element prototype -->
    <complexType name="resourcesPrototype">
    </complexType>

    <complexType name="resourcePrototype">
          <attribute name="src" type="anyURI" use="required"/>
          <attribute name="type" type="string" use="required"/>
          <attribute name="label" type="string" use="required"/>
    </complexType>

    <!-- define the global resources elements -->
    <element name="resources" type="xtemplatelang:resourcesType"
substitutionGroup="xtemplatelang:resources"/>
    <element name="resource" type="xtemplatelang:resourceType"
substitutionGroup="xtemplatelang:resource"/>
</schema>
```

12.12. XT-struct.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>
```

```
    <!-- define the structure module attribute group -->
    <attributeGroup name="structureModuleAttrs">
    </attributeGroup>

    <!-- define the xtemplate element prototype -->
    <complexType name="xtemplatePrototype">
    </complexType>

    <!-- define the head element prototype -->
    <complexType name="headPrototype">
    </complexType>

    <!-- define the body element prototype -->
    <complexType name="bodyPrototype">
    </complexType>

    <!-- declare global elements -->
    <element name="xtemplate" type="xtemplatelang:xtemplateType"
substitutionGroup="xtemplatelang:xtemplate"/>
    <element name="head" type="xtemplatelang:headType"
substitutionGroup="xtemplatelang:head"/>
    <element name="body" type="xtemplatelang:bodyType"
substitutionGroup="xtemplatelang:body"/>
</schema>
```

12.13. XT-Structure.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        xmlns:Structure="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Structure"
        targetNamespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Structure"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <!-- import the definitions in the ncl namespace -->
    <import namespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XTemplate21.xsd"/>
    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- declare global elements in this module -->
    <element name="xtemplate" type="xtemplatelang:xtemplateType"
substitutionGroup="xtemplatelang:xtemplate"/>
    <element name="head" type="xtemplatelang:headType"
```

```
substitutionGroup="xtemplatelang:head"/>
    <element name="body" type="xtemplatelang:bodyType"
substitutionGroup="xtemplatelang:body"/>
</schema>
```

12.14. XT-vocabulary.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xtemplate="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        xmlns:xtemplatelang="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
        targetNamespace="http://www.telemidia.puc-rio.br/specs/xml/XTemplate/"
        elementFormDefault="qualified" attributeFormDefault="unqualified" >

    <import namespace="http://www.telemidia.puc-
rio.br/specs/xml/XTemplate/Language"
schemaLocation="file:../mediaContent/data/schemas/ncl21/XT-language.xsd"/>

    <!-- define the vocabulary element prototype -->
    <complexType name="vocabularyPrototype">
    </complexType>

    <complexType name="componentPortPrototype">
               <attribute name="type" type="NMTOKEN" use="required"/>
               <attribute name="ctype" type="string"/>
               <attribute name="minOccurs" type="positiveInteger"/>
               <attribute name="maxOccurs" type="xtemplate:maxUnion"/>
    </complexType>

    <!-- define the component element prototype -->
    <complexType name="componentPrototype">
        <complexContent>
            <extension base="xtemplate:componentPortPrototype">
            </extension>
        </complexContent>
    </complexType>

<simpleType name="CompTypes">
    <restriction base="string">
               <enumeration value="video" />
               <enumeration value="audio" />
               <enumeration value="text" />
               <enumeration value="img" />
               <enumeration value="animation" />
               <enumeration value="script" />
               <enumeration value="application" />
               <enumeration value="composite" />
    </restriction>
</simpleType>
```

```
<simpleType name="maxUnion">
     <union memberTypes="positiveInteger string"/>
</simpleType>

    <!-- define the global component attribute -->
    <!-- define the port element prototype -->
    <complexType name="portPrototype">
        <complexContent>
            <extension base="xtemplate:componentPortPrototype">
            </extension>
        </complexContent>
    </complexType>

    <!-- define the component attributeGroup -->
    <attributeGroup name="componentAttrs">
        <attribute name="component" type="IDREF" use="optional"/>
    </attributeGroup>

    <!-- define the global vocabulary elements -->
    <element name="vocabulary" type="xtemplatelang:vocabularyType"
substitutionGroup="xtemplatelang:vocabulary"/>
    <element name="port" type="xtemplatelang:portType"
substitutionGroup="xtemplatelang:port"/>
    <element name="component" type="xtemplatelang:componentType"
substitutionGroup="xtemplatelang:component"/>
</schema>
```