UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

SOLUÇÃO PARA SUPORTE A HTTP STREAMING NO MIDDLEWARE GINGA-NCL

Raphael Neves André

JUIZ DE FORA
MARÇO, 2013

SOLUÇÃO PARA SUPORTE A HTTP STREAMING NO MIDDLEWARE GINGA-NCL

RAPHAEL NEVES ANDRÉ

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharel em Ciência da Computação

Orientador: Marcelo Ferreira Moreno

JUIZ DE FORA
MARÇO, 2013

SOLUÇÃO PARA SUPORTE HTTP STREAMING NO MIDDLEWARE GINGA-NCL

Raphael Neves André

MONOGRAFIA SUBMETIDADA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

D.Sc. Marcelo Ferreira Moreno
(Orientador)

D.Sc. Eduardo Barrere

M.Sc. Eduardo Pagani Julio

D.Sc. Alex Borges Vieira
(Suplente)

JUIZ DE FORA, MG – BRASIL
MARÇO, 2013

AGRADECIMENTOS

À minha família e amigos, pelo apoio e compreensão constantes, auxiliando no alcance de sonhos e objetivos.

Ao Prof. Dr. Marcelo Ferreira Moreno, orientador, pelos ensinamentos, conhecimento compartilhado e disponibilidade.

Enfim, a todos que de alguma forma apoiaram a realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

A transmissão de conteúdo multimídia transformou-se em um importante serviço provido aos usuários na Internet. O streaming adaptativo possibilita que a distribuição desse conteúdo na Web seja feita de forma a considerar as condições do ambiente de execução, seja avaliando a infraestrutura de rede ou a capacidade de processamento do mesmo. Por intermédio da NCL (*Nested Context Language*) e do middleware Ginga, essa técnica também pode ser utilizada em terminais de TV Digital e IPTV, de forma que aplicações que contam com a utilização de streaming de conteúdo multimídia possam ser executadas de maneira adaptativa, permitindo que a reprodução desse conteúdo ocorra com maior fluidez e, consequentemente, aprimorando a qualidade do serviço (QoS – *Quality of Service*) oferecida aos telespectadores. O presente trabalho apresenta uma proposta de solução de streaming adaptativo sobre o protocolo HTTP para essas tecnologias, através de uma varredura bibliográfica sobre o assunto contextualizado e do desenvolvimento de uma implementação que utiliza-se dos recursos da NCL 3.0 para possibilitar a reprodução de conteúdo multimídia de modo a se adequar às alterações do ambiente de execução.

Palavras-chave: Multimídia, streaming, HTTP, Web, IPTV, TV Digital, adaptativo.

ABSTRACT

The transmission of multimedia content became an important service provided to users on the Internet. The adaptive streaming enables the distribution of content on the Web to be performed considering some conditions of the execution environment, by evaluating its connectivity status or processing capacity. Using NCL (Nested Context Language) and the Ginga middleware, this technique can also be applied to Digital TV and IPTV terminals, so that applications that rely on the use of multimedia content streaming can be implemented in an adaptive way, allowing the playback of that content to occur with greater fluidity and consequently improving the quality of service (QoS) offered to viewers. This paper proposes a solution for adaptive streaming over HTTP protocol based on those technologies, founded on a research in the literature for this subject and the development of an implementation that uses NCL 3.0 resources. This proposal allows the multimedia playback to be adapted in face of changes in the execution environment.

Keywords: Multimedia, streaming, HTTP, Web, IPTV, Digital TV, adaptive.

1 INTRODUÇÃO

A transmissão de conteúdo multimídia tem sido um importante serviço provido aos mais diversos tipos de usuários na Internet nos últimos anos.

As técnicas de *streaming* têm se mostrado grandes contribuidoras desse fato, visto que estão sendo impulsionadas por uma melhora considerável na infraestrutura de rede, como por exemplo o aumento da largura de banda. O *streaming* baseado no protocolo HTTP (*Hypertext Transfer Protocol*) já vem sendo bastante utilizado e bem aceito pelos usuários. Por contar com um protocolo bem conhecido, bastante compatível com a Web e, ainda, com uma arquitetura de distribuição de conteúdo bastante simples e barata, algumas técnicas, sobretudo o *download progressivo*, contribuíram de maneira importante para a escalabilidade hoje percebida para aplicações de conteúdo multimídia na Internet.

No entanto, a fim de melhorar ainda mais a qualidade do serviço (QoS – *Quality of Service*) oferecida e na tentativa de aumentar ainda mais a escalabilidade de aplicações envolvidas nesse contexto, fez-se necessária a elaboração de uma solução adaptativa de *streaming*, que levasse em consideração as alterações do ambiente de execução, tais como as condições da rede e a capacidade de processamento, durante o tempo de apresentação do conteúdo.

Em se tratando de TV Digital e IPTV, essas técnicas implantadas na Web ainda não estão sendo utilizadas pelos criadores de conteúdo, sejam eles as próprias emissoras ou os desenvolvedores de aplicações. Todavia, os terminais dessas tecnologias possuem capacidade de agregar técnicas de *streaming* adaptativo e prover um novo serviço aos telespectadores.

Através de recursos da linguagem NCL (*Nested Context Language*) e do *middleware* Ginga, o presente trabalho propõe-se a apresentar uma solução de *HTTP Streaming* adaptativo para essas plataformas. No próximo capítulo será apresentada uma breve definição de TV Digital, com suas características principais, a possibilidade de se criar interatividade e uma breve descrição do *middleware* adotado no padrão brasileiro de TV Digital. No terceiro capítulo encontram-se a linguagem NCL e a apresentação dos seus principais elementos. Os principais detalhes e um breve histórico da evolução do *HTTP Streaming*, apresentando uma descrição breve dos principais protocolos utilizados e da maneira como é realizada a distribuição de conteúdo, bem como detalhes sobre algumas soluções propostas de *streaming* adaptativo para a *Web*, são encontrados no quarto capítulo deste trabalho. No quinto capítulo, o relacionamento entre o *streaming* adaptativo e a TV Digital, mencionando de que forma é possível agregar essa técnica aos terminais de TV Digital e IPTV, por meio da NCL e do Ginga, como descrito anteriormente, e apresentando detalhes sobre o desenvolvimento e implementação da solução proposta,

expondo alguns requisitos e mostrando como os recursos e elementos da NCL podem ser utilizados para a realização de *streaming* adaptativo. Por fim, o sexto capítulo expõe as intenções para a realização de trabalhos futuros, bem como melhorias que podem ser feitas na implementação realizada.

2 TV DIGITAL

A TV digital, segundo BOLAÑO E VIEIRA (2004), é um sistema de radiodifusão que transmite sinais digitais em lugar dos analógicos. É mais eficiente no que diz respeito à recepção dos sinais e apresenta uma série de inovações sob o aspecto estético, como a possibilidade de ter-se uma imagem mais larga e com maior resolução, e ainda um som estéreo envolvente e disponibilidade de vários programas em um mesmo canal.

A maior novidade parece fazer referência à capacidade de permitir a convergência entre diversos meios de comunicação eletrônicos, tais como telefonia fixa e móvel, radiodifusão, transmissão de dados e o acesso à Internet.

Para SOARES E BARBOSA (2012), uma das principais diferenças sensíveis, decorrentes da mudança da TV analógica para a TV digital, é a melhor qualidade de imagem e som, impulsionada pelo tratamento dos ruídos no sinal original transmitido.

Esse ruído aleatório, porém, está presente em todo o espectro de frequências e não há como ser evitado. Pode provocar queda na qualidade do sinal recebido, no caso da TV analógica, ou pode modificar um nível digital do sinal transmitido a ponto de o mesmo passar a ser confundido com outro nível, quando se trata de TV digital.

2.1 CARACTERÍSTICAS

Conforme expõe SOARES E BASBOSA (2012), a qualidade do sinal está diretamente relacionada com a potência do mesmo e do ruído nele percebido. À medida que a potência do sinal diminui e/ou a intensidade do ruído aumenta, a probabilidade de erro de bit é acrescida.

