3
Trabalhos Relacionados

Este capítulo apresenta os principais trabalhos relacionados a esta dissertação. A Subseção 3.1 apresenta algumas linguagens declarativas baseadas em grafos de cena, comparando-as com a proposta deste trabalho. Em especial, são discutidas as linguagens: *eXtensible 3D Graphics* (X3D) (WEB3D CONSORTIUM, 2009a), padrão ISO para descrição e distribuição de cenas 3D na Web; e *Extensible MPEG-4 Textual Format* (XMT) (KIM, WOOD e CHEOK, Novembro de 2000), padrão MPEG-4 para a descrição de cenas 3D em sintaxe textual.

Na Subseção 3.2 é discutida a arquitetura do Ginga-NCL, subsistema declarativo dos padrões ISDB-T_B e ITU-T H.761, que oferece o suporte a execução de aplicações NCL. O objetivo é destacar os principais pontos da arquitetura que serão estendidos visando adicionar exibidores para objetos de mídia 3D.

3.1.Linguagens baseadas em Grafos de Cena

Para especificar grafos de cena (vide Seção 2.1) os autores dispõem de linguagens de programação que definem uma API para sua criação e manipulação. Neste texto, tais linguagens são denominadas “linguagens baseadas em grafos de cena”. Grande parte dessas linguagens é declarativa e baseada em XML, como é o caso de X3D e XMT, discutidas nesta seção. Porém, também existem linguagens que permitem a construção e manipulação de grafos de cena de forma imperativa, como é o caso de Java3D (SELMAN, 2002), OpenSceneGraph (OSG COMMUNITY, 2010), dentre outras.

Não está no escopo deste trabalho discutir as linguagens imperativas para a construção de grafos de cena, nem embutir objetos imperativos 3D em NCL, já que isso pode ser realizado com poucas alterações na forma como NCL, hoje, já embute objetos imperativos. Sendo assim, os trabalhos apresentados nesta seção restringem-se a linguagens baseadas em grafos de cena *declarativas*.

3.1.1.Extensible 3D (X3D) Graphics

Extensible 3D (X3D) *Graphics* é a linguagem padrão ISO/IEC (WEB3D CONSORTIUM, 2009a) para descrição e distribuição de mundos 3D interativos na Web. X3D nasceu como uma evolução natural de VRML97 (WEB3D CONSORTIUM, 1997). De uma forma geral, as principais funcionalidades suportadas por X3D são:

- Gráficos 3D: geometria poligonal, geometria paramétrica, transformações hierárquicas, iluminação, materiais e mapeamento de textura (*multipass* e *multi-stage*)
- Gráficos 2D: Texto, vetor 2D e formas planares desenhadas sobre transformações 3D hierárquicas.
- Animações: *Timers* (relógios) e interpoladores para lidar com animações contínuas.
- Interação com o usuário por meio de: mouse (seleção e "arraste"), e teclado.
- Navegação: câmeras; movimento do usuário na cena 3D; detecção de colisão, proximidade e visibilidade.
- Rede: habilidade de compor uma única cena 3D por meio de recursos localizados em outros pontos na rede; hiperelos para objetos de outras cenas ou recursos na *World Wide Web*.
- Objetos definido pelo usuário: habilidade de estender as funcionalidades do exibidor X3D por meio da criação de tipos de dados definido pelo usuário (*PROTOs*).
- Extensão por meio de scripts: é possível a alteração da cena dinamicamente por meio de linguagens de programação imperativas, de sistema (por exemplo, Java) ou de *script* (por exemplo, *ECMAScript*)
- Simulação física: animação de humanóides; dados geoespaciais; integração com protocolos de Simulação Interativa Distribuída (DIS).

Por meio de um *grafo de cena*, X3D permite a modelagem dos objetos gráficos que constituem um ambiente virtual (ou modelo). De uma forma geral, o

grafo de cena X3D é aproximadamente idêntico ao grafo de cena definido pelo VRML97, apenas novos tipos de nós são adicionados. O grafo de cena suportado pelo X3D é um DAG. A Figura 8 apresenta um exemplo de grafo de cena.

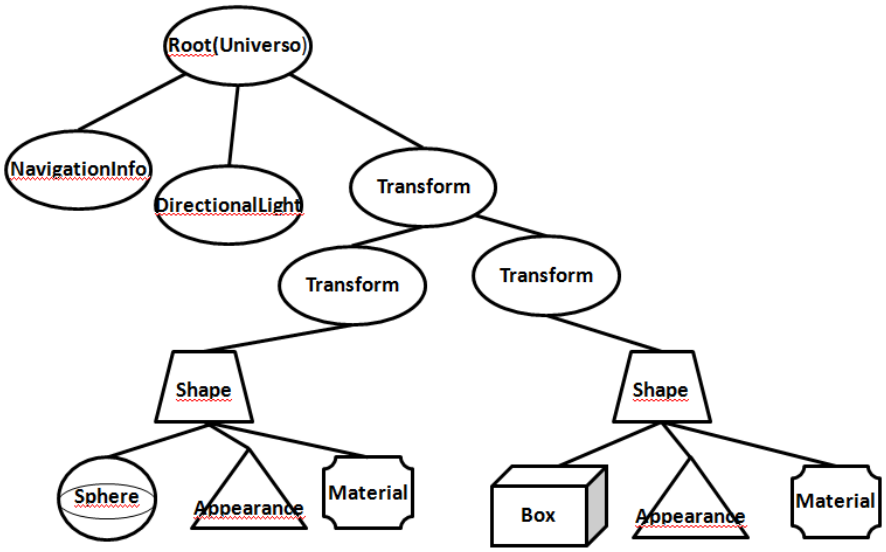

Figura 8 Exemplo de um Grafo de Cena X3D.

Cada nó no grafo de cena é uma instância de um dos tipos de nós disponibilizados pela linguagem. Seguem alguns exemplos de nós disponibilizados por X3D:

- *Shape*: define uma forma geométrica;
- *Box*, *Sphere*, *Cylinder*, *Cone etc.*: Especializam o tipo de forma geométrica do nó *Shape*.
- *Transform*: define a posição, orientação e escala de seus filhos;
- *Appearence*: determina a aparência de uma forma geométrica. Essa forma geométrica deve ser pai do nó *Appearence*;
- *Anchor*[3]: permite agrupar objetos que ao serem selecionados redirecionam o usuário para outro documento (similar a uma âncora HTML).

[3] É importante ressaltar a diferença entre âncoras em documentos X3D (elemento <Anchor>) e âncoras de conteúdo em NCL (elemento <area>). As âncoras em X3D são similares às âncoras HTML. Define-se um conjunto de objetos (uma região do texto, em HTML) que, quando o usuário seleciona um desses objetos, o usuário *vai para* outro modelo (outra página, em HTML). Sendo assim, a semântica do comportamento daquela porção de conteúdo definida pela âncora (*ao clicar*, *vai para*) já está embutida na própria definição do elemento <Anchor>. Por outro lado, NCL define uma âncora de conteúdo apenas como uma porção do conteúdo de uma mídia, a qual pode ser tanto temporal (uma parte do tempo de exibição do objeto) como espacial (uma região desse objeto). Em NCL, são os elos que definem o que deve acontecer quando uma âncora inicia sua exibição, quando ela é selecionada pelo usuário etc.

- *Switch*: Seleciona um (ou nenhum) dos nós filhos para serem renderizados. Um inteiro informa qual o filho que está ativo, sendo que o valor "-1" informa que nenhum dos seus filhos deve ser renderizado. É possível alterar o valor de qual o filho ativo manualmente (por meio de um grafo de rotas, linguagem imperativa, e também será por meio de NCL, como é proposto neste trabalho).

Para controlar o comportamento do grafo de cena, por exemplo, permitindo-se criar animação, reações a eventos de seleção, proximidade etc., X3D utiliza-se da abstração de **grafo de rotas**. Conforme discutido na Subseção 2.2, as rotas interligam eventos de entrada e eventos de saída relacionados a nós. Isso informa ao exibidor X3D que, quando o evento de entrada ocorrer, deve-se disparar o respectivo evento de saída.

Adicionalmente, X3D também permite que o conteúdo da cena seja expresso em diferentes codificações. As principais são: *Classic VRML Encoding*, *XML Encoding*, além de um formato binário visando distribuição de modelos. A Figura 9 apresenta um mesmo modelo em *Classic VRML Encoding* e *XML Encoding*. A Figura 10 apresenta o resultado da execução desse modelo.

```
#X3D v3.0 utf8
PROFILE Interchange

Group {
  children [
  NavigationInfo { type [ "EXAMINE" ] }
  DirectionalLight{}
  Transform {
    children [
    Transform {
      translation 3.0 0.0 1.0
      children [
        Shape {
          geometry Sphere { radius 2.3 }
          appearance Appearance {
            material Material {
              diffuseColor 1.0 0.0 0.0
            } } } ] }
    Transform {
      translation -2.4 0.2 1.0
      rotation 0.0 0.707 0.707 0.9
      children [
        Shape {
          geometry Box {}
          appearance Appearance {
            material Material {
              diffuseColor 0.0 0.0 1.0
            } } } ] }
    ]
  }
 ]
}
```

Classic VRML Encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Interchange' version='3.0'>
<head>
</head>
<Scene>
  <Transform>
    <NavigationInfo type='EXAMINE'/>
    <DirectionalLight/>
    <Transform translation='3.0 0.0 1.0'>
      <Shape>
        <Sphere radius='2.3'/>
        <Appearance>
          <Material diffuseColor='1.0 0.0 0.0'/>
        </Appearance>
      </Shape>
    </Transform>
    <Transform rotation='0.0 0.707 0.707 0.9'
               translation='-2.4 0.2 1.0'>
      <Shape>
        <Box/>
        <Appearance>
          <Material diffuseColor='0.0 0.0 1.0'/>
        </Appearance>
      </Shape>
    </Transform>
  </Transform>
</Scene>
</X3D>
```

XML Encoding

Figura 9 *Classic VRML* e *XML Encoding* de um mesmo modelo X3D.

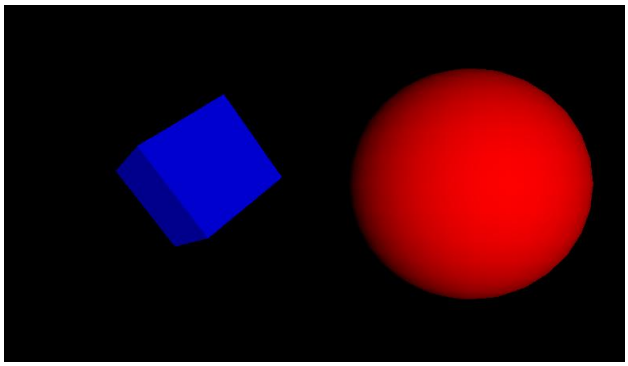

Figura 10 Modelo da Figura 9 em execução

A especificação da linguagem X3D é dividida em diversos perfis (*profiles*), evidenciados na Figura 11. Cada perfil agrupa um conjunto de funcionalidades inter-relacionadas. A existência de perfis possibilita que os desenvolvedores de exibidores X3D suportem níveis intermediários da linguagem, ajudando a estender o conjunto de funcionalidades de X3D para dispositivos com menos processamento, tais como PDAs ou telefones celulares.

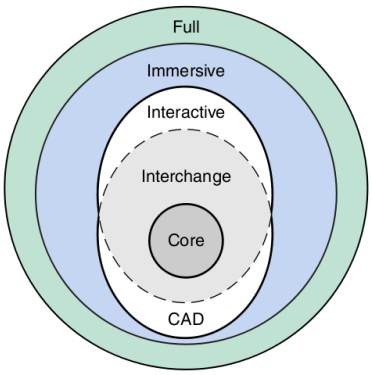

Figura 11 Perfis X3D. Fonte: (BRUTZMAN e DAILY, 2007).

Para exibir corretamente um documento X3D, faz-se necessário que exibidores (ou *players*) sejam desenvolvidos. Como X3D é uma linguagem declarativa, seus exibidores são responsáveis por ler, interpretar e executar da melhor forma possível o documento. A Figura 12 apresenta os principais componentes da arquitetura de um exibidor X3D.

Visando possibilitar o controle externo da exibição de objetos de mídia X3D, grande parte dos de seus exibidores permite que programadores construam e manipulem o grafo de cena de forma imperativa. Para isso, eles usualmente implementam duas interfaces padrões: SAI (*Scene Access Interface*) (WEB3D CONSORTIUM, 2009b) e EAI (*External Access Interface*).

SAI é uma API que permite a construção e manipulação do grafo de cena internamente (por meio de nós *scripts*) ou por meio de aplicações que embutem

exibidores X3D. EAI permite esse mesmo controle por meio de programas externos (por exemplo, através de uma conexão de rede). Ambas definem um conjunto abstrato de serviços que são independentes de linguagem. Essas APIs serão utilizadas na implementação de parte deste trabalho, ao embutir um exibidor X3D no Ginga-NCL.

Além disso, X3D também é a linguagem de descrição de grafos de cena utilizada como caso de uso por esta dissertação. Dessa forma, todos os exemplos de grafo de cena, quando necessitarem de uma apresentação textual, estarão em X3D.

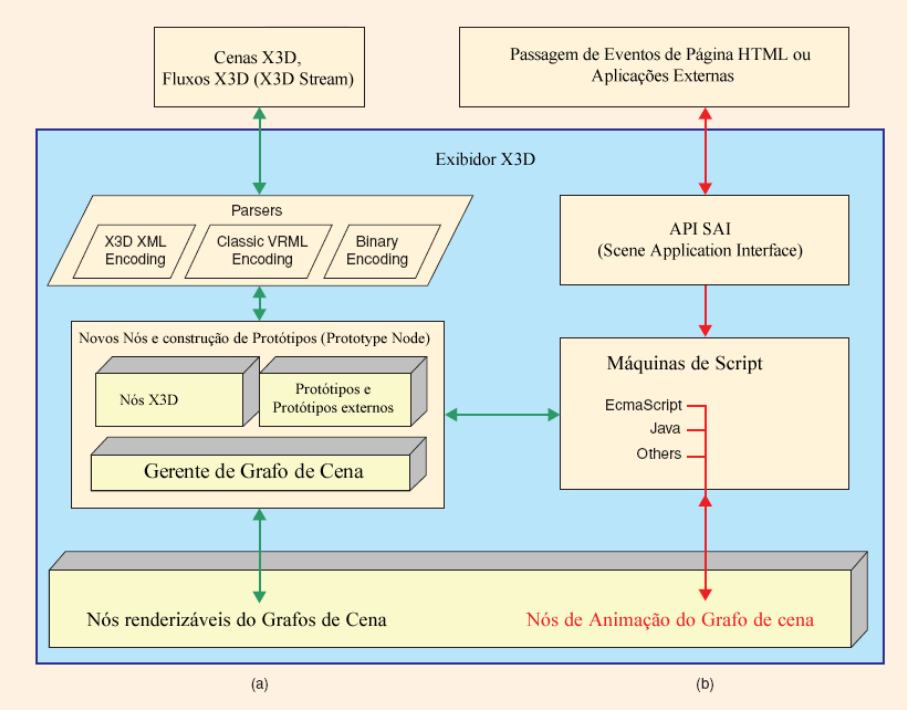

Figura 12 Exemplo dos principais componentes da arquitetura de um exibidor X3D. Adaptado de: (DALY e BRUTZMAN, 2008).

3.1.2.Extensible MPEG-4 Textual Format (XMT)

MPEG-4 (ISO/IEC, 1998) é um padrão ISO/IEC para a codificação de mídias audiovisuais. Além de definir a codificação individual para cada elemento de mídia (áudio, vídeo, texto, objetos sintéticos etc.), o MPEG-4 também define padrões para a descrição de cenas interativas compostas por esses elementos relacionados.

Binary Format for Scene (BIFS) é a camada de cena definida pelo MPEG-4. A partir do BIFS, entretanto, não é possível por meio de uma engenharia reversa,

representar a intenção original do autor. O *framework Extensible MPEG-4 Textual Format* (XMT) foi desenvolvido visando permitir o intercâmbio de conteúdo entre as intenções dos autores em um formato textual de alto nível (KIM, WOOD e CHEOK, Novembro de 2000). XMT consiste de dois níveis de sintaxe e semântica textual: XMT-A e XMT-Ω (ou XMT-O).

XMT-A é uma linguagem cujo objetivo é ser mapeada diretamente para a representação binária (BIFS). Essa linguagem utiliza os mesmos conceitos de grafo de cena e grafo de rotas, herdados de X3D e também representados no BIFS em formato binário. XMT-A favorece o processo de autoria em relação ao BIFS, pois permite ao autor escrever textualmente a especificação do documento. Contudo, no aspecto semântico, não acrescenta nenhuma funcionalidade a essa representação binária (COSTA, 2005). Sendo assim, por ser simplesmente um mapeamento de uma linguagem binária para um formato textual, XMT-A ainda é demasiadamente complexa para ser utilizada por usuários finais.

XMT-Ω, por outro lado, é uma abstração de alto nível das facilidades do BIFS baseada na linguagem SMIL (W3C, 2005), visando simplificar a autoria de cenas MPEG-4. Ao utilizar-se de SMIL, os autores podem descrever o comportamento temporal e o *layout* de apresentações multimídia, bem como associar *hyperlinks* (hiper-elos) com objetos de mídia na apresentação (KIM, WOOD e CHEOK, Novembro de 2000). Adicionalmente, também é possível mesclar comandos do XMT-A e do XMT-Ω em um mesmo documento, aproveitando o melhor de cada um deles.

XMT-Ω reúsa os principais módulos de SMIL. Quando necessário, são feitas adaptações a esses módulos. Em especial, o módulo *Media* e o módulo *Timing and Synchronization* são as principais inovações em relação ao XMT-A e possibilitam que os documentos sejam concebidos com base na intenção dos autores (COSTA, 2005).

Ao estender o módulo *Media*, a linguagem XMT-Ω define novos tipos de mídia específicos do MPEG-4, estendendo o elemento xMedia de SMIL, tanto mídias 2D como 3D. O módulo *Timing and Synchronization* define os contêineres temporais de SMIL e permitem que os diversos objetos que constituem a aplicação sejam sincronizados de uma forma simples.

A Figura 13 apresenta um exemplo de um documento XMT-Ω.

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
```

```
2: <XMT-O>
3:   <head>
4:     <layout metrics="pixel">
5:       <topLayout width="640" height="480"/>
6:     </layout>
7:   </head>
8:   <body>
9:     <par>
10:      <rectangle size="10 10" dur="5s"/>
11:      <img src="img.jpg"/>
12:      <seq>
13:        <box id="box" size="20 20 20" dur="20s"/>
14:        <sphere id="ball" radius="8" dur="20s"/>
15:      </seq>
16:    </par>
17:  </body>
18:</XMT-O>
```

Figura 13 Exemplo de um documento XMT-Ω

A linguagem XMT-Ω divide a estrutura do documento em duas partes, o cabeçalho (<head>) e o corpo (<body>). No cabeçalho estão as informações relacionadas à metadados e ao leiaute da apresentação. O leiaute (*layout*) é definido por uma única janela (<topLayout>), que pode ser composta por várias regiões (<region>). A distribuição espacial das regiões é definida por meio de coordenadas cartesianas, com origem no centro da janela. O corpo do documento é uma composição. Dentro dessa composição têm-se outras composições recursivamente e objetos de mídia que fazem parte da apresentação. As composições XMT-Ω têm semântica temporal, e são herdadas de SMIL. São elas:

- <par>: exibe um ou mais filhos em paralelo.
- <seq>: exibe os filhos um de cada vez, em sequência.
- <excl>: exibe um filho por vez, mas não impõe qualquer ordem específica.

Em adição aos eventos básicos, provenientes de SMIL, XMT-Ω também suporta eventos mais avançados como, por exemplo, aqueles relacionados a objetos 3D. Tanto os objetos 2D como 3D podem gerar eventos como *click*, *mouseup*, *mousedown*, *mouseout* e *mouseover*. Os eventos *viewable*, *near* e *collide* são eventos relacionados a objetos 3D (PEREIRA e EBRAHIMI, 2002).

Diferente de X3D, onde o evento de colisão é entre o objeto na cena e o usuário, o evento de colisão de XMT-Ω é entre dois ou mais objetos. Esses objetos devem ser agrupados em um nó de agrupamento que possui o atributo *collide* igual a *true*. Quando dois ou mais objetos internos a esse agrupamento colidirem entre si, é disparada a ocorrência de um evento de colisão nesse

agrupamento. A Figura 14 apresenta um exemplo que reproduz um objeto de áudio quando ocorre a colisão entre os objetos *ball* e *box*.

```
1: <body>
2:   <group id="ballBox" collide="true">
3:     <sphere id="ball" radius="8" dur="20s">
4:       <transformation translation="-100 50 30">
5:         <animateMotion to="0 0 0" dur="5s"/>
6:       </transformation>
7:     </sphere>
8:     <box id="box" size="20 20 20" dur="20s"/>
9:   </group>
10:  <audio src="crash.wav" begin="ballBox.collide"/>
11: </body>
```

Figura 14 Exemplo de código XMT-Ω demonstrando evento de colisão entre objetos 3D.

Nesta dissertação, são propostos mecanismos que permitem à NCL embutir objetos de mídia 3D representados por grafos de cena (como X3D ou XMT-A, por exemplo). Adicionalmente, também é proposto como NCL pode controlar o comportamento desses objetos, em detrimento do grafo de rotas, apresentando suas vantagens. Quando comparado ao XMT-Ω, a proposta desta dissertação também pode substituir o uso dos contêineres temporais de SMIL. Tanto esta proposta como a do XMT-Ω, entretanto, são similares na medida em que cogitam o uso de linguagens multimídia para o controle da apresentação de cenas 3D.

Diferente do XMT-Ω, o trabalho aqui proposto foca em não embutir na linguagem multimídia a definição do universo tridimensional (ou seja, substituir a estrutura do grafo de cena). O objetivo é manter a linguagem NCL como uma *linguagem de cola*, independente de qual o conteúdo dos objetos que ela controla. Ao se utilizar uma linguagem de cola, tem-se uma melhor separação entre a estrutura espacial, o conteúdo e o comportamento dos objetos. XMT-Ω, por outro lado, simplesmente inclui os *Schemas* de SMIL e X3D, em alguns casos alterando o nome de seus elementos. Tal abordagem entrelaça o conteúdo da cena com o seu comportamento e traz os inconvenientes descritos por (SOARES NETO e SOARES, 2009).

Ao entrelaçar o conteúdo da cena e o seu comportamento, por meio de contêineres temporais, XMT-Ω também torna inviável para um exibidor utilizar as otimizações apresentadas na Subseção 2.1.1. O autor do documento não mais se utiliza da abstração de grafo de cena para organizar a cena, já que na hierarquia do documento também deve considerar o comportamento dos objetos.

O exemplo a seguir, Figura 15, exemplifica um modelo onde existem duas esferas, espacialmente separada (uma na sala e outra no quarto), mas que devem ser apresentadas em sequência (ao final de *bola1*, deve-se iniciar a exibição de *bola2*). Um documento, que leva em consideração a organização espacial da cena, seria o apresentado em (a). Contudo, ao entrelaçar o conteúdo da cena, com sua estrutura espacial, o máximo que o XMT-Ω consegue expressar é o apresentado em (b), que não reflete a semântica espacial do modelo.

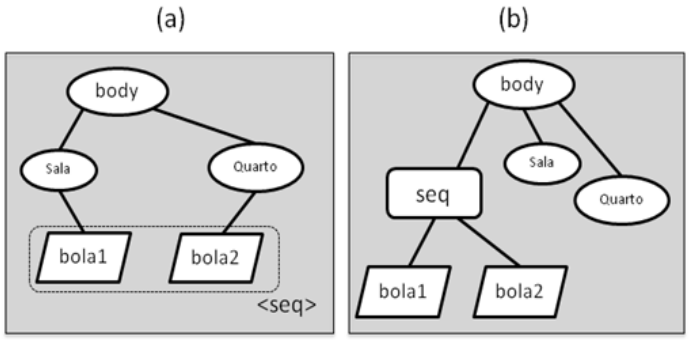

Figura 15 Diferença entre modelagem baseada em grafo de cena (a) e XMT-Ω (b), ao entrelaçar conteúdo e comportamento.

3.1.3.XMT e NCL

Outro trabalho também relacionado a esta dissertação é o encontrado em (COSTA, 2005), onde é apresentado um comparativo entre as linguagens NCL e XMT. Naquele trabalho, se cogita a possibilidade de inclusão de documentos XMT em documentos NCL, ainda na sua versão 2.0, como composições NCL. Entretanto, não é discutido como tais documentos XMT podem ser incorporados à NCL como *objetos de mídia*, embora isso seja citado como um dos possíveis trabalhos futuros. Por exemplo, não se define como as âncoras definidas em NCL devem ser mapeadas nesses objetos.

Adicionalmente, ao mencionar a incorporação dos objetos XMT como composições NCL, (COSTA, 2005) ainda mantém os contêineres temporais (em XMT-Ω) e o grafo de rotas (em XMT-A) como abstrações para controle do comportamento do documento XMT. O que se discute é apenas a notificação dos eventos gerados pelos sensores no grafo de rotas para o formatador NCL. Ainda assim, não fica claro como tais eventos seriam notificados e nem como o autor do documento NCL poderia tirar proveito desses eventos, já que não são propostos novos eventos à NCL.

3.2. Ginga-NCL

Middleware é a camada de software intermediária, responsável por abstrair as especificidades de *hardware* e sistema operacional dos diversos dispositivos existentes. Assim, ao desenvolver uma aplicação seguindo a API de um *middleware*, o autor tem a certeza de que sua aplicação funcionará independente de qual o fabricante ou versão do dispositivo na qual ela irá executar. O Ginga (ABNT, 2007) é o *middleware* padrão do Sistema Brasileiro de TV Digital (SBTVD-T) e é dividido em três subsistemas principais: *Ginga-NCL*, *Ginga-J* e *Ginga-CC (Common Core)*.

O Ginga-NCL é o subsistema responsável por executar aplicações declarativas, desenvolvidas em NCL; o Ginga-J é responsável por executar aplicações imperativas, desenvolvidas em Java; enquanto o Ginga-CC é responsável por suprir funcionalidades básicas de exibição de mídias, gerenciamento de contexto, persistência etc., que são necessárias tanto para o Ginga-NCL como para o Ginga-J.

Esta seção apresenta os principais componentes do Ginga-NCL e do Ginga-CC, que serão úteis na fase de implementação desta dissertação. A Implementação de Referência do Ginga-NCL (COMUNIDADE GINGA-NCL, 2010) foi tomada como base para este trabalho. A Figura 16 apresenta a arquitetura do Ginga-NCL e do Ginga-CC, conforme definido na Implementação de Referência do Ginga-NCL. A seguir são discutidos os principais módulos dessa arquitetura.

Figura 16 Arquitetura do Ginga-NCL e Ginga-CC (*Common Core*) Fonte: (MORENO, 2010)

Componentes do Ginga-NCL

O *Formatador* (exibidor NCL) é o módulo do Ginga-NCL responsável por receber e controlar a exibição de aplicações multimídia escritas em NCL. As aplicações são entregues ao Formatador pelo Ginga-CC. O Formatador requisita os serviços dos módulos Parser XML e Conversores para traduzir o código fonte da aplicação NCL em uma estrutura de dados interna, necessária para a apresentação do documento.

O *Parser XML* é o módulo responsável por ler documentos NCL e traduzi-los em estruturas de dados que representam o modelo NCM. Tais estruturas são mantidas pelo componente *Gerente de Base Privada*, e estarão aptas a receber comandos de edição ao vivo. (COSTA, MORENO, *et al.*, 2006). O *Gerente de Base Privada* é o componente responsável pelo controle do ciclo de vida de aplicações NCL.

O *Escalonador de Apresentação* é responsável por orquestrar a apresentação do documento NCL. A pré-busca de objetos de mídia, a *avaliação de condições nos elos causais* e o *agendamento das ações* que guiam o fluxo da apresentação estão entre as funções desse componente. O Escalonador solicita ao *Conversor* que as entidades do NCM sejam convertidas em estruturas de dados próprias para a apresentação, quando necessário. O Escalonador também é responsável por

comandar o *Gerente de Exibidores*, responsável por instanciar o *Exibidor* (ou *player* de mídia) apropriado, de acordo com o conteúdo da mídia que será exibida em um determinado momento. Na Implementação de Referência do Ginga-NCL, o *Gerenciador de Exibidores* é configurável por meio de um arquivo que associa cada *MIME Type* a uma classe C++, que implementa o respectivo *Adaptador* – discutido mais a frente – para aquele tipo de mídia.

O *Gerente de Leiaute* é o módulo responsável por mapear todas as regiões definidas em uma aplicação NCL para sua respectiva área de apresentação na tela. Adicionalmente, essa área de apresentação também pode estar em outro dispositivo, permitindo ao Ginga-NCL suportar múltiplos dispositivos de exibição. Esse componente, em especial, será estendido, no Capítulo 5, onde são propostas novos tipos de regiões para exibição de objetos de mídia NCL, definidas como superfícies de objetos 3D.

O *Gerenciador de Contexto NCL* é o módulo responsável por adaptar a apresentação de aplicações NCL. Para isso, esse módulo utiliza as informações providas pelo *Gerenciador de Contexto*, discutido mais adiante. A aplicação NCL pode se adaptar ao ambiente em que está executando (perfil do sistema) ou ao perfil do usuário (conforme discutido na Seção 2.3.2).

Componentes do Ginga-CC (Common-Core)

No Ginga-CC, o módulo *Sintonizador* é responsável por receber o conteúdo proveniente de um canal de TV Digital pelo ar. O módulo *Transporte* é responsável por receber dados de outras interfaces de rede. Enquanto, o módulo *Processador de Dados* monitora informações sobre a existência e origem de aplicações interativas.

O *Gerente de Contexto* é responsável por obter informações das características da plataforma e perfil do telespectador e atualizar um banco de dados de variáveis. Esse banco de dados pode ser consultado pelo *Gerenciador de Contexto NCL* e disponibilizado para aplicações NCL.

Além da apresentação de objetos de mídia pré-codificados (imagem, áudio, vídeo, texto, etc.), o Ginga-NCL também possibilita a execução de objetos de mídia imperativos e declarativos. No que se refere à apresentação de objetos de mídia imperativos, um dos tipos suportados pelo Ginga-NCL, padrão do ISDB-T_B e Recomendação ITU-T, são códigos Lua (os denominados NCLua). O módulo

Máquina Lua é o responsável por fazer essa comunicação entre objetos NCLua e a Máquina de Apresentação.

O *Gerenciador Gráfico* é responsável por controlar o modelo gráfico definido por uma determinada plataforma de exibição. Esse componente é modificado pelo presente trabalho ao propor extensões às regiões de NCL. A princípio, a implementação de referência utiliza como *framework* gráfico o DirectFB (DIRECTFB, 2010), o qual, atualmente, dá suporte apenas a gráficos 2D. Com o propósito de utilizar também gráficos 3D, tal componente será modificado para basear-se em OpenGL (SEGAL e AKELEY, 2004), o que também servirá de base para pesquisas futuras.

Cada componente *Exibidor* é responsável por conhecer as especificidades de determinados tipos de conteúdo de mídia e decodificá-los, ou interpretá-los (em casos de linguagens declarativas ou imperativas). O *Gerente de Exibidores* é responsável por escolher qual *Exibidor* deve ser instanciado para exibir, em um determinado momento, um objeto de mídia. O Ginga-NCL foi desenvolvido de forma a facilmente integrar novos tipos de mídia. Para isso, cada *Exibidor* deve implementar uma API comum que permite a comunicação com a Máquina de Apresentação. Os *Exibidores* também são responsáveis por notificar a Máquina de Apresentação sobre eventos que ocorrem enquanto aquele determinado objeto de mídia está em execução, tais como a ocorrência de alguma âncora, início ou fim da apresentação, seleção ou alteração de alguma propriedade do nó de mídia.

Exibidores que não seguem a especificação da API Ginga devem se utilizar de componentes *Adaptadores*. Os *Adaptadores*, na arquitetura do Ginga-NCL, são componentes de software responsáveis por intermediar a comunicação entre o Ginga e os *Exibidores*, de forma que seja possível utilizar-se de *Exibidores* fornecidos por terceiros que não implementam a API determinada pelo Ginga.

Neste trabalho, um novo Exibidor é desenvolvido para apresentar propriamente objetos de mídia X3D. Tal implementação visa servir como caso de uso para a proposta de embutir e controlar o comportamento de objetos de mídia 3D representados por grafos de cena em apresentações NCL. Um novo *Adaptador* também será necessário para manter a compatibilidade com a API Ginga.