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.

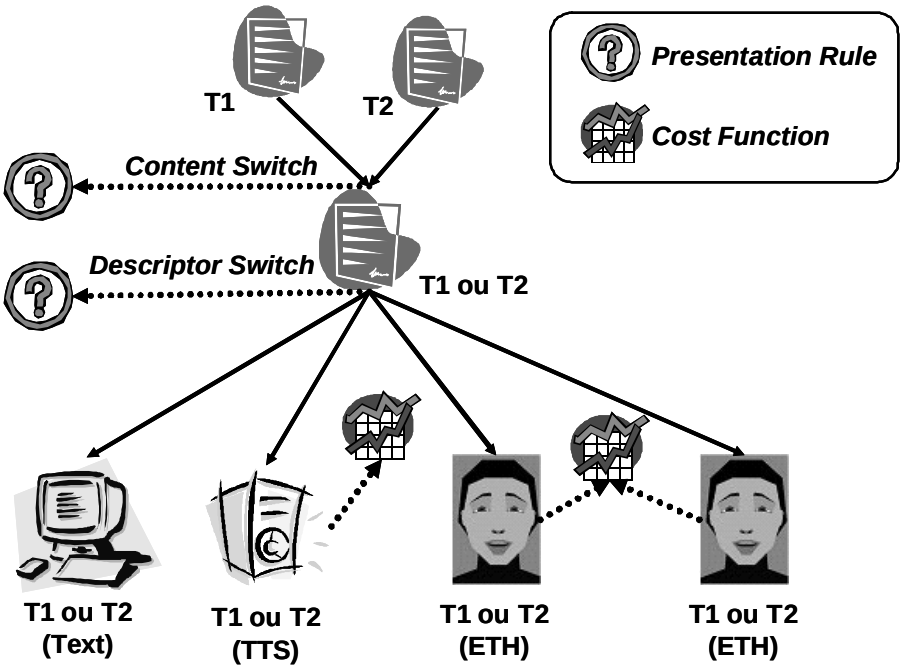

Figura 3:5 - *Switch* de conteúdo e de descritores.

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).

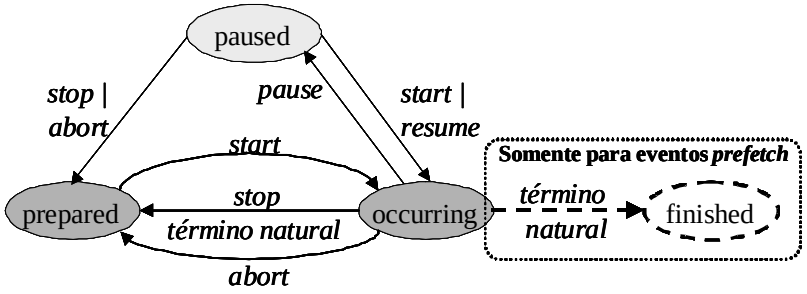

Figura 3:6 - Máquina de estados de um evento.

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*).

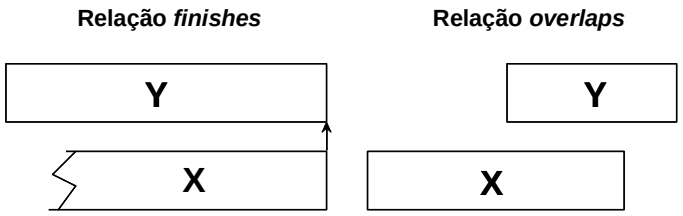

Figura 3:8 - Relações *finishes* e *overlaps*.

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.

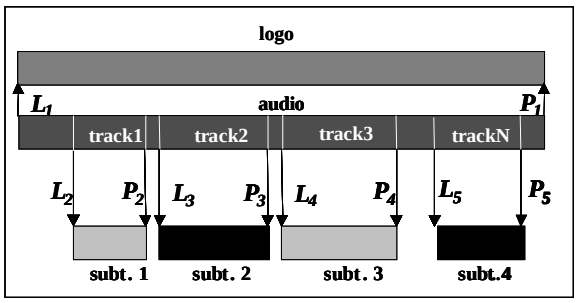

Figura 3:11 - Visão temporal de uma composição NCL.

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*.

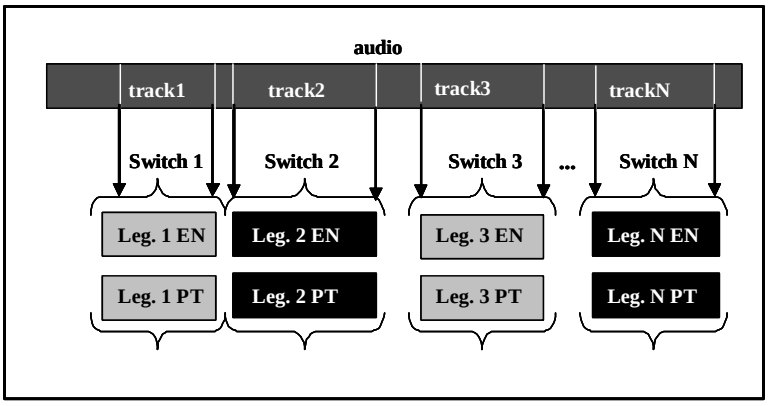

Figura 3:13 - Visão temporal de uma composição NCL.

```
<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*.