Para evitar o problema, todos os padrões de transmissão dos sistemas de TV digital terrestre utilizam códigos corretores de erro, que são responsáveis por corrigir os erros introduzidos pelo canal caso a taxa de erro encontre-se abaixo de um limiar. Nesse caso, não haverá queda da qualidade da imagem. Todavia, caso a taxa de erros esteja acima do limite estabelecido, o sistema não será capaz de corrigir o código transmitido e, além disso, poderá até mesmo introduzir mais erros.

Outra diferença entre a TV digital e a TV analógica é o fato de que, por pior que o sinal analógico chegue à televisão, o telespectador consegue recebê-lo. No caso da TV digital, ou o sinal chega com boa qualidade ou não será recebido. Por isso, tem-se a preocupação a despeito da importância da avaliação de qual sistema de transmissão é o mais robusto dado um respectivo cenário, para garantir a chegada adequada do sinal digital nas residências.

Ainda de acordo com SOARES E BARBOSA (2012), a melhor qualidade de imagem e som também decorre da aplicação de técnicas de compressão de dados em sinais digitais, permitindo a produção de sinais de melhor qualidade na fonte. Dentro de uma mesma faixa de 6 MHz é possível transmitir imagens de 1080 linhas, com 1920 pixels cada uma delas, e ainda transmitir áudio no padrão 5.1, dando ao telespectador maior sensação de imersão na cena.

O impacto da TV digital é muito mais significativo do que apenas uma simples troca do sistema de transmissão e um aumento de qualidade de imagem e som. Mais do que isso, um sistema de TV digital permite um nível de flexibilidade inatingível com a difusão analógica, proveniente da possibilidade de expandir as funções do sistema por aplicações construídas sobre a base de um sistema padrão de referência. Essas aplicações podem garantir uma integração de capacidade computacional significativa no dispositivo receptor do sinal digital, permitindo o surgimento de uma vasta gama de novos serviços.

2.2 TV DIGITAL INTERATIVA

De acordo com MONTEZ E BECKER (2004), para um sistema ser interativo, é necessário que o usuário possa modificar a forma e o conteúdo do ambiente. A TV interativa que conhecemos é meramente reativa, visto que os telespectadores apenas reagem a estímulos proporcionados pela emissora. Ainda não há um papel ativo em relação à programação audiovisual.

Para FERNANDES (2006), é possível classificar a TV interativa atual em nove grandes grupos:

- TV avançada (Enhanced TV) – tipo de conteúdo televisivo que abrange texto, vídeo e elementos gráficos. Na sua forma mais simples, observa-se uma apresentação integrada desses elementos, organizada por uma grade de programação;
- Internet na TV – permite o acesso à Internet, com todas suas funcionalidades, através do aparelho televisor;
- TV individualizada – permite a personalização da TV, de forma que o usuário poderá escolher ângulos de câmeras, personalizar a interface e organizar as janelas na tela;
- Vídeo sob demanda – permite aos telespectadores assistirem a determinado programa no momento em que desejarem, sem restrição quanto ao horário em que é transmitido pela emissora. A emissora poderá disponibilizar toda a grade de programação, com exceção dos programas ao vivo;

- Personal Video Recorder (PVR) – também recebe a denominação de Personal TV ou Digital Video Recorder (DVR). Permite a gravação digital de programas apenas especificando detalhes sobre a atração televisiva. O equipamento utilizado para atender a este modelo de interatividade contém um disco rígido, onde ficam armazenadas as mídias requisitadas;
- Walled Garden – cabe-se de um portal que contém um guia de aplicações interativas. Esclarece ao usuário o que é possível fazer, o que encontra-se disponível, e serve de canal de entrada para essas aplicações;
- Console de jogos – permite o uso da TV para jogos, seja escolhendo a própria TV como adversária ou em rede, com outros jogadores;
- Guia de programação eletrônica – cabe-se de um portal contendo um guia da programação, semelhante ao encontrado na TV a cabo;
- Serviços de teletexto – informações são fornecidas pelos transmissores em forma de texto, podendo se sobrepor às imagens ou ocupar a tela inteira do vídeo. Informações adicionais à programação, informações econômicas e meteorológicas são bastante comuns.

Com a TV, de fato, interativa, o telespectador passa a ter um canal de interatividade para se comunicar com a emissora. O grau dessa interatividade é dependente da oferta dos serviços e, principalmente, das condições do canal (MONTEZ E BECKER, 2004).

Como expõe FERNANDES (2006), é através desse canal de interatividade que as respostas do telespectador interativo chegam até a emissora de televisão. Por mais simples que a aplicação seja, é preciso um meio que leve as interações dos usuários até o transmissor. Esse meio pode ser o telefone, fibra ótica, rede sem fio, rádio, entre outros.

No caso do Brasil, o problema da adoção de um sistema de TV digital interativa gira em torno de gargalos infraestruturais nos lares brasileiros. O candidato natural a canal de retorno, o telefone fixo, está presente em somente 30% das residências. Ao se tratar da telefonia móvel, menos da metade da população teria acesso a um canal de interatividade. As redes sem fio ou por cabos, seja fibra ótica ou cabos coaxiais, representam uma boa opção somente para cidades amplamente cabeadas. Cidades pequenas estariam exclusas desse domínio. É nesse estágio que se encontra a interatividade da TV digital no nosso país.

De acordo com SOARES E BARBOSA (2012), podemos encontrar quatro níveis de interatividade em um sistema de TV digital. O primeiro deles é chamado de *interatividade local*, que é definido quando não há um canal de retorno e as aplicações utilizam apenas os dados transmitidos por difusão; o segundo pode ser notado quando o sistema opera com um canal de retorno de forma unidirecional, permitindo ao telespectador somente o envio de dados, seja solicitação de compra de um produto ou seu voto em uma enquete proposta; o

terceiro nível de interatividade é definido por um canal de retorno bidirecional assimétrico, que possibilita o receptor fazer o carregamento de dados utilizados pelas aplicações, através da Web, por exemplo; o quarto e último nível de interatividade conta com um canal bidirecional que permite o envio de dados, de forma que o receptor atue como uma espécie de pequena emissora. Esse nível de interatividade, denominada *interatividade plena*, possibilita o que vem sendo chamado de “TV Social”, que é caracterizada pela troca de informações dentro de um grupo de telespectadores de um mesmo programa.

O sistema brasileiro de TV digital terrestre permite, em suas normas, todos os níveis de interatividade (SOARES E BARBOSA, 2012).

2.3 MIDDLEWARE

De acordo com SOARES E BARBOSA (2012), uma aplicação de TV deve ser capaz de ser executada em qualquer plataforma de hardware e sistema operacional. Para isso, uma nova camada é acrescentada aos padrões de referência de um sistema de TV digital. A essa nova camada atribui-se o nome de *middleware*.

O *middleware* é um dos componentes mais importantes de um sistema de TV digital porque, na prática, é o responsável por regular as relações entre os produtores de conteúdo e os fabricantes de aparelhos receptores. No sistema brasileiro de TV digital, o *middleware* adotado recebe o nome de “Ginga”.

O Ginga é processado a partir dos receptores *set-top-boxes* com a função de suportar as aplicações que são transmitidas em conjunto com a programação recebida. É então responsável pela intermediação entre os aplicativos e o sistema operacional Linux, como mencionam ZANCANARO *et al.* (2009).

Como expõe SOARES E BARBOSA (2012), um *middleware*, bem como o Ginga, deve atender aos seguintes requisitos:

- Suporte ao sincronismo de uma forma geral e, particularmente, à interação do usuário;
- Suporte à adaptação de conteúdo e da forma como um conteúdo é exibido;
- Suporte a múltiplos dispositivos de exibição;
- Suporte à edição ao vivo (em tempo de exibição);
- Suporte à definição de relacionamentos de sincronismo espacial e temporal separado da definição do conteúdo dos objetos de mídia relacionados.

Desenvolvido pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) em conjunto com a Universidade Federal da Paraíba (UFPB), o sistema Ginga é subdividido em dois subsistemas principais interligados, chamados de Ginga-NCL (para aplicações declarativas em NCL) e Ginga-J (para aplicações procedurais em Java). O uso de um

paradigma ou outro é dependente das funcionalidades requeridas no projeto de cada aplicação (ZANCANARO *et al.*, 2009).

3 NCL

A NCL é uma linguagem declarativa, uma aplicação XML, e é baseada no modelo conceitual NCM (*Nested Context Model*), trazendo uma separação clara entre os conteúdos de mídia e a estrutura de uma aplicação. Um documento NCL apenas define como os objetos de mídia são estruturados e relacionados, nos aspectos temporal e espacial (SOARES E BARBOSA, 2012).

Segundo ZANCANARO *et al.* (2009), a linguagem foi desenvolvida pela PUC-Rio almejando facilitar as especificações de interatividade, sincronismo espaço-temporal entre objetos de mídia, adaptabilidade, suportar múltiplos dispositivos e suportar programas ao vivo interativos não lineares. O formatador NCL, presente no subsistema Ginga-NCL, recebe um documento NCL e controla a sua apresentação, fazendo com que os objetos de mídia sejam apresentados no momento programado.

Para a construção de documentos de hipermídia, é necessário responder a quatro perguntas básicas, que são: **O que** se quer tocar? **Onde** tocar? **Como** tocar? **Quando** tocar?

Ao responder à primeira pergunta, o programador irá escolher os objetos de mídia propriamente ditos, tais como: vídeos, imagens, sons. O resultado da segunda pergunta introduzirá as regiões da tela onde os objetos de mídia escolhidos estarão localizados quando executados. A resposta da próxima indagação leva o programador a definir os descritores da linguagem, que são responsáveis pela relação entre a mídia e a região, definindo como a mídia deverá ser apresentada no local determinado. Respondendo à última pergunta, enfim, chega-se à definição dos conectores, links, portas e outros elementos (ZANCANARO *et al.*, 2009).

Cada um dos elementos básicos da linguagem será descrito nas subseções a seguir.

3.1 ESTRUTURA BÁSICA DE UM DOCUMENTO NCL

Um documento NCL é sempre formado por um elemento raiz, denominado *ncl*, que contém outros dois elementos, *head* e *body*. O primeiro representa o cabeçalho do documento e possui as informações relativas à exibição dos componentes. O último representa o corpo do documento, onde são definidos os componentes e os seus relacionamentos (ANTONACCI *et al.*, 2000). A Figura 1 ilustra a estrutura básica inicial de um documento NCL.

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

Figura 1 – Estrutura de um documento NCL.
Fonte: ANTONACCI ET AL, 2000.

A NCL permite adicionar objetos de mídia baseados em XHTML (*Extensible Hyper Text Markup Language*), mas não os embute. A linguagem apenas tem suporte do formatador para exibir esses arquivos.

Além do XHTML, outra linguagem que a NCL dá suporte é a LUA: uma linguagem de *script* que combina sintaxe procedural e declarativa. Escrita em C, é simples, leve, robusta, embarcada e de código aberto. O programa principal escrito em NCL tem a função de invocar trechos de código LUA, escrevendo e lendo variáveis (ZANCANARO *et al.*, 2009).

3.2 ELEMENTO *MEDIA*

Um nó de mídia caracteriza o objeto de conteúdo propriamente dito, seja vídeo, imagem, texto, entre outros. O nó de mídia deve identificar o arquivo com o conteúdo da mídia e também pode referenciar um descritor usado para controlar a apresentação desse objeto de mídia (RATAMERO, 2007).

Todo elemento *media* possui um identificador, definido pelo atributo *id*, para que possa ser posteriormente referenciado. Também possui um atributo denominado *src*, que contém um URI (*Uniform Resource Identifier*) para a localização do conteúdo de mídia.

Os elementos *media* podem conter elementos filhos denominados *area*. Os elementos *area* possuem um identificador (atributo *id*) e delimitam trechos, no tempo e no espaço, do conteúdo do seu objeto de mídia pai. Os atributos *begin* e *end* definem o instante no qual a âncora é iniciada e terminada, respectivamente.

Ademais, elementos *media* ainda podem conter elementos *property*. Todo elemento *property* possui um identificador válido no escopo do objeto de mídia, definidos por seu atributo *name*. Propriedades como “left”, “top” e “width” são exemplos de como definir a área

de apresentação do objeto de mídia em relação à tela total do dispositivo utilizado para a reprodução da aplicação NCL (SOARES E BARBOSA, 2012).

A Figura 2 ilustra como os elementos *media* podem ser usados nos documentos NCL.

```
<media id="animation" src="../media/animGar.mp4">
        <area id="segDrible" begin="12s"/>
        <area id="segPhoto" begin="41s"/>
        <property name="width" value="100%"/>
        <property name="heigth" value="100%"/>
        <property name="zIndex" value="2"/>
</media>
<media id="choro" src="../media/choro.mp4"/>
<media id="drible" src="../media/drible.mp4">
        <property name="left" value="5%"/>
        <property name="top" value="6.7%"/>
        <property name="width" value="18.5%"/>
        <property name="heigth" value="18.5%"/>
        <property name="zIndex" value="3"/>
</media>
```

Figura 2 – Elementos <media>, <area> e <property>.
Fonte: SOARES E BARBOSA, 2012.

3.3 ELEMENTO *REGION*

Uma região é uma área na tela (ou outro dispositivo de saída) onde será exibido um determinado nó de mídia. Essas regiões podem ser aninhadas, tornando a estrutura mais organizada. A definição de todas as regiões deve ocorrer no cabeçalho do documento NCL (RATAMERO, 2007).

De acordo com SOARES E BARBOSA (2012), as regiões onde os diversos objetos de mídia serão posicionados não precisam ser especificadas por meio dos elementos *property*, filhos dos elementos *media* correspondentes. Essa definição feita de forma independente, através dos elementos *region*, permite o reuso das mesmas regiões em outras aplicações NCL.

O conjunto de elementos *region* é definido como filho do elemento *regionBase*. Todo elemento *region* possui um identificador e atributos homônimos aos definidos pelos elementos *property*, definindo sua área de exibição em relação à região pai. Quando o elemento não possui uma região pai, o posicionamento é determinado levando-se em consideração a tela total do dispositivo de exibição.

A Figura 3 exemplifica como os elementos *region* são utilizados em NCL.

```
<regionBase>
       <region id="screenReg" width="100%" height="100%"
                                                  zIndex="2">
              <region id="frameReg" left="5%" top="6.7%"
                     width="18.5%" height="18.5%" zIndex="3"/>
       </region>
</regionBase>
```

Figura 3 – Elementos <region> e <regionBase>.
Fonte: SOARES E BARBOSA, 2012.

3.4 ELEMENTO *DESCRIPTOR*

Segundo RATAMERO (2007), um descritor define como será apresentado um nó de mídia, incluindo em que região ele aparecerá. Os descritores devem ser definidos no cabeçalho do documento NCL.

Como expõe SOARES E BARBOSA (2012), é preciso especificar como as regiões definidas são associadas aos vários objetos de mídia definidos. Isso se dá através dos elementos *descriptor*. O conjunto de todos os descritores é definido como filho do elemento *descriptorBase*.

O elemento *descriptor* especifica as características iniciais para a exibição de um objeto de mídia, inclusive seu posicionamento, dado pelo seu atributo *region*, que é usado para referenciar determinada região. É ligado a um objeto de mídia pelo atributo *descriptor* do elemento *media* que representa o objeto.

A Figura 4 ilustra como os descritores são usados nas aplicações NCL.

```
<descriptorBase>
    <descriptor id="screenDesc" region="screenReg"/>
    <descriptor id="photoDesc" region="frameReg" explicitDur="5s"/>
    <descriptor id="audioDesc"/>
    <descriptor id="dribleDesc" region="frameReg"/>
</descriptorBase>
```

Figura 4 – Elementos <descriptor> e <descriptorBase>.
Fonte: SOARES E BARBOSA, 2012.

3.5 ELEMENTO *CONNECTOR*

Em NCL o sincronismo não é feito por marcas de tempo (timestamps), mas sim por relações de causalidade definidas nos conectores. Um elemento causalConnector representa uma relação causal que pode ser utilizadas por elementos link na definição de relacionamentos entre objetos. Nessa relação causal, a partir do momento em que uma condição é satisfeita, uma ação é disparada.

Os elementos simpleCondition, compoundCondition, simpleAction e compoundAction, filhos do elemento causalConnector, definem as condições e ações (simples ou compostas) de ativação e execução dos elos, respectivamente. O elemento connectorParam define parâmetros para o conector cujos valores serão definidos pelos elos que o utilizam (SOARES E BARBOSA, 2012).

O conjunto de elementos *causalConnector* é definido como filho do elemento *connectorBase* e fica localizado no cabeçalho do documento NCL, como apresentado na Figura 5.

```
<head>
   <connectorBase>
      <causalConnector id="onBeginStart_delay">
        <connectorParam name="delay"/>
        <simpleCondition role="onBegin"/>
        <simpleAction role="start" delay="$delay"/>
      </causalConnector>
      <causalConnector id="onBeginStart">
        <simpleCondition role="onBegin"/>
        <simpleAction role="start"/>
      </causalConnector>
      <causalConnector id="onEndStop">
        <simpleCondition role="onEnd"/>
        <simpleAction role="stop"/>
      </causalConnector>
   </connectorBase>
</head>
```

Figura 5 – Elemento <head> e seus elementos filhos.
Fonte: SOARES E BARBOSA, 2012.

3.6 ELEMENTO *LINK*

De acordo com SOARES E BARBOSA (2012), um elo (ou link) define o relacionamento de sincronismo entre interfaces de objetos de uma aplicação NCL. Seu comportamento é definido pelo conector associado.

Para fazer a associação de objetos com os papéis de um conector, um elo utiliza elementos *bind*. O atributo *component* de um bind define qual o objeto que desempenhará determinado papel indicado no atributo *role* do mesmo. Os papéis são definidos pelo conector utilizado pelo elo, informado através do atributo *xconnector* do elemento *link*. Há ainda a possibilidade de se definir papéis para as âncoras das mídias (elemento *area*, filho do elemento *media*), através do atributo *interface* do bind.

O elemento *bind* ainda pode conter um elemento filho, denominado *bindParam*, que é responsável por atribuir valores aos parâmetros definidos no conector associado ao elo.

A Figura 6 mostra como os elos são definidos nos documentos NCL, dentro do corpo dos mesmos.

```
<link  id="lMusic" xconnector="onBeginStart_delay">
       <bind role="onBegin" component="animation"/>
       <bind role="start" component="choro">
              <bindParam name="delay" value="5s"/>
       </bind>
</link>
```

Figura 6 – Elementos <link> e <bind>.
Fonte: SOARES E BARBOSA, 2012.

3.7 ELEMENTO *CONTEXT*

Um contexto agrupa objetos e elos. O elemento *body* de todo documento NCL é um caso particular de contexto que representa a aplicação como um todo.

Os demais contextos de uma aplicação NCL são definidos pelo elemento *context*. Um contexto pode aninhar outros contextos, mas existe uma restrição: um contexto não pode conter recursivamente a si mesmo. O aninhamento de contextos pode refletir a estrutura do documento e ajudar o programador a organizar os segmentos da aplicação (SOARES E BARBOSA, 2012).

Segundo RATAMERO (2007), o atributo *id* do elemento *context* define um nome único pelo qual o contexto será referenciado, o atributo *descriptor* identifica qual descritor definirá a apresentação do contexto, e o atributo *refer* faz referência a outro contexto já definido, do qual este contexto herdará todas as informações, exceto o atributo *id*.

Como afirma SOARES E BARBOSA (2012), o contexto encapsula os objetos e elos que contém. Sendo assim, para acessar um objeto dentro de um contexto, é necessário definir portas no contexto que mapeiam para os objetos que se deseja acessar, como descrito na próxima seção.

A Figura 7 ilustra como um contexto é definido em uma aplicação NCL, dentro do corpo do documento.

```
<context id="advert">
...
    <media id="reusedGlobalVar" refer="globalVar"
                                          instance="instSame"/>
...
    <link id="lIcon" xconnector="conEx#onBeginVarStart">
       <bind role="onBegin" component="reusedAnimation"
                                          interface="segIcon"/>
       <bind role="var" component="reusedGlobalVar"
                              interface="service.interactivity"/>
       <bind role="start" component="icon"/>
    </link>
...
</context>
```

Figura 7 – Elemento <context> e elementos filhos.
Fonte: SOARES E BARBOSA, 2012.

3.8 ELEMENTO *PORT*

Portas servem para garantir acesso externo ao conteúdo de um contexto. Dessa forma, para que um elo aponte para um nó de mídia interno a um contexto, deve apontar para uma porta que leve ao nó interno desejado. É necessária uma porta de entrada que aponte para o primeiro nó a ser apresentado quando da execução do documento NCL.

O atributo *id* atribui um nome único, pelo qual a porta será referenciada. O atributo *component* define a qual mídia ou contexto esta porta está associada. E o atributo *interface* indica a qual porta ou a qual âncora a porta definida deve ser relacionada (RATAMERO, 2007).

A Figura 8 mostra como um elemento *port* deve ser definidono *body* do documento.

```
<body>
    <port id="entry" component="animation"/>

    <!–definição dos diversos elementos de <media> do documento -->

    <!–definição dos diversos relacionamentos, elementos <link> do
                                                documento -->
</body>
```

Figura 8 – Elementos <body> e <port>.
Fonte: SOARES E BARBOSA, 2012.

4 HTTP STREAMING

Até a década de 1990, a Internet era usada primordialmente em meio acadêmico para interligação com servidores remotos, envio e recebimento de notícias e correio eletrônico. Assim sendo, no início da década mencionada, entrou em cena uma nova aplicação importantíssima, a *World Wide Web*. A *Web* é a aplicação da Internet que chamou a atenção do público em geral, transformando drasticamente a maneira como as pessoas interagiam dentro e fora do seu ambiente de trabalho.

Talvez tamanha atração se dê pelo funcionamento sob demanda da *Web*. Usuários recebem o que querem, quando querem, diferentemente da transmissão de rádio e televisão. Além disso, a *Web* possui muitas outras características que são muito bem vistas pelas pessoas, e cada vez mais oferece um menu de interfaces para vastas quantidades de material de vídeo e áudio armazenado na Internet, e acessados sob demanda (KUROSE E ROSS, 2010).

Atualmente a distribuição de vídeo tem um peso dominante no tráfego de Internet, maioritariamente devido a streaming de vídeo armazenado e streaming de vídeo em tempo real. Tal fato está associado em parte ao aumento das ofertas de largura de banda, que possibilitou os usuários a utilizarem cada vez mais os serviços de vídeo, visto que agora possuem maior velocidade de acesso à Internet (MARQUES *et al.*, 2012).

De acordo com KUROSE E ROSS (2010), em uma aplicação multimídia, normalmente o programa cliente inicia a reprodução do conteúdo alguns segundos após começar a receber o arquivo do servidor. Isso significa que o cliente estará reproduzindo áudio/vídeo de uma parte do arquivo simultaneamente ao recebimento de partes do arquivo que estão mais à frente. Essa técnica, conhecida como fluxo contínuo (*streaming*), evita ter de descarregar todo o arquivo antes de começar a reproduzi-lo.

Conforme TSCHÖKE (2001), entende-se por *streaming* de vídeo armazenado a aplicação cujos clientes requisitam *streams* a partir de arquivos que estão armazenados em servidores. O conteúdo multimídia, nesse caso, foi pré-gravado e armazenado. Como resultado, o usuário pode controlar o vídeo mostrado à distância com funções similares às disponíveis em um videocassete. Para esse tipo de aplicação, a transmissão de conteúdo só acontecerá sob demanda do cliente, podendo existir vários clientes conectados simultaneamente ao servidor.

A aplicação que utiliza o *streaming* de vídeo ao vivo é similar à tradicional transmissão de rádio e televisão (*broadcast*), na qual o cliente assume um papel passivo e não controla quando o *stream* começa ou termina, nem possui qualquer controle sobre a exibição da mídia. A diferença dessas aplicações para transmissões de radiodifusão se dá pelo fato de a transmissão ser feita via Internet (TSCHÖKE, 2001). De acordo com KUROSE

E ROSS (2010), como o fluxo de áudio e vídeo ao vivo não é armazenado, um usuário não pode adiantar a exibição do conteúdo que está recebendo. Contudo, com armazenamento local de dados dos *streams* recebidos, operações interativas tais como pausa e retrocesso se tornam possíveis. Essa técnica, conhecida como *trick mode*, é facilmente encontrada em dispositivos DVRs (*Digital Video Recorders*) e sistemas de vídeo sob demanda.

Ainda existem as aplicações de vídeo interativo em tempo real, que permitem às pessoas utilizarem áudio e vídeo para comunicação. Softwares de telefonia e videoconferência na Internet são exemplos desse tipo de aplicação.

4.1 PROTOCOLO HTTP

O protocolo de transferência hipertexto (HTTP) é um protocolo da camada de aplicação da *Web* e é implementado em dois programas: um programa cliente e outro servidor. Esses programas, executados em sistemas finais diferentes, conversam entre si por meio de troca de mensagens HTTP.

Uma página Web é constituída de objetos, que são basicamente arquivos, acessados com um URL único. O HTTP define como clientes requisitam páginas *Web* aos servidores e como elas se transferem a clientes. Basicamente, quando um usuário requisita uma página, o programa cliente (normalmente o *browser*) envia ao programa servidor mensagens de requisição HTTP para os objetos da página. O servidor recebe as requisições e responde com mensagens de resposta HTTP que contêm os objetos.

O HTTP usa o TCP (*Transmission Control Protocol*) como seu protocolo de transporte subjacente. O cliente HTTP primeiramente inicia uma conexão TCP com o servidor e, depois de receber a resposta de conexão sucedida, envia mensagens e recebe respostas do servidor com os devidos objetos requisitados. O servidor HTTP não mantém nenhuma informação sobre clientes, logo é um protocolo sem estado. Se um determinado cliente solicita o mesmo objeto mais de uma vez em um período de poucos segundos, o servidor enviará o objeto quantas vezes requisitado for (KUROSE E ROSS, 2010).

De acordo com TANEMBAUM (2003), o modo habitual de um navegador entrar em contato com um servidor é estabelecer uma conexão TCP para a porta 80 da máquina servidora, embora esse procedimento não seja exigido formalmente. A vantagem de se usar o TCP é que nem os navegadores nem os servidores têm de se preocupar com mensagens perdidas, mensagens duplicadas, mensagens longas ou confirmações. Todos esses aspectos são tratados pela implementação do TCP.

No HTTP/1.0, depois que a conexão era estabelecida, uma única solicitação era enviada e uma única resposta era devolvida. A partir daí, a conexão TCP era encerrada. Em um mundo no qual as páginas Web típicas consistiam em texto HTML (*Hypertext Markup*

Language), esse método era adequado. Depois de alguns anos, as páginas *Web* começaram a ganhar grande número de ícones, imagens e outros atrativos visuais e, dessa forma, estabelecer uma conexão TCP para transportar um único ícone se tornou um modo de operação muito dispendioso.

Lançou-se então o HTTP/1.1, especificado na RFC 2616, que admite conexões persistentes. Com elas, é possível estabelecer uma conexão TCP, enviar uma solicitação e obter uma resposta e, logo em seguida, enviar solicitações adicionais e receber respostas adicionais, amortizando o custo da instalação e liberação do TCP para cada solicitação.

4.2 STREAMING

Streaming é uma técnica na qual o sinal do conteúdo multimídia é transmitido ao cliente e sua apresentação inicia-se após uma momentânea espera para armazenamento dos dados em um *buffer*. Tal abordagem reduz o tempo de inicio da exibição e também elimina a necessidade de armazenamento local dos arquivos (TSCHÖKE, 2001).

De acordo com MARQUES *et al.* (2012), experimentalmente o conceito de *streaming* baseia-se na idéia de que, quando um usuário opta por reproduzir um conteúdo multimídia em uma página *Web*, tal conteúdo começa a ser reproduzido imediatamente e continua de um modo aproximadamente constante até o fim.

Segundo SANTOS (2010), uma das principais características do *streaming* tradicional é o fato de possuir sessões. Iniciadas no momento em que é estabelecida a conexão e terminadas quando a transmissão é completada ou interrompida, essas sessões são monitoradas e contam apenas com uma transmissão contínua de um fluxo de dados.

Conforme CLEMENTE (2006), normalmente os protocolos envolvidos em um *streaming* de vídeo são o RTSP e o RTP. Tais protocolos serão descritos nas subseções a seguir.

4.2.1 RTSP

O RTSP (*Real Time Streaming Protocol*) foi especificado para controle na transferência de *streamings* de mídia armazenada. O protocolo ainda torna possível a transferência sob demanda de dados em tempo real, como áudio e vídeo. Serve estritamente para estabelecer e controlar um único ou vários fluxos sincronizados de mídias contínuas pertencentes a uma apresentação (RIBEIRO, 2006). Para CLEMENTE (2006), é importante salientar que o RTSP não é responsável pela transmissão do conteúdo propriamente dito. Para realizá-lo, o protocolo em questão relaciona-se com outros protocolos, normalmente o RTP.

O RTSP não define um protocolo de transporte a ser utilizado. De forma análoga ao que acontece com o FTP (*File Transfer Protocol*), esse protocolo abre uma conexão de controle paralela às conexões usadas para as transmissões de mídias e, por esse motivo, é conhecido como protocolo “fora da banda” (RIBEIRO, 2006).

O protocolo em questão pode ser comparado a um controle de videocassete para servidores multimídia na Internet. Através do RTSP o usuário controla o que deseja assistir, com comandos equivalentes a “play”, “pause”, “stop”, etc. Além disso, funciona através de trocas de mensagens, requisições e respostas entre o cliente e o servidor. Não existe o conceito de conexão RTSP, ao invés disso, o servidor identifica cada sessão através de um identificador único, para guardar os estados dos clientes (CLEMENTE, 2006).

4.2.2 RTP

Segundo CLEMENTE (2006), o protocolo RTP (*Real-time Transport Protocol*) foi especificado em 2003 através da RFC 3550. O RTP é um protocolo que provê serviços completos de entrega de dados em tempo real, tais como: identificação da codificação da mídia, numeração de sequência, marcação temporal e monitoramento de entrega.

Embora possa ser utilizado com outros protocolos de transporte de mais baixo nível, normalmente o RTP utiliza-se do protocolo UDP (*User Datagram Protocol*). Na RFC 3550, o RTP é divido em duas partes: o *Real-time Transport Protocol* (RTP) e o *RTP Control Protocol* (RTCP). A primeira parte é responsável pelo transporte dos dados, e a segunda tem a função de monitorar a qualidade do serviço e colher informações sobre os participantes de uma sessão.

De acordo com RIBEIRO (2006), o RTP especifica uma estrutura de transporte de dados de áudio e vídeo. Foi projetado para ser uma extensão do protocolo de transporte UDP, introduzindo marcas de tempo e números de sequência, que são essenciais para o sincronismo, a ordenação e a identificação de perdas de pacotes.

É importante ressaltar que o RTP não possui nenhum mecanismo que garanta a qualidade do serviço (QoS). Contudo, através do RTCP é possível obter informações necessárias para uma implementação de mecanismos de suporte à QoS (CLEMENTE, 2006). Para RIBEIRO (2006), a fim de fornecer QoS para uma aplicação, a Internet deve prover um mecanismo, tal como o RSVP (*Resource Reservation Protocol*), para que a aplicação possa reservar recursos da rede.

4.3 STREAMING SOBRE O PROTOCOLO HTTP

Segundo SANTOS (2010), no *streaming* tradicional na Internet a informação transmitida é codificada em pequenos pacotes, que podem ser transportados sobre UDP ou TCP. Caso sejam codificados em UDP, e uma vez que esses protocolos utilizam portas incomuns, muitos *firewalls* e *proxies* acabam por bloquear os pacotes, comprometendo a qualidade do serviço. Essa é uma das desvantagens deste tipo de protocolo, levando à necessidade de criação de novas alternativas melhor adaptadas às condições atuais da Internet. As soluções então adotadas baseiam-se na utilização do protocolo HTTP, amplamente utilizado na WWW (*World Wide Web*).

O *HTTP Streaming* é, portanto, uma forma adaptada de uso do protocolo HTTP e funciona contando com uma divisão do *stream* total numa sequência de pequenos *downloads* de arquivos baseados em HTTP, possibilitando utilizar os servidores *Web* já existentes ao invés de um servidor distribuidor de *streaming* para a difusão dos conteúdos (MARQUES *et al.*, 2012).

Dessa forma, pode-se perceber que o *HTTP Streaming* não é um *streaming* de dados propriamente dito. Trata-se de uma série de *downloads* de arquivos que são reproduzidos em um momento específico conforme uma determinada sequência, diferentemente do *streaming* tradicional, que conta com um fluxo contínuo de dados transmitidos.

A Figura 9 abaixo ilustra a arquitetura de funcionamento do *HTTP Streaming*.

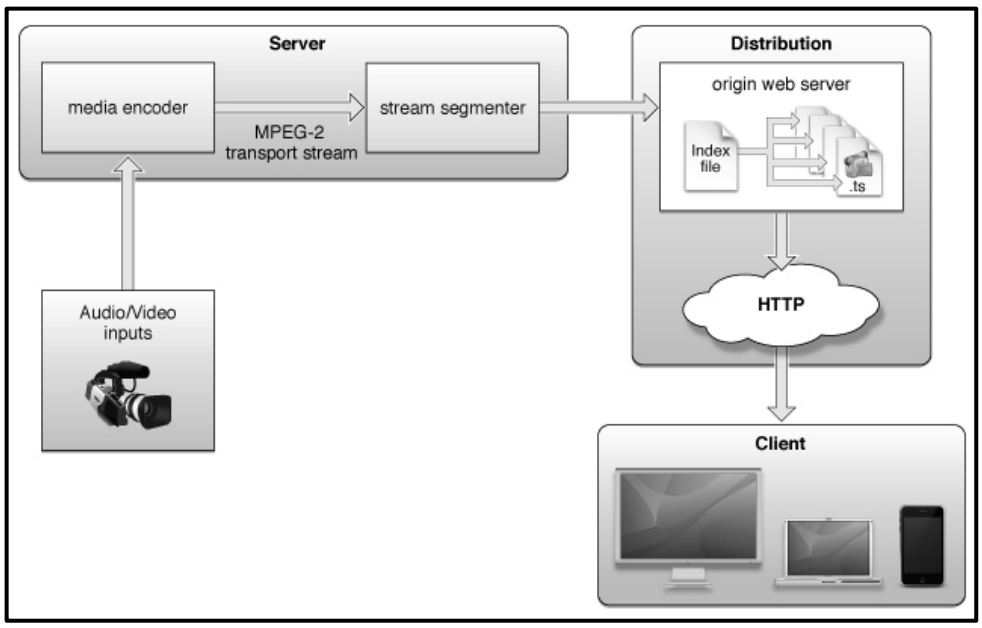

Figura 9 – Arquitetura HTTP Streaming.
Fonte: MARQUES *et al.*, 2012.

A partir da idéia básica de funcionamento e da arquitetura utilizada pelo *HTTP Streaming*, surgiram abordagens como o *download progressivo* e o *streaming adaptativo*.

Segundo MARQUES *et al.* (2012), quando se fala em *download progressivo*, refere-se a conteúdo multimídia disponibilizado por servidores de Internet HTTP tradicionais. Conteúdos transmitidos utilizando-se essa técnica recorrem a armazenamento no disco do usuário à medida que são recebidos para reprodução.

Para AZAMBUJA E PEREIRA (2009), o próprio nome já define bem o funcionamento da técnica de *download progressivo*: *download*, referente ao processo de cópia de um arquivo via HTTP para um computador à partir de um servidor *Web*; *progressivo*, referente à possibilidade de se iniciar a reprodução do conteúdo antes que a cópia completa seja finalizada.

Conforme define SANTOS (2010), o *download progressivo* consiste no *download* de arquivos disponibilizados por um servidor *Web* HTTP. Assim sendo, é suportado pela maioria das plataformas e aplicações de reprodução de conteúdos multimídia. Além disso, usuários que contam com o suporte para HTTP/1.1 podem também visualizar uma parte do conteúdo que ainda não esteja disponível no programa cliente. Isso é possível devido à troca de mensagens entre servidor e cliente, que envia requisições denominadas "*byte range requests*". Essa solução é adotada por vários *websites* de compartilhamento de vídeos, como o *YouTube*, *Vimeo*, *MySpace*, entre outros.

No entanto, o *download progressivo* também possui desvantagens. Uma delas se dá pelo fato de que, se o usuário escolher por reproduzir o conteúdo e depois de algum tempo decidir parar de assisti-lo, tanto ele como o fornecedor dos conteúdos terão desperdiçado largura de banda utilizada para *download* dos trechos do conteúdo que foram descartados (SANTOS, 2010).

Apesar de ser uma solução bastante simples e com baixo custo de implementação, o fato de copiar totalmente o conteúdo para a máquina do usuário, exatamente como acontece com um *download* de arquivo qualquer, torna o *download progressivo* relativamente inseguro, uma vez que existem diversas formas extremamente simples de se obter, copiar e distribuir ilegalmente o conteúdo (AZAMBUJA E PEREIRA, 2009).

Outra desvantagem do download progressivo é a falta de suporte a serviços de mídia ao vivo, visto que os arquivos precisam estar armazenados em um servidor para que o cliente possa baixá-los e reproduzi-los. Além disso, essa abordagem não leva em consideração as condições do ambiente de rede sendo utilizado.

De acordo com STOCKHAMMER (2011), o *streaming* sobre o protocolo HTTP deve estar o mais alinhado possível com a abordagem de *download progressivo*, visto que esta possui grande aceitação, mas considerar as deficiências desse modelo, aprimorando assim a qualidade do serviço percebida pelos usuários.

Em face disso, a 3GPP (*3rd Generation Partnership Project*) identificou a necessidade de fornecer uma especificação para uma distribuição escalável de vídeo e uma solução flexível para as redes móveis. Assim, tomou a iniciativa de especificar uma solução adaptativa para *HTTP Streaming*, o 3GPP-AHS (*Adaptive HTTP Streaming*).

Depois de especificar sua solução de *streaming* adaptativo, a 3GPP procurou o alinhamento com outros fóruns e organizações que trabalham na área de distribuição de vídeo, sobretudo o OIPF (*Open IPTV Forum*) e o MPEG (*Moving Picture Experts Group*). Breves descrições a despeito das soluções propostas por algumas instituições especializados no contexto de conteúdo multimídia são encontradas a seguir.

4.3.1 Adobe Dynamic HTTP Streaming

O *Adobe Dynamic HTTP Streaming*, segundo a ADOBE SYSTEMS INC. (2011), suporta tanto o *streaming* sob demanda quanto o *streaming* de vídeo ao vivo. No entanto, o processo de preparação do conteúdo para cada tipo é ligeiramente diferente, de forma que o conteúdo de *streaming* demandado por requisições do cliente é preparado por uma etapa de codificação que produz fragmentos de arquivo de vídeo, juntamente com um arquivo de manifesto; e o *streaming* ao vivo realiza uma fragmentação em tempo real e necessita de um servidor de empacotamento que irá encapsular e proteger um *stream* proveniente de um codificador que suporta o formato em questão.

4.3.2 Microsoft Smooth Streaming

O *Microsoft Smooth Streaming*, de acordo com a MICROSOFT CORP. (2009), utiliza o conceito de entrega de pequenos fragmentos de conteúdo (normalmente 2 segundos de vídeo). Uma verificação se encarrega de certificar se cada um deles foi recebido em momento oportuno e retorna o nível de qualidade esperado. Caso o fragmento recebido não atenda aos requisitos, o fragmento seguinte será entregue a um nível de qualidade diferente (inferior ou superior), dependendo da mensuração da expectativa para a reprodução do conteúdo multimídia. Dessa maneira, faz-se necessário que o conteúdo seja codificado em vários níveis de qualidade para, assim, ser entregue por um servidor de origem. Uma vez que esse servidor recebe uma solicitação de mídia, criará dinamicamente fragmentos virtuais a partir dos arquivos de vídeo e entregará o conteúdo adequado a cada usuário.

4.3.3 Apple HTTP Live Streaming

O sistema *Apple HTTP Live Streaming*, apresentado em APPLE INC. (2010), presume que o servidor deve dividir o fluxo em arquivos de mídia individuais cuja duração seja inferior ou igual a um valor alvo constante. O *stream* deve ser dividido em pontos que possibilitem decodificar de maneira eficaz os arquivos de mídia individuais. Normalmente, a divisão do fluxo ocorre nos limites dos quadros. O servidor, ainda, deve criar um arquivo de reprodução (*playlist*), que deve identificar um arquivo de mídia como um segmento da apresentação global. Um arquivo que contém o vídeo deve ter pelo menos um quadro-chave e informações suficientes para inicializar o decodificador de vídeo.

4.3.4 3GPP Adaptive HTTP Streaming

Segundo 3GPP (2011), *3GPP Adaptive HTTP Streaming* descreve uma sessão de *streaming* por uma descrição de apresentação de mídia (MPD - *Media Presentation Description*), que é responsável por estabelecer todo o cronograma e a disposição com que os segmentos de mídia serão executados, sendo assim considerada crucial para a adaptabilidade do *streaming* de conteúdo multimídia. O processo de fragmentação do conteúdo multimídia a ser reproduzido se dá pela divisão presente na apresentação de mídia, cuja localização normalmente é dada pelo lado do servidor. Essa apresentação é composta por períodos, representações e segmentos do conteúdo. Os períodos de uma apresentação delimitam os conteúdos que serão transmitidos, permitindo a inserção de anúncios publicitários, por exemplo. As representações são alternativas de escolha do conteúdo de mídia ou um subconjunto destes diferenciados por características que variam desde a resolução até o idioma de preferência. Assim, são organizadas de maneira que representações contidas em um mesmo agrupamento são alternativas entre si. Os segmentos de conteúdo de mídia, por sua vez, são elementos que podem ser unicamente referenciados por uma URL (*Uniform Resource Locator*), via protocolo HTTP.

Mantida essa organização hierárquica da apresentação de conteúdo, criou-se o conceito de descrição de apresentação de mídia (MPD), que normalmente é um documento declarativo, no formato XML (*Extensible Markup Language*), bem formatado que mapeia de maneira objetiva todos os períodos, representações e segmentos (juntamente com suas respectivas URLs) para cada conteúdo de mídia a ser transmitido e, posteriormente, executado (STOCKHAMMER, 2011).

4.3.4.1 Open IPTV Forum

O *Open IPTV Forum* (OIPF), como menciona 3GPP (2011), adotou a solução 3GPP para os segmentos de mídia e ampliou a descrição de apresentação de mídia (MPD) da 3GPP para os segmentos de mídia MPEG-TS (*MPEG Transport Stream*), que é um formato padrão para transmissão de dados multimídia proposto pelo MPEG (*Motion Picture Experts Group*). A segmentação do conteúdo de mídia segue o modelo proposto pela 3GPP, baseando-se na divisão de representações periodicamente alinhadas (OPEN IPTV FORUM, 2010).

4.3.4.2 MPEG Dynamic Adaptive Streaming over HTTP

O DASH (*Dynamic Adaptive Streaming over HTTP*) é uma tecnologia recente, evoluída a partir do *3GPP Adaptive HTTP Streaming*, que visa abordar os pontos fracos de tecnologias já então utilizadas, aprimorando a qualidade de serviço percebida aos olhos dos usuários. O MPEG-DASH é resultado de uma atividade de padronização realizada por um subgrupo da ISO, o MPEG, e oferece suporte a vários serviços, tais como *streaming* sob demanda, TV linear e visualização *time-shift* (STOCKHAMMER, 2011).

4.4 HTTP STREAMING ADAPTATIVO

Até recentemente, soluções de *streaming* eram caracterizadas por protocolos específicos, o que requer infraestrutura especifica e configurações de redes especiais relativas a *proxies*, *firewalls* e NATs. Hoje, observa-se uma gravitação das plataformas de comunicação para o HTTP, realizando os devidos controles operacionais no nível de software ao invés de hardware. A vantagem dessa abordagem é que a infraestrutura HTTP é baratíssima, integra-se facilmente com *proxies*, *firewalls* e NATs, além de aproveitar *caches*. Essas simplicidades são particularmente interessantes para redes de distribuição de conteúdo (CDNs – *Content Distribution Networks*), que efetivamente promovem as evoluções técnicas nessa área (MONTEIRO E SANTOS, 2012).

O *streaming* adaptativo consiste em um método de transmissão híbrido que tem funcionamento semelhante ao *streaming* tradicional, mas baseando-se no *download progressivo* sobre o protocolo HTTP. Em vez de transmitir apenas um único fluxo de dados completo, transmite vários segmentos, os quais, quando reproduzidos ordenadamente, forma o conteúdo multimídia completo.

Em uma implementação típica de *streaming* adaptativo, o conteúdo multimídia é dividido em muitos segmentos com duração curta, normalmente de 2 a 10 segundos, e

depois de codificados são guardados e disponibilizados por um servidor *Web* HTTP. A aplicação cliente requisita os segmentos ao servidor e descarrega-os utilizando a lógica de operação do *download progressivo*. À medida que os segmentos vão sendo recebidos pelo cliente, os mesmos são ordenados linearmente para reprodução. Isto só é possível porque os segmentos são codificados cuidadosamente, sem que exista espaçamento ou sobreposição, possibilitando que, ao serem reproduzidos ordenadamente, formem um fluxo contínuo.

A característica de adaptabilidade se dá quando, para o mesmo conteúdo de áudio ou vídeo, existem múltiplas codificações com taxas de transferência de *bits* (*bit rates*) diferentes, isto é, com qualidades distintas. Isso origina, consequentemente, arquivos com tamanhos diferentes. Em ambientes nos quais a largura de banda é limitada, revela-se necessária uma solução que possibilite continuar a visualizar o conteúdo, mesmo que em qualidade inferior. Dessa forma, a aplicação cliente poderá optar por diminuir ou aumentar a qualidade do conteúdo durante a reprodução do mesmo, equilibrando de um lado a fluidez da aplicação e do outro a qualidade do serviço.

O *streaming* adaptativo oferece aos provedores de conteúdo algumas vantagens, tais como: uma implementação mais barata, podendo utilizar-se de *caches*/*proxies* HTTP genéricos e sem a necessidade de servidores especializados; maior escalabilidade e alcance, uma vez que se adapta às más condições de rede; possibilidade de cada usuário receber os conteúdos com a qualidade adequada à sua conexão, não prejudicando outros usuários que possuam melhor qualidade de conexão (SANTOS, 2010).

Segundo MONTEIRO E SANTOS (2012), após o surgimento de diversas plataformas proprietárias de streaming adaptativo sobre HTTP, o grupo ISO MPEG definiu o novo padrão *Dynamic Adaptive Streaming over HTTP* (DASH), que já conta com implementações livres publicadas pela Universidade de Klagenfurt, na Áustria.

5 APLICABILIDADE DO HTTP STREAMING NA TV DIGITAL

Os estudos apresentados mostram-se bastante evoluídos na tentativa de resolver, na camada de software, gargalos infraestruturais que são encontrados com grande facilidade atualmente. A decisão de usar um protocolo considerado mais simples, mais bem suportado na Web e que conta com uma infraestrutura mais barata e de fácil manutenabilidade, deu prosseguimento ao aumento da escalabilidade, aprimoramento da qualidade do serviço oferecida aos usuários e disseminação de conteúdos multimídia pela Internet.

No entanto, esses serviços ainda encontram-se bastante limitados à *Web*. Novas tecnologias, como a TV Digital e o IPTV, não contam com essas novas técnicas. Em face disso, o principal objetivo deste trabalho é oferecer uma proposta de solução para a TV Digital, por intermédio da linguagem NCL e do middleware Ginga, de forma que possa ser estendido também para IPTV.

Isso se faz possível através da utilização dos próprios recursos com os quais conta a NCL (versão 3.0). Um elemento importante para a implementação e funcionamento da proposta apresentada, que não foi mencionado no capítulo 4 e será descrito a seguir, cabe-se do *switch*. Outros elementos essenciais para operacionalizar o *streaming* adaptativo em terminais de TV Digital e IPTV são: conectores, elos e contextos.

Como descrito anteriormente, os contextos são capazes de agrupar objetos de mídia e elos e, além disso, favorecem a organização e compreensão dos segmentos da aplicação NCL. Os contextos são muito importantes para realizar a chamada adaptação do conteúdo, que permite que o conteúdo multimídia seja exibido levando-se em consideração informações do usuário, o dispositivo que está exibindo a informação ou até mesmo o local onde se encontra o telespectador. Se uma aplicação contém intervalos comerciais com anúncios publicitários, por exemplo, o programador pode fazer com que essas propagandas sejam direcionadas ao público alvo devido. Através de um *switch* e de contextos, o criador da aplicação pode fazer com que seja exibida uma propaganda de cerveja, caso um adulto esteja assistindo ao conteúdo oferecido, ou então um comercial de refrigerante, caso o telespectador seja uma criança. Esse é apenas um exemplo de uma vasta gama de adaptações que podem ser criadas.

Outra característica importante para que a adaptabilidade do *HTTP Streaming* vigore, é a possibilidade de que um objeto de mídia (vídeo, áudio, imagens, texto, etc) seja inserido em uma aplicação NCL através de uma URL, que será atribuída ao atributo *src* do elemento *media*.

A Figura 10 mostra como um objeto de mídia pode ser referenciado através de um URL.

```
<media id="choro" src="http://www.telemidia.puc-rio.br/choro.mp3"/>
```

Figura 10 – Elemento <*media*> e URL no atributo *src*.

De maneira análoga, o objeto de mídia pode referenciar também os segmentos do objeto de mídia, por exemplo um vídeo, que se queira reproduzir através de *HTTP Streaming* adaptativo. Por meio dos contextos, pode-se separar e selecionar as diferentes codificações do conteúdo mediante algumas variáveis de ambiente que podem ser encontradas no *middleware* Ginga, isto é, consegue-se criar vários contextos de um mesmo conteúdo segmentado e codificado com qualidades diferentes e que serão selecionados de acordo com as condições da rede e capacidade de processamento do receptor, que são monitoradas com variáveis mantidas pelo próprio *middleware*.

Essas variáveis de ambiente são as mesmas utilizadas para a realização de adaptação de conteúdo. No caso particular do *streaming* adaptativo, uma dessas variáveis pode chamar mais a atenção e ser mais utilizada do que outras: a "*system.returnBitRate(i)*", que retorna qual a taxa do canal interativo (i) em *Kbps*. Através dessa informação, pode-se adaptar o *streaming* selecionando os contextos mais adequados à medida que as taxas de retorno do canal variam. Além disso, é interessante que se considerem outros parâmetros, tais quais a capacidade do processamento (desempenho da CPU) e o espaço de memória, sabendo que os arquivos correspondentes aos segmentos possuem tamanhos diferentes e serão descarregados no receptor por intermédio da metodologia do *download progressivo*.

Assim sendo, simultaneamente à reprodução do conteúdo multimídia contido na aplicação NCL pode ser realizada a adaptabilidade do *streaming* do mesmo, sobre o protocolo HTTP. Por intermédio do elemento *switch*, que será descrito na subseção a seguir, seleciona-se o contexto que melhor se adequa às condições encontradas no ambiente de execução na medida em que a reprodução progride.

5.1 ELEMENTO *SWITCH*

Antes mesmo de descrever os detalhes do elemento *switch*, é necessário conhecer como são definidas as regras as quais o *switch* irá avaliar.

De acordo com SOARES E BARBOSA (2012), as regras usadas em uma aplicação NCL são definidas no elemento *ruleBase*, que deve estar localizado no cabeçalho do documento. Cada regra possui um atributo *var*, que faz referência a uma propriedade do objeto *settings* (<media type="*application/x-ncl-settings*">), um operador de comparação e um valor, conforme ilustra a Figura 11.

```
<ruleBase>
   <rule id="rLegendaLigada" var="legenda"
                                    comparator="eq" value="ligada" />
   <rule id="rLegendaDesligada" var="legenda"
                               comparator="eq" value="desligada" />
</ruleBase>
```

Figura 11 – Definição de uma base de regras.
Fonte: SOARES E BARBOSA, 2012.

Um *switch* é uma composição contendo nós (objetos de mídia, contextos ou outros *switches*) alternativos. A decisão sobre qual nó será selecionado é dado por regras de mapeamento, definidas através de elementos *bindRule*. As regras serão avaliadas na ordem em que foram definidas e a primeira regra avaliada como verdadeira terá seu nó correspondente selecionado. Pode-se definir um nó que será selecionado por *default*, no caso de nenhuma regra ser satisfeita, através do elemento *defaultComponent* (SOARES E BARBOSA, 2012).

A Figura 12 exemplifica como o *switch* é definido juntamente com as regras a serem avaliadas.

```
<ruleBase>
   <rule id="rEn" var="system.language" comparator="eq" value="en" />
   <rule id="rPt" var="system.language" comparator="eq"  value="pt" />
</ruleBase>
                                                    ... trecho da seção <head>
<switch id="switchAudioIdioma">
   <bindRule rule="rEn" constituent="audioEn" />
   <bindRule rule="rPt" constituent="audioPt" />
   <defaultComponent component="audioPt" />
   <media id="audioEn" src="media/audioEn.mp3" descriptor="dAudio1" />
   <media id="audioPt" src="media/audioPt.mp3" descriptor="dAudio1" />
</switch>
                                                    ... trecho da seção <body>
```

Figura 12 – Regras e switch que seleciona o áudio conforme o idioma em vigor
Fonte: SOARES E BARBOSA, 2012.

5.2 IMPLEMENTAÇÃO

O desenvolvimento da implementação da solução proposta foi realizado utilizando a linguagem de programação C. Com o auxílio de uma biblioteca para manipulação de documentos XML, a TinyXML (http://www.grinninglizard.com/tinyxml), construiu-se um programa conversor de um documento NCL estendido proposto para um documento NCL 3.0.

Através de estudos realizados acerca da *Media Presentation Description* (MPD) utilizada na solução MPEG-DASH, notou-se a possibilidade de elaborar algo semelhante para TV Digital e IPTV, mediante utilização da NCL. Tanto a MPD como a NCL podem fazer referência a segmentos de mídia por meio de uma URL e, como visto anteriormente, ambas também são capazes de separar diferentes alternativas do conteúdo, de modo que, à medida que o mesmo é apresentado, qualidades diferentes consideradas mais adequadas para o momento em questão podem ser selecionadas. Uma diferença sensível é a possibilidade de construção de *templates* na MPD, indicando o número do segmento a ser buscado para a apresentação do conteúdo e, assim, compor a URL para que o arquivo HTTP correspondente seja descarregado e reproduzido. Dessa forma, o documento XML que compõe a MPD não precisa identificar todos os segmentos de mídia envolvidos no *streaming* adaptativo. Já em um documento NCL, todos os fragmentos da mídia devem estar definidos, visto que não existe um mecanismo na linguagem semelhante aos *templates* encontrados na abordagem anterior.

A idéia na criação do documento NCL que será convertido foi baseada no fato de que o programador de aplicações NCL não precisaria se preocupar com os relacionamentos entre os segmentos do conteúdo de mídia envolvidos no *streaming*. Através da utilização de alguns valores de atributos para elementos básicos da NCL, bastaria especificar os componentes da apresentação, tentando fazer com que o documento estendido proposto fosse o mais próximo possível de um documento NCL 3.0 convencional.

O primeiro caminho seguido, todavia, coube-se da utilização de um conector especial, que contava com um novo atributo informando que o conector em questão seria o componente que desencadearia a rotina de conversão. Além disso, o documento de entrada precisaria conter um *link* referenciando esse conector, indicando que o nó contido no atributo *component* de cada *bind* do elo seria o objeto de mídia utilizado no *streaming* adaptativo. Criava-se automaticamente, a partir daí, contextos contendo um conjunto de