4
Objetos de mídia tridimensionais em NCL

Embutir objetos de mídia atômicos (mesmo objetos 3D como, por exemplo, um objeto representado por uma malha de polígonos) e renderizá-los nas regiões 2D de NCL, já existentes, é trivial do ponto de vista de NCL e do modelo NCM, e facilmente contemplado pela máquina de apresentação Ginga-NCL. Para isso, é suficiente adicionar um exibidor responsável por apresentar esse objeto. O mesmo se pode afirmar quando o objetivo é embutir objetos de mídia 3D imperativos[4] e renderizá-los nas regiões já suportadas por NCL. Nesse último caso, basta que o objeto 3D *imperativo* utilize uma API similar àquela de objetos NCLua para a notificação de eventos entre os objetos imperativos e o formatador NCL.

Por outro lado, quando estamos interessados em embutir objetos de mídia 3D *declarativos* em apresentações NCL, isso não é uma tarefa direta. Os objetos 3D declarativos definem cenas 3D que geralmente são compostas por vários objetos geométricos, suas características de apresentação (cor, textura, material, entre outros) e, possivelmente, sua evolução no tempo (seu *comportamento*). Dessa forma, para embuti-los em apresentações NCL, em primeiro lugar, devem-se especificar quais são as *âncoras de conteúdo* e *de propriedade* que o documento NCL pode definir nesses objetos. Adicionalmente, o exibidor desses objetos declarativos, por si só, pode tratar e gerar eventos que a própria NCL desconhece. Como discutido na Subseção 2.3.1, a versão corrente NCL 3.0 define apenas os eventos de *apresentação*, *seleção* e *atribuição*. Em ambientes 3D, porém, também é natural o uso de eventos de colisão, visibilidade, entre outros.

Com o objetivo de possibilitar à NCL embutir objetos 3D *declarativos* e renderizá-los em regiões 2D já existentes, neste capítulo são propostas algumas extensões para a linguagem. Em especial, os objetos 3D declarativos utilizados

[4] Os objetos 3D imperativos são aqueles onde a especificação do conteúdo do objeto é feito por meio de uma linguagem de programação imperativa, por exemplo, Java, C, C++ etc. Frequentemente esta especificação utiliza-se de uma API específica para desenhos tridimensionais, como é o caso de Java3D (SELMAN, 2002), OpenSceneGraph (OSG COMMUNITY, 2010) e OpenGL (SHREINER, WOO, *et al.*, 2005).

como caso de uso neste trabalho serão aqueles que definem *grafos de cena*. Com esse objetivo, será feita uma análise visando:

(i) Definir âncoras de conteúdo e âncoras de propriedade em objetos 3D declarativos;

(ii) Propor novos tipos de eventos para NCL, específicos para ambientes 3D, relacionados às âncoras definidas sobre objetos declarativos 3D;

(iii) Relacionar os eventos definidos pelas âncoras dos objetos de mídia 3D declarativos e os eventos definidos pelos objetos de mídia 2D de NCL, incluindo os eventos definidos no item anterior;

(iv) Controlar o comportamento de um objeto 3D declarativo por meio de elos e conectores NCL**.** Em objetos declarativos baseado em grafo de cena, tal abordagem pode substituir o uso de grafos de rotas, discutidos na Seção 2.2; e

(v) Relacionar objetos de mídia 3D declarativos independentes. Por exemplo, relacionar dois grafos de cenas independentes;

Com as extensões propostas, será possível embutir cenas 3D definidas por grafos de cena em uma apresentação NCL. Mais ainda, será possível sincronizá-las temporal e espacialmente a outros objetos de mídia, permitir a adaptação de seus conteúdos, e possibilitar suas exibições em múltiplos dispositivos secundários.

Adicionalmente, e talvez a maior contribuição desta dissertação, as extensões propostas possibilitam ao autor especificar o *comportamento* de uma cena 3D em uma linguagem hipermídia de alto nível e mais expressiva que o grafo de rotas, mantendo as otimizações possibilitadas pelos grafos de cena. Diferente, por exemplo, de XMT-Ω que, como discutido na Subseção 3.1.2, entrelaça o comportamento com o conteúdo da cena e impossibilita o uso dessas otimizações.

O foco principal deste trabalho é ainda manter NCL como uma linguagem de cola, que permite sincronizar objetos de mídia em apresentações multimídia sem especificar os seus conteúdos. Linguagens específicas (e possivelmente mais eficientes) para cada tipo de mídia devem especificar o conteúdo individual dos objetos de mídia. A Figura 17 ilustra NCL como linguagem de cola, incluindo também objetos de mídia 3D *declarativos*, especificados em X3D e XMT.

Figura 17 NCL como linguagem de cola, incluindo objetos de mídia 3D.

Como caso de uso para a abordagem proposta, a linguagem X3D é utilizada para a descrição de grafos de cena. É importante salientar que embora todos os exemplos e a implementação aqui apresentados sejam baseados em X3D, tal proposta não se restringe a essa linguagem, sendo também válida para outras linguagens de descrição de grafos de cena.

O restante deste capítulo está dividido como segue. Na Seção 4.1, é apresentada a proposta de embutir objetos de mídia 3D compostos, representados por grafos de cena em NCL. Já a Seção 4.2 discute como NCL pode controlar o comportamento de grafos de cena. Um quadro comparativo entre grafos de cena, XMT e NCL é apresentado na Seção 4.3.

4.1.Objetos de mídia 3D declarativos como objetos de mídia em documentos NCL

Esta seção tem como objetivo discutir como NCL pode embutir objetos de mídia 3D declarativos definidos por grafos de cena (por simplificação, a partir deste ponto, sempre que forem mencionados objetos 3D, estaremos nos referindo a objetos de mídia 3D *declarativos* definidos por *grafos de cena*).

As Subseções 4.1.1 e 4.1.2 abordam como é possível definir âncoras de conteúdo e de propriedade, respectivamente, em objetos 3D. A Subseção 4.1.3 define novos eventos para NCL, específicos de ambientes 3D. Com isso, torna-se possível sincronizar eventos gerados por objetos internos ao grafo de cena, com os

eventos já existentes de um documento NCL. A Subseção 4.1.4 propõe uma nova abordagem para parametrizar eventos em NCL. Tal parametrização é útil tanto para os novos eventos 3D, como também para os eventos 2D. Por fim, a Subseção 4.1.5 apresenta exemplos de como é possível sincronizar eventos em âncoras de objetos 3D e eventos em objetos 2D.

4.1.1.Definição de Âncoras de Conteúdo em Grafos de Cena

Na proposta desta dissertação, um objeto de mídia definido por um grafo de cena é apenas mais um objeto de mídia NCL, onde a definição de âncoras de conteúdo também deve ser possível. Por exemplo, uma âncora temporal pode ser definida como uma parte temporal (duração) da exibição do grafo de cena, definida por meio de um tempo de início e fim relativos. O mapeamento do que uma âncora de conteúdo significa em um determinado objeto de mídia deve ser responsabilidade do exibidor desse objeto (ou como é apresentado na Subseção 6.1, também pode ser responsabilidade do adaptador desse exibidor).

De uma forma geral, NCL permite definir âncoras de conteúdo por meio do elemento <area>, filho do elemento <media>. Para objetos definidos por meio de código imperativo ou declarativo, que é o caso dos grafos de cena, o elemento <area> pode identificar uma âncora de conteúdo também através dos atributos *label* e *clip*.

O atributo *label* permite que o autor defina uma âncora de conteúdo baseada em um nome que o exibidor daquele objeto deve ser capaz de identificar. Se o objeto é composto, por exemplo, o valor do atributo *label* pode identificar um objeto individual que compõe o objeto composto. Em especial, o atributo *label* pode ser utilizado para *identificar um subgrafo do grafo de cena* de forma unívoca, ou seja, defini-lo como uma âncora de conteúdo em um grafo de cena.

Dito isso, é possível observar que, nesta proposta, só é possível criar âncoras de conteúdo naqueles objetos que têm um identificador único no grafo de cena. Em X3D, caso de uso desta dissertação, o atributo DEF permite identificar um nó do grafo de forma unívoca. Assim, ao definir o atributo DEF em um nó interno ao X3D, automaticamente o autor também está informando que será possível criar âncoras de conteúdo nesses objetos. Isso é equivalente a externalizar este nó para que possa ser usado em relacionamentos do documento NCL.

A Figura 18, em (a) apresenta um exemplo de grafo de cena definido em X3D. Em (b), é apresentado como, a partir de NCL, é possível identificar âncoras de conteúdo no grafo de cena. Para isso, basta que o atributo *label* da âncora no nó de mídia tenha o mesmo valor do identificador daquele objeto. Nesse exemplo, o objeto do grafo de cena tem o identificador *mySphere*. Uma âncora de conteúdo (*mySphereAnchor*) é definida no código NCL com esse mesmo *label*.

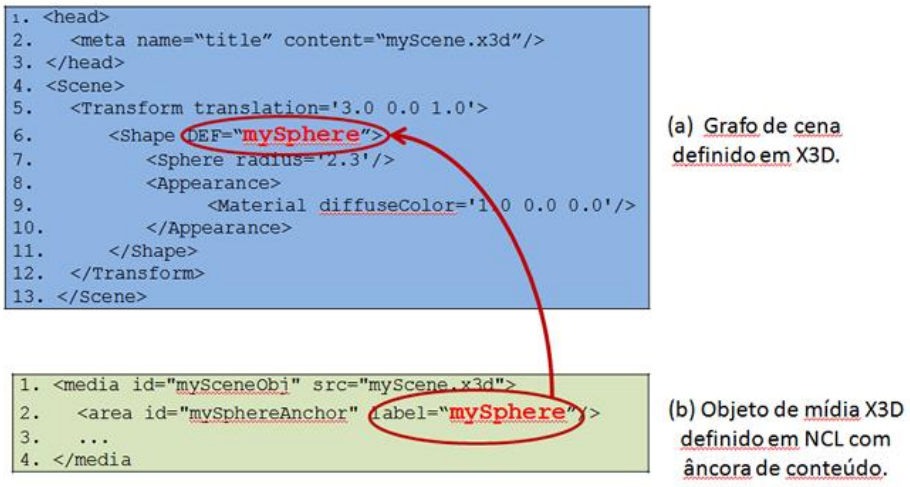

Figura 18 Definição de âncora de conteúdo em objetos de mídia definidos por grafo de cena, por meio do atributo *label*.

Um caso especial do apresentado acima é a utilização de *âncoras de conteúdo que apontem para sensores* (os nós de Sensores são discutidos na Subseção 2.2), isto é, se o grafo de cena em questão possuir nós de sensores. Nesse caso, quando os nós de sensores são ativados, eles devem notificar o início da apresentação da âncora (*eventType=presentation*, *transition=start*) de conteúdo e, quando forem desativados, devem notificar o final de sua apresentação (*eventType=presentation, transition=stop*).

Ao definir os novos eventos para NCL, na subseção seguinte, contudo, a própria utilização de nós de sensores torna-se desnecessária. A Subseção 4.3 discute mais sobre a real utilidade dos nós de sensores. Os eventos gerados por sensores também podem possuir parâmetros. A Subseção 4.1.4 discute como NCL pode especificar esses parâmetros.

Outra forma de definir âncoras de conteúdo em objetos declarativos por meio de NCL é através do atributo *clip*. O atributo *clip* é definido por uma tupla "(*chanId*, *beginOffset*, *endOffset*)". O parâmetro *chainId* identifica uma *cadeia temporal* em um objeto declarativo. Uma cadeia temporal corresponde a uma

sequência de eventos, iniciados pelo evento que corresponde ao início da apresentação do objeto declarativo (SOARES, MORENO, *et al.*, 2010).

Uma forma simples de definir uma cadeia temporal em um grafo de cena é como a duração da exibição de um subgrafo desse grafo de cena. O exemplo a seguir, Figura 19, demonstra como definir uma âncora temporal que dure de 2 a 10 segundos a partir do objeto *mySphere* definido no grafo de cena, por meio do atributo *clip*.

```
1: <media id="mySceneObj" src="myScene.x3d">
2:   <area id="mySphereClipAnchor" clip="(mySphere,2,10)"/>
3: </media>
```

Figura 19 Definição de âncora temporal em NCL por meio do atributo *clip*.

Adicionalmente, também é possível omitir algum desses parâmetros. Caso o parâmetro *chainId* seja omitido, por exemplo, o comportamento padrão é que a âncora faça referência ao nó raiz do grafo de cena. O exemplo da Figura 20 apresenta um exemplo que define uma âncora que dure de 2s a 10s, enquanto aquele grafo de cena estiver sendo executado, conforme o exemplo a seguir.

```
1: <media id="mySceneObj" src="myScene.x3d">
2:   <area id="myTemporalAnchor" clip="(,2,10)"/>
3: </media>
```

Figura 20 Definição de uma âncora temporal no grafo de cena.

4.1.2.Definição de Âncoras de Propriedades em Grafos de Cena

Conforme definido na Subseção 2.3.1, âncoras de *propriedades* (ou simplesmente propriedades) representam propriedades ou conjunto de propriedades dos nós de mídia. São alguns exemplos de propriedades de objetos de mídia: *transparency*, *zIndex*, *fontColor*, *visible*, *bounds*, entre outas. Em NCL, as propriedades são definidas pelo elemento <property>, filho do elemento <media> e possuem os atributos *name* e *value*.

Ao embutir objetos de mídia declarativos, NCL também permite que as propriedades definidas para esses objetos sejam mapeadas para suas variáveis internas (SOARES e BARBOSA, 2009). Por exemplo, ao embutir outros documentos NCL como objeto de mídia é possível definir propriedades nesses objetos que mapeiam para propriedades de objetos internos ao documento NCL embutido. Para isso, basta que a propriedade definida no elemento <media> tenha o mesmo nome de uma das portas (interfaces) do documento NCL embutido.

A Figura 21 mostra um exemplo de um documento NCL (a) que embute outro documento NCL (b), como objeto de mídia. Ao definir a propriedade *propExterna* em (a), é realizado um mapeamento para a porta do documento (b) que tem esse mesmo nome. Tal porta, por sua vez, também pode fazer referência a uma propriedade interna de um objeto documento (b), como é mostrado no exemplo. Resumindo, nesse exemplo, pode-se afirmar que ao alterar o valor de *propExterna* em (a) também estamos alterando o valor de *propInterna* em (b).

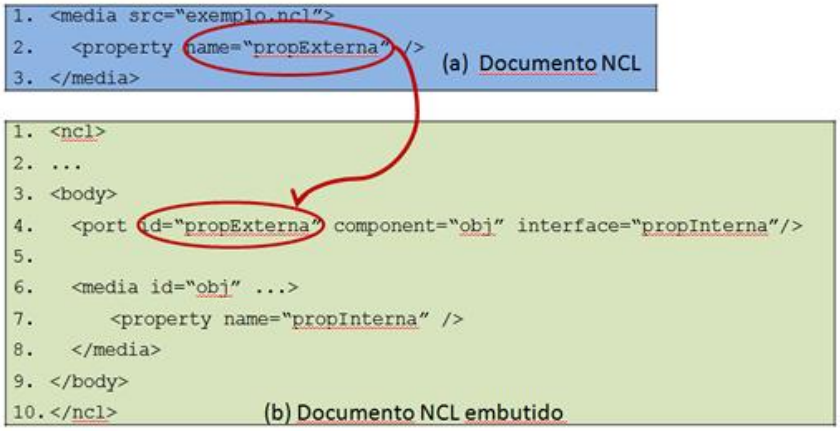

Figura 21 Mapeamento de âncoras de propriedade ao embutir objetos de mídia NCL.

Objetos definidos no modelo de grafo de cena, entretanto, não possuem portas para definir quais são as variáveis visíveis para objetos externos. Seguindo a mesma proposta da definição de âncoras de conteúdo, discutida na subseção anterior, externalizar um nó do grafo em X3D é equivalente a definir um identificador único para ele. Dessa forma, é possível referenciar a atributos internos dos nós do grafo de cena por meio de uma notação simples que identifique a qual nó estamos nos referindo e qual é o seu atributo. Neste trabalho, a seguinte notação é proposta: "IDNO#ATTR", onde IDNO é o identificador único daquele nó e ATTR é o nome do atributo daquele nó.

A Figura 22 apresenta um exemplo de definição de âncoras de propriedades que mapeiam para atributos internos de um nó em um grafo de cena X3D. A partir de agora, torna-se possível alterar os valores dessas propriedades também por meio de elos e conectores NCL.

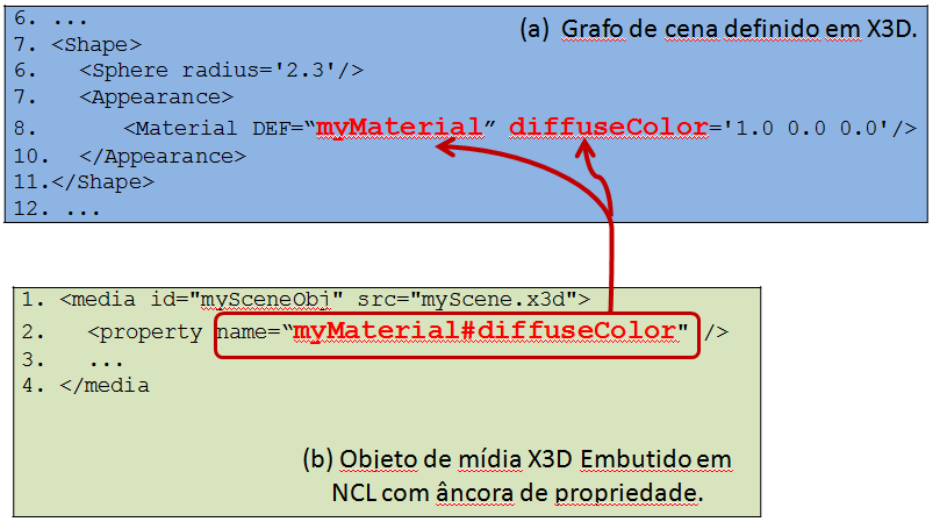

Figura 22 Definição de âncoras de propriedades em grafos de cena.

4.1.3.Eventos NCL para Ambientes 3D

Em ambientes 3D, é comum a definição de eventos que não são tratados pela versão corrente 3.0 da NCL. Definir esses eventos em NCL permite que os objetos 3D sejam não só embutidos em NCL, mas também que todos os eventos gerados por eles possam ser sincronizados com os eventos em objetos de mídia NCL já existentes. Esta seção adiciona novos eventos à NCL, visando permitir que o autor também disponha de eventos específicos de ambientes 3D para criar sua aplicação e possa sincronizar eventos em objetos 3D com outros objetos NCL.

Recapitulando o que foi apresentado na Subseção 2.3.2, tanto *relações* como *relacionamentos* são entidades de primeira classe em NCL. Ao especificar uma relação, o autor define qual é a semântica daquela relação, definindo os possíveis papéis de objetos de mídia, mas sem informar quais os objetos de mídia que são relacionados. Os relacionamentos mapeiam os papéis definidos nas relações em âncoras de objetos de mídia, criando uma instância daquela relação. Em NCL, as relações são definidas pelos *conectores*, enquanto os relacionamentos são especificados por meio dos elos (*links*).

Os conectores causais NCL (definidos pelos elementos <causalConnector>) são responsáveis por definir relações causais. Ao criar uma nova relação causal, além de nomear essa relação (atributo *id*), o autor também deve informar quais os *papéis de condição* – elementos <simpleCondition>, que podem ser agrupados em elementos <compoundCondition> – e quais os *papéis de ação* – elementos

<simpleAction>, que podem ser agrupados em elementos <compoundAction> –, bem como qual a cardinalidade de cada um desses papéis.

Ao definir um papel, o autor deve nomeá-lo (atributo *role*) e também informar: a qual evento aquele papel está relacionado (*eventType*), que em NCL 3.0 pode assumir os valores *presentation*, *attribution* e *selection*; e a qual tipo de transição na máquina de estados desse evento. Segue um exemplo:

```
1: <causalConnector id="onBeginStart">
2:   <simpleCondition role="onBegin" eventType="presentation"
                                     transition="start"/>
3:
4:   <simpleAction role="start" eventType="presentation"
                                 actionType="start"/>
5: </causalConnector>
```

Figura 23 Exemplo de um conector causal em NCL.

Para simplificar a tarefa do autor, entretanto, NCL define algumas palavras reservadas que podem ser utilizadas como nome de papéis, abstraindo a necessidade de especificar o *eventType*, *transition* ou *actionType*. Por exemplo, os papéis do conector da Figura 23, acima, podem ser reescritos apenas com o nome dos papéis (conforme a Figura 24), pois *onBegin* e *start* são palavras reservadas da linguagem.

```
1: <causalConnector id="onBeginStart">
2:   <simpleCondition role="onBegin"/>
3:   <simpleAction role="start"/>
4: </causalConnector>
```

Figura 24 Exemplo de um conector causal utilizando nome de papéis reservados em NCL.

Seguindo os mesmos critérios apresentados acima, nesta dissertação são propostos novos eventos à linguagem NCL, específicos de ambientes 3D. A Tabela 1 apresenta os eventos propostos, qual o nome do papel reservado para eles, bem como a qual transição na máquina de estado do evento esse papel está relacionada. Esses eventos estarão disponíveis para os usuários definirem condições em novos conectores NCL.

Tabela 1 Novos eventos NCL, específicos de objetos de mídia 3D.

role	eventType	transition
onBeginColllision	collision	start
onEndCollision	collision	stop
onBeginCloseness	proximity	start
onEndCloseness	proximity	stop
onBeginVisibility	visibility	start
onEndVisibility	visibility	stop

Conforme discutido na Subseção 2.3.1, para cada um dos eventos propostos acima (colisão, proximidade e visibilidade) deve ser mantida uma máquina de estado de eventos, para cada âncora do objeto que possui algum relacionamento que utiliza esse evento. Por *default*, a máquina de estados é mantida no estado de dormindo (*sleeping*).

Para o evento de colisão, quando é detectada a colisão do usuário com o objeto, a máquina de estado de colisão deve alterar seu estado para ocorrendo. Permanecendo neste estado até que o usuário deixe de colidir com o objeto, quando o evento de colisão retorna ao estado de dormindo.

Para o evento de visibilidade, se o objeto (por completo, ou apenas uma porção dele) estiver aparecendo na tela, sua máquina de estado de visibilidade deve estar no estado de ocorrendo. Quando deixar de aparecer na tela (ou porque saiu do campo de visão do usuário, ou porque sua exibição foi interrompida), a máquina de estados do evento de visibilidade desse objeto deve mudar seu estado para dormindo.

Por fim, para o evento de proximidade, a máquina de estados de evento deve permanecer no estado dormindo enquanto o usuário permanecer, no mínimo, a uma determinada distância do objeto. Tal distância deve ser um parâmetro desse evento. Uma forma de definir essa distância é por meio de um número real. Outra forma é definindo, por exemplo, as dimensões de um cubo que englobe o objeto 3D. Alternativamente, quando o evento estiver associado a um objeto 2D, a posição do usuário deve ser a posição do *mouse* e a distância deve ser um número inteiro ou as dimensões de um quadrado que englobe a região de apresentação do objeto 2D.

A Figura 25 apresenta um exemplo de um possível trajeto que um usuário pode fazer ao interagir com uma aplicação (por simplificação, o trajeto é apresentado em 2D). A máquina de estados de eventos de *proximidade* deve permanecer no estado de dormindo entre A e B, quando então deve ser alterada para o estado de ocorrendo. Entre B e C, deve permanecer no estado de ocorrendo e, somente então, deve retornar ao estado de dormindo.

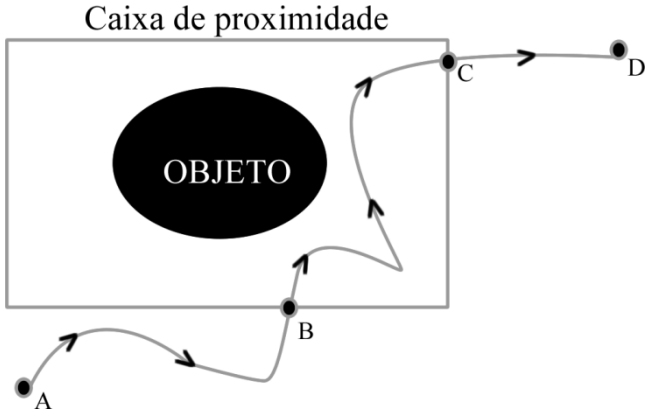

Figura 25 Possível trajeto de um usuário ao interagir com uma aplicação.

Conforme discutido na Seção 2.3.2, os conectores, em NCL, são agrupados em bases de conectores. Tais bases podem também ser disponibilizadas em documentos separados e, depois, reutilizadas por outros documentos NCL. Os usuários iniciantes na linguagem muitas vezes sequer precisam ter conhecimento de como definir um conector, apenas como utilizá-los.

A partir da definição das âncoras (de conteúdo e propriedades), discutidas nas seções anteriores, juntamente com os eventos acima propostos, já é possível sincronizar os eventos gerados por objetos 3D declarativos e os objetos de mídia 2D de NCL. Porém, existem eventos que necessitam de parâmetros, como é o caso do evento de proximidade.

Atualmente, NCL permite a definição de parâmetros de eventos, porém, de forma pouco flexível, como é discutido a seguir. A subseção a seguir apresenta uma nova proposta de definição de parâmetros de eventos. A subseção seguinte apresenta alguns exemplos de como é possível sincronizar eventos entre objetos representados por grafo de cena (em especial documentos X3D) e outros objetos de mídia NCL.

4.1.4.Parametrização de eventos NCL

Alguns eventos NCL podem ser parametrizados, como é o caso do evento de atribuição, que possui o valor a ser atribuído a uma propriedade – atributo *value* do elemento <simpleAction> – como parâmetro. Atualmente (NCL versão 3.0), entretanto, tais parâmetros são fixos e definidos por atributos do elemento <simpleAction> ou <simpleCondition>, no conector. Outra solução, proposta nesta dissertação, é possibilitar que tais parâmetros sejam arbitrários, o que

aumenta a flexibilidade e é útil, por exemplo, para os novos eventos definidos acima.

Nesta dissertação, é proposto um novo elemento, denominado <eventParam>, que pode ser filho, do elemento <simpleAction> ou do elemento <simpleCondition>, Esse elemento possui os atributos *name* e *value*, respectivamente, o nome do parâmetro do evento e o valor daquele parâmetro. A Figura 26 apresenta exemplos de parâmetros de evento tanto em ações como em condições.

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <ncl>
3:   <head>
4:     <connectorBase>
5:       <!-- ... -->
6:       <causalConnector id="onBeginClosenessSet">
7:         <simpleCondition role="onBeginCloseness">
8:           <eventParam name="size" value="100,100,100"/>
9:         </simpleCondition>
10:        <simpleAction role="set">
11:          <eventParam name="value" value="x"/>
12:        </simpleAction>
13:      </causalConnector>
14:      <!-- ... -->
15:    </connectorBase>
16:    <!-- ... -->
17:  </head>
18:  <body>
19:    <!-- ... -->
20:  </body>
21:</ncl>
```

Figura 26 Exemplo de parâmetros de eventos.

Adicionalmente, NCL também permite que os elos passem parâmetros para os conectores, visando aumentar o reúso desses últimos. Para isso, os conectores devem definir elementos <connectorParam> que informam quais os parâmetros que esse conector recebe. O elo, ao instanciar tal conector deve informar o valor para esses parâmetros por meio dos elementos <linkParam> ou <bindParam>.

O <linkParam> define o valor do parâmetro padrão para aquele elo. Caso um <bindParam> seja especificado, ele deve sobrescrever o valor do <linkParam> daquele elo e é válido somente para o papel de condição ou ação relacionado àquele <bind>. Da mesma forma, esses parâmetros de conectores também podem ser utilizados como parâmetros para eventos. A Figura 27 apresenta um exemplo de um conector parametrizado e de um elo que se utiliza desse conector.

```
22:<?xml version="1.0" encoding="ISO-8859-1"?>
23:<ncl>
24:  <head>
```

```
25:     <connectorBase>
26:       <!-- ... -->
27:       <causalConnector id=" onBeginClosenessStart">
28:         <connectorParam name="size"/>
29:         <simpleCondition role="onBeginCloseness">
30:           <conditionParam name="size" value="$size"/>
31:         </simpleCondition>
32:         <simpleAction role="start"/>
33:       </causalConnector>
34:     </connectorBase>
35:     <!-- ... -->
36:   </head>
37:   <body>
38:     <!-- ... -->
39:     <link xconnector=" onBeginClosenessStart">
40:       <bind role=" onBeginCloseness " component="scene"
                                          interface="sphere">
41:         <bindParam name="size" value="100, 100, 100"/>
42:       </bind>
43:       <bind role="start" component="video"/>
44:     </link>
45:     <!-- ... -->
46:   </body>
47:</ncl>
```

Figura 27 Exemplo de conector parametrizado e elo utilizando esse conector

4.1.5.Relacionamento entre eventos em objetos 3D e objetos de mídia NCL

Esta subseção tem o objetivo de demonstrar exemplos simples de como é possível relacionar eventos gerados por objetos 3D e objetos de mídia NCL, com o objetivo de esclarecer a proposta deste capítulo.

Exemplo 1: Evento de seleção em uma âncora de um objeto 3D.

Neste exemplo, o grafo de cena é composto apenas por uma esfera (*ball*). O documento NCL deve embutir esse grafo de cena como um objeto de mídia e definir uma âncora de conteúdo para essa esfera, informando que esse será o primeiro objeto de mídia a ser apresentado. Adicionalmente, o documento NCL também define outro objeto de mídia (*video*), que deve ser iniciado quando a esfera (*aBall*) for selecionada (condição *onSelection*).

A Figura 28 apresenta a visão estrutural deste exemplo. A visão estrutural é uma abstração gráfica que evidencia a estrutura do documento NCL (nós de contexto e nós de mídia) bem como os elos entre nós e será utilizada nos exemplos aqui apresentados para dar uma visão geral do documento.

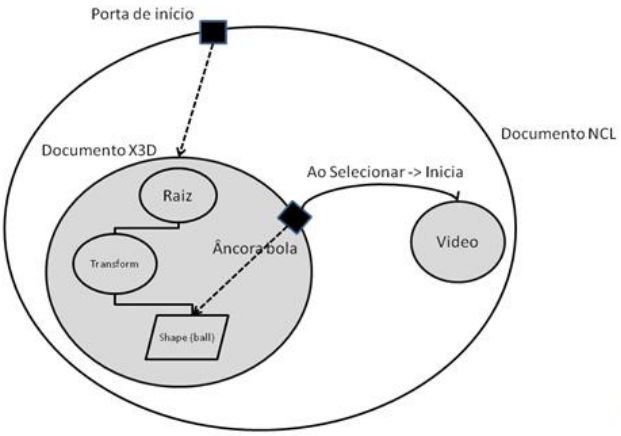

Figura 28 Visão estrutural do Exemplo 1.

A Figura 29 apresenta o código X3D do objeto de mídia 3D, enquanto a Figura 30 apresenta o código NCL deste exemplo.

```
1: <X3D>
2:   <Scene>
3:     <Transform translation="3.0 0.0 1.0">
4:       <Shape DEF="ball">
5:         <Sphere radius="2.3"/>
6:         <Appearance>
7:           <Material diffuseColor="1.0 0.0 0.0"/>
8:         </Appearance>
9:       </Shape>
10:    </Transform>
11:  </Scene>
12:</X3D>
```

Figura 29 Código fonte do objeto de mídia X3D do Exemplo 1.

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <ncl id="examplo1">
3:   <head>
4:     <regionBase>
5:       <region id="rgScene" ... />
6:       <region id="rgVideo" ... />
7:     </regionBase>
8:     <descriptorBase>
9:       <descriptor id="dsScene" region="rgScene"/>
10:      <descriptor id="dsVideo" region="rgVideo"/>
11:    </descriptorBase>
12:    <connectorBase>
13:      <importBase alias="conn" src="connBase.ncl"/>
14:    </connectorBase>
15:  </head>
16:  <body>
17:    <port id="pInicio" component="scene"/>
18:    <media id="scene" src="sphere.x3d" descriptor="dsScene">
19:      <area id="aBall" label="ball"/>
20:    </media>
21:    <media id="video" src="video.mp4" descriptor="dsVideo"/>
22:    <link xconnector="conn#onSelectionStart">
23:      <bind role="onSelection" component="scene"
   interface="aBall"/>
24:      <bind role="start" component="video"/>
```

```
25:     </link>
26:   </body>
27:</ncl>
```

Figura 30 Código fonte NCL do Exemplo 1.

Exemplo 2: Evento de proximidade em uma âncora de um objeto 3D.

A Figura 31 apresenta a visão estrutural deste exemplo. Ele é composto por um documento X3D, que especifica uma esfera (*ball*) e um cilindro (*cylinder*), e dois objetos de vídeo. O objetivo é, por meio de NCL, especificar que: "quando o usuário se aproximar da esfera, um determinado vídeo (*video1*) deve iniciar, parando a exibição do *video2*; e, quando o usuário se aproximar do cilindro, deve-se iniciar o *video2*, parando o *video1*".

Este exemplo simples pode ser facilmente estendido para uma aplicação de passeio virtual onde, ao navegar em um determinado ambiente, são exibidas mais informações sobre os objetos que fazem parte da cena, por meio de mídias auxiliares. As mídias a serem apresentadas são sensíveis à localização do usuário. Dessa forma, as mídias são executadas quando o usuário se aproxima de um determinado objeto na cena.

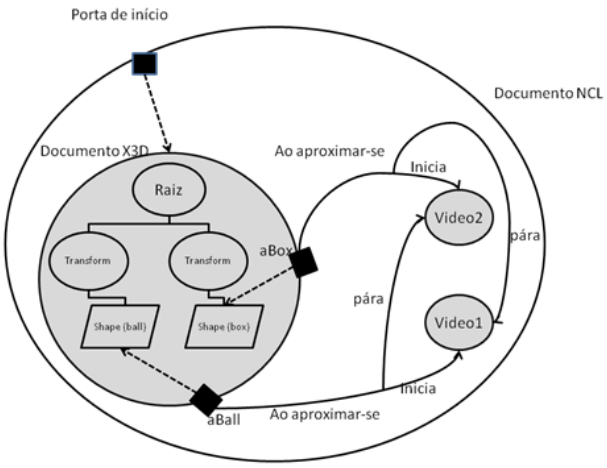

Figura 31 Visão estrutural do Exemplo 2.

A Figura 32 apresenta o código fonte do objeto X3D deste exemplo, enquanto a Figura 33 apresenta o código NCL.

```
1: <X3D profile="Interchange">
2:   <Scene>
3:     <Transform translation="3.0 0.0 1.0">
4:       <Shape DEF="ball">
5:         <Sphere radius="2.3"/>
6:         <Appearance>
7:           <Material diffuseColor="1.0 0.0 0.0"/>
8:         </Appearance>
9:       </Shape>
```

```
10:     </Transform>
11:     <Transform rotation="0.0 0.707 0.707 0.9"
                   translation =  "-2.4 0.2 1.0">
12:       <Shape DEF="cylinder">
13:         <Cylinder/>
14:         <Appearance>
15:           <Material diffuseColor="0.0 1.0 0.0"/>
16:         </Appearance>
17:       </Shape>
18:     </Transform>
19:   </Scene>
20:</X3D>
```

Figura 32 Código fonte do objeto de mídia X3D do Exemplo 2.

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <ncl>
3:   <head>
4:     <regionBase/>
5:     <descriptorBase/>
6:     <connectorBase>
7:       <importBase alias="conn" src="connBase.ncl"/>
8:     </connectorBase>
9:   </head>
10:  <body>
11:    <port id="pInicio" component="scene" ... />
12:    <media id="scene" src="scene.x3d">
13:      <area id="aBall" label="ball" ... />
14:      <area id="aCylinder" label="cylinder"/>
15:    </media>
16:    <media id="video1" src="video1.mp4" ... />
17:    <media id="video2" src="video2.mp4" ... />
18:    <link xconnector="onBeginClosenessStartStop">
19:      <bind role="onBeginCloseness" component="scene"
                                       interface="aBall">
20:        <bindParam name="size" value="2.3 2.3 2.3"/>
21:      </bind>
22:      <bind role="start" component="video1"/>
23:      <bind role="stop" component="video2"/>
24:    </link>
25:    <link xconnector="onBeginClosenessStartStop">
26:      <bind role="onBeginCloseness" component="scene"
                                       interface="aCylinder">
27:        <bindParam name="size" value="2.3 2.3 2.3"/>
28:      </bind>
29:      <bind role="start" component="video2"/>
30:      <bind role="stop" component="video1"/>
31:    </link>
32:  </body>
33:</ncl>
```

Figura 33 Código fonte NCL do Exemplo 2.

Exemplo 3: Animação em uma âncora de propriedade de um objeto 3D.

Este exemplo demonstra como criar animações em objetos internos ao grafo de cena. Para isso, é definido um documento X3D, composto apenas por um nó *Transform* e um nó Shape (que será uma esfera), filho do *Transfom*. A Figura 34 apresenta a visão estrutural do Exemplo 3.

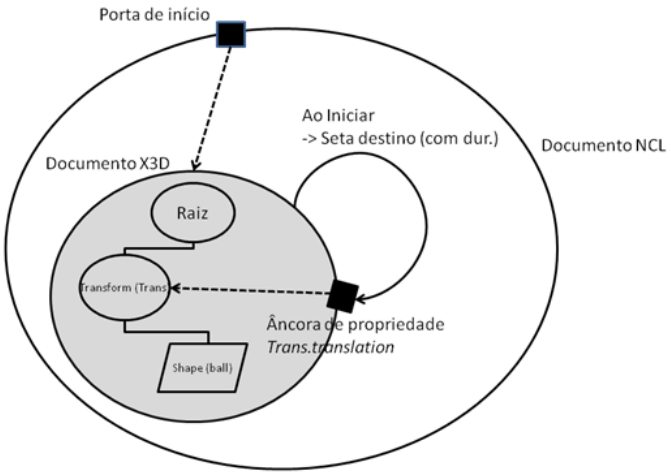

Figura 34 Visão estrutural do Exemplo 3.

Para externalizar a propriedade *translation* do nó *Transform*, é definido um identificador para esse nó. A Figura 35 apresenta o código X3D desse objeto.

```
1: <X3D profile="Interchange">
2:   <head/>
3:   <Scene>
4:     <Transform DEF="MyTransform" translation="3.0 0.0 1.0">
5:       <Shape DEF="ball">
6:         <Sphere radius="2.3"/>
7:         <Appearance>
8:           <Material diffuseColor="1.0 0.0 0.0"/>
9:         </Appearance>
10:      </Shape>
11:    </Transform>
12:  </Scene>
13:</X3D>
```

Figura 35 Código fonte do objeto de mídia X3D do Exemplo 3.

O documento NCL, ao embutir esse objeto X3D, define uma âncora de propriedade que mapeia a propriedade interna *translation* do nó *Transform*, permitindo ao documento NCL alterar o seu valor.

O elo *Animation*, na Figura 36, define que, quando o objeto X3D começar sua exibição, deve-se disparar uma ação de atribuição na propriedade *translation*, alterando a posição da esfera. Essa atribuição, entretanto, deve ocorrer durante um determinado intervalo de tempo (10s), resultando em uma animação na posição da esfera.

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <ncl>
3:   <head>
4:     <regionBase>
5:        ...
6:     </regionBase>
7:     <descriptorBase>
8:        ...
9:     </descriptorBase>
```

```
10:     <connectorBase>
11:       <importBase alias="conn" src="connBase.ncl"/>
12:     </connectorBase>
13:   </head>
14:   <body>
15:     <port id="pInicio" component="scene" .../>
16:     <media id="scene" src="scene.x3d" ... />
17:       <property name="MyTransform#translation"/>
18:     </media>
19:     <link id="Animation" xconnector="onBeginSetDur">
20:       <bind role="onBegin" component="scene"/>
21:       <bind role="set" component="scene"
    interface="MyTransform#translation">
22:         <bindParam name="value" value="2.3 2.3 2.3"/>
23:         <bindParam name="dur" value="10s"/>
24:       </bind>
25:     </link>
26:   </body>
27:</ncl>
```

Figura 36 Código fonte NCL do Exemplo 3.

Apenas para permitir uma melhor comparação com a programação baseada em grafos de rota, o mesmo exemplo acima pode também ser desenvolvido em X3D. Em X3D, entretanto, para definir animações, é necessário o uso de nós de sensores (em especial, o nó *TimeSensor*) e nós de interpolação, novos conceitos que também devem ser aprendidos pelos usuários. Nesse exemplo, onde modificamos a posição de um nó, estamos interessado no nó de interpolação denominado *PositionInterpolator*.

O nó *PositionInterpolator* faz uma interpolação linear entre posições (informadas como três valores reais, respectivamente, x, y e z) gerando eventos de saída *value_changed*, os quais podem ser usados por grafos de rotas para informar a posição de outros nós. O atributo *key* deve informar uma lista de valores reais. Para cada um dos valores informados no atributo *key* deve ser informado uma posição (três reais) no atributo *keyValue*. A Figura 37 apresenta um exemplo de como definir a mesma animação do Exemplo 3 em X3D, utilizando grafos de rotas.

```
1: <X3D profile="Interchange">
2:   <head/>
3:   <Scene>
4:     <Transform DEF="MyTransform" translation="3.0 0.0 1.0">
5:       ...
6:       <Shape DEF="ball">
7:         <Sphere radius="2.3"/>
8:         <Appearance>
9:           <Material diffuseColor="1.0 0.0 0.0"/>
10:        </Appearance>
11:      </Shape>
12:      ...
13:    </Transform>
```

```
14:     <TimeSensor DEF="TIME" cycleInterval="10" loop="false"/>
15:     <PositionInterpolator DEF="POSITION" cycleInterval="false"
   key="0 0.99" keyValue="3.0 0.0 1.0 2.3 2.3 2.3"/>
16:
17:     <ROUTE fromNode="TIME" fromField="fraction_changed"
              toNode="POSITION" toField="set_fraction" />
18:
19:     <ROUTE fromNode="POSITION" fromField="value_changed"
              toNode="MyTransform" toField="set_translation" />
20:   </Scene>
21:</X3D>
```

Figura 37 Exemplo de animação utilizando apenas X3D (grafo de rotas).

4.2.Controlando o Comportamento de Grafos de Cena por meio de NCL

Como discutido na Introdução, utilizar-se de linguagens hipermídia para descrever aplicações 3D permite que mecanismos poderosos para descrição e busca de informação sejam também utilizados em aplicações 3D. Adicionalmente, as linguagens hipermídia permitem uma descrição declarativa, e em alto nível, de relações entre os objetos que podem fazer parte de uma cena 3D.

As linguagens hipermídia com foco no sincronismo espaço-temporal, como é o caso de NCL, permitem ainda descrever sincronismos entre eventos de forma simples, sem a necessidade de linguagens imperativas (como *scripts*, por exemplo). Fora isso, NCL também permite o uso de múltiplos dispositivos e a adaptação de conteúdo baseado em informações do usuário ou da plataforma.

Uma consequência natural de se definir as âncoras de conteúdo e eventos em grafos de cena, conforme discutido na seção anterior, é permitir que os elos e conectores NCL também possam ser utilizados para controlar o comportamento de cenas 3D, – etapa (iii) discutida no início deste capítulo – em detrimento dos grafos de rotas. Os elos e conectores NCL, podem especificar todo o comportamento da cena, os eventos interativos, as animações e a evolução da cena no tempo e tirar proveito de todas as demais funcionalidades de NCL. Como os elos e conectores NCL são totalmente independentes da estrutura do grafo de cena, inclusive, definidos em outro documento, todas as otimizações para grafos de cena continuam sendo passíveis de serem implementadas.

Como exemplo, podemos recapitular a cena da Figura 15. Conforme discutido anteriormente, tal cena era composta por dois objetos (bolas), que

embora estivéssem espacialmente separadas (uma em uma sala e outra em um quarto) deveriam ser apresentadas em sequência.

Na Figura 38, em (a), pode-se observar qual seria a modelagem ideal para essa cena. Enquanto (b) apresenta como XMT consegue modelar tal cena. Como visto na Subseção 3.1.2, ao modelar essa cena em XMT, não é possível manter as otimizações para grafos de cena, pois ela embute na própria hierarquia do documento o comportamento da cena, através dos contêineres temporais. Em (c), por outro lado, vemos como NCL, ao embutir grafos de cena como objetos de mídia, permite essa modelagem, o que é bastante similar ao apresentado em (a). A Figura 39 apresenta o código NCL desse exemplo.

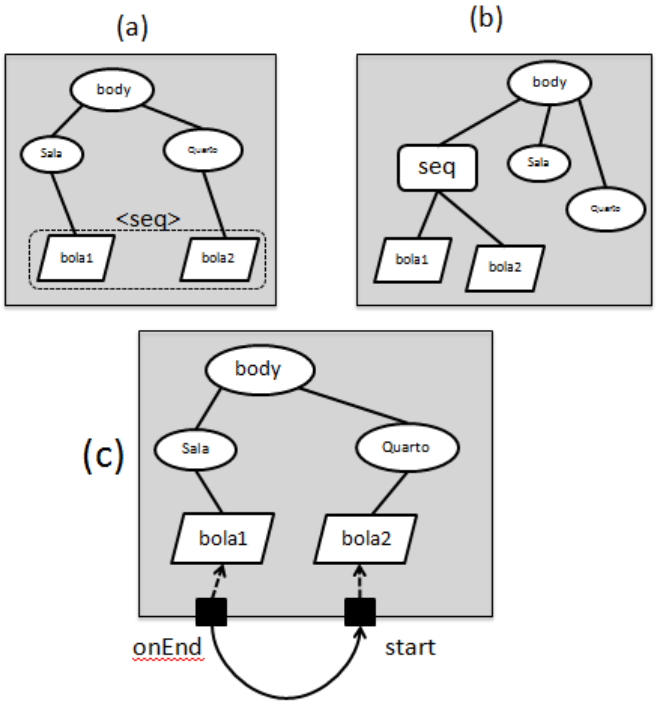

Figura 38 Exemplo de relacionamento seqüencial entre duas mídias.

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <ncl>
3:   <head>
4:     <!—Conectores NCL (possivelmente em outro arquivo) -->
5:    <connectorBase>
6:       <importBase alias="conn" src="connBase.ncl"/>
7:     </connectorBase>
8:   </head>
9:
10:  <body>
11:    <port id="p1" component="scene"/>
12:
13:    <media id="scene" ... >
14:      <area id="aBola1" label="bola1"/>
15:      <area id="aBola2" label="bola2"/>
```

```
16:     </media>
17:
18:     <!-- ELO NCL onEnd->start -->
19:     <link id="Animation" xconnector="onEndStart">
20:       <bind role="onEnd" component="scene"
                             interface="aBola1"/>
21:
22:       <bind role="start" component="scene"
                             interface="aBola2"/>

23:       </bind>
24:     </link>
25:   </body>
26:</ncl>
```

Figura 39 Exemplo de código fonte NCL que pode representar o relacionamento sequencial do exemplo da Figura 38.

Mais ainda, pelo caráter de linguagem de cola de NCL, também é possível relacionar diversos grafos de cena diferentes em uma mesma aplicação – a etapa (iv) mencionada no início deste capítulo. Até o momento, entretanto, tais grafos de cena ainda devem ser renderizados em regiões 2D distintas de NCL.

4.2.1.Elos e conectores NCL em documentos X3D

Conforme discutido anteriormente, NCL é modular. Cada módulo da linguagem agrupa um conjunto de elementos semanticamente relacionados. Outra forma possível de controlar o comportamento dos grafos de cena por meio das facilidades de NCL é incluir seus módulos responsáveis pela definição dos elos e conectores (*Linking* e *Connector*, respectivamente) em documentos que definem esses grafos de cena. Isso resulta em uma nova linguagem.

A Figura 40 apresenta o *Exemplo 3*, demonstrado na Figura 35 e Figura 36 como um objeto de mídia X3D embutido em um documento NCL, em um documento único. Esse documento é escrito em uma nova linguagem (denominada aqui de XNCL3D) que agrega as funcionalidades dos elos conectores NCL em documentos X3D.

```
27:<?xml version="1.0" encoding="ISO-8859-1"?>
28:<XNCL3D>
29:   <head>
30:     <!—Conectores NCL (possivelmente em outro arquivo) -->
31:    <connectorBase>
32:       <importBase alias="conn" src="connBase.ncl"/>
33:     </connectorBase>
34:   </head>
35:
36:   <!-- Cena X3D -->
37:   <Scene>
38:     <Transform DEF="MyTransform" translation="3.0 0.0 1.0">
```

```
39:       <Shape DEF="ball">
40:         <Sphere radius="2.3"/>
41:         <Appearance>
42:           <Material diffuseColor="1.0 0.0 0.0"/>
43:         </Appearance>
44:       </Shape>
45:     </Transform>
46:   </Scene>
47:
48:   <!-- ELOS NCL -->
49:   <link id="Animation" xconnector="onBeginSetDur">
50:     <bind role="onBegin" component="MyTransform"/>
51:     <bind role="set" component="MyTransform"
                        interface="translation">
52:       <bindParam name="value" value="2.3 2.3 2.3"/>
53:       <bindParam name="dur" value="10s"/>
54:     </bind>
55:   </link>
56:</XNCL3D>
```

Figura 40 Exemplo de documento X3D com nós e elos NCL.

Como vantagem, essa alternativa é aparentemente mais simples para o autor que deseja apenas descrever uma cena 3D e seu comportamento, do que embutir grafos de cena como objetos de mídia em documentos NCL. Isso porque os elos relacionam não mais objetos de mídia, mas diretamente os próprios nós do grafo de cena. Dessa forma, entretanto, não é possível sincronizar os eventos gerados pelo grafo de cena com outros objetos de mídia, apenas com outros nós no grafo de cena e perde-se o caráter de NCL como uma linguagem de cola.

Mesmo assim, nada impede que um documento XNCL3D também seja embutido como um *novo objeto de mídia* NCL, assim como objetos NCL podem ser embutidos em outros documentos NCL. Uma possibilidade interessante é, por exemplo, a utilização dos elementos que definem as portas de NCL em documentos XNCL3D, visando externalizar alguns de seus nós.

Essa proposta não foi explorada em detalhes por esta dissertação, porque o foco do seu trabalho não é definir uma nova linguagem, mas sim embutir objetos de mídia X3D (conforme a linguagem é hoje) em documentos NCL (com mínimas alterações). Entretanto, como é discutido no Capítulo 7, isso é visto como um importante trabalho futuro.

4.3.Comparativo entre grafo de rotas, XMT e NCL

Esta seção apresenta um quadro comparativo entre a programação do comportamento de uma cena 3D utilizando os modelos de programação e linguagens discutidos até aqui. São avaliadas as linguagens XMT, o modelo de

programação por grafo de rotas (especificamente por meio da linguagem X3D) e o modelo NCM (através da linguagem NCL). A Tabela 2 resume a discussão que segue, segundo a notação: "--", "-", "+" e "++", informando em ordem crescente o quanto a linguagem é "melhor avaliada" levando em consideração aquele critério.

Tabela 2 Tabela comparativa entre XMT, Grafo de Rotas (X3D) e o modelo NCM para o controle do comportamento de cenas 3D.

Critério de comparação	**XMT**	**Grafo de Rotas/X3D**	**NCM/NCL**
Expressividade	+	+	++
Simplicidade	++	+/-	+
Prolixidade	+	++	-
Múltiplos dispositivos	--	--	++
Falta da necessidade de nós de sensores	++	+/-	++
Separação entre conteúdo da cena e comportamento (maior possibilidade de reúso)	--	++	++
Otimizações possibilitadas pelos grafos de cenas	--	++	++
Adaptação de conteúdo (perfil do usuário ou do sistema) de forma declarativa	+	+	++
Proximidade da linguagem natural (maior abstração ou mais declarativa)	+	-	+
Suporte declarativo ao sincronismo temporal	+	--	++

A principal vantagem do uso de NCL ou grafo de rotas em comparação com XMT está no fato de que, em ambos, existe uma *separação* clara entre o *conteúdo* e o *comportamento da cena*, aumentando o reúso na aplicação, já que tanto o comportamento como o conteúdo podem ser reusados separadamente (SOARES NETO e SOARES, 2009).

De forma diferente, em XMT, tanto o conteúdo como o comportamento são entrelaçados em uma mesma hierarquia. Como consequência, o uso das otimizações para grafos de cena (discutidas na Subseção 2.1.1) também se torna inviável em XMT. Isso se deve ao fato de que, muitas vezes, pode não ser possível ao autor criar uma estrutura hierárquica levando em consideração principalmente a disposição espacial dos objetos, como é de vital importância para grafos de cena.

O uso de linguagens multimídia, tais como NCL e XMT, torna o processo de construção de aplicações de cenas multimídia 3D bem mais simples. Em especial, quando nos referimos ao sincronismo temporal dos objetos de mídia na cena. XMT, por exemplo, por meio de seus contêineres temporais (<par> e <seq>) torna bastante simples a tarefa de especificar relacionamentos simples. Definir animações, também é uma tarefa bem menos árdua utilizando NCL ou XMT, quando comparado ao grafo de rotas.

Se XMT é bastante simples por um lado, por outro, ela peca em ser pouco expressiva, quando comparada a NCL. As relações possíveis entre objetos de mídia são pré-determinadas (<par>, <seq> e <excl>), enquanto em NCL o autor está apto a criar novas relações por meio dos conectores. Para permitir a criação dessas novas relações, entretanto, NCL se torna mais prolixa do que as outras abordagens.

O fato de XMT ser pouco expressiva já foi identificado como um problema há bastante tempo na sua precursora, a linguagem SMIL. Por isso, hoje, ela também permite especificar o sincronismo temporal por meio dos atributos *begin* e *end* presente nos elementos que representam mídias. Esse artifício, entretanto, traz o inconveniente de permitir ao autor ignorar totalmente a semântica dos contêineres temporais.

O exemplo da Figura 41 mostra que, embora dois objetos de mídia estejam em uma composição <par>, o que informa que eles devem executar em paralelo, os atributos *begin* e *end* podem ignorar totalmente essa semântica, produzindo, no mínimo, um código de difícil leitura.

```
1: <par>
2: ..<audio id="audio1" src="audio.wav"/>
3: ..<video src="video.mp4" begin="video1.begin"/>
4: </par>
```

Figura 41 Exemplo de código XMT onde a semântica do contêiner temporal <par> pode ser ignorada para executar mídias em sequência.

Outro ponto a se destacar nesta comparação é a real necessidade dos *nós de sensores*. Tais nós são extremamente úteis para manter a eficiência da implementação. Contudo, pode-se perceber que não é exatamente necessário que os autores de documentos tenham conhecimento de sua existência. Por meio de um *parser* inicial no documento, baseado nos eventos que são roteados em grafos de rotas ou nos elos e conectores NCL, é possível deduzir quais os sensores que

serão necessários no momento da execução. O exibidor pode então criar esses sensores de forma totalmente automática e abstrata para o autor da aplicação.

Em XMT não existem os nós de sensores. Na proposta desta dissertação o uso dos sensores torna-se opcional já que o autor é quem decidirá se deve criar uma âncora de conteúdo que aponte para um sensor ou para qualquer outro nó no grafo de cena. Como é apresentado na Seção 6.1, caso o autor opte por não utilizar âncoras que apontem para nós de sensores, os nós de sensores são criados dinamicamente baseados nos elos e conectores presentes no documento.

Adicionalmente, NCL é a única das abordagens que prevê a utilização de múltiplos dispositivos na apresentação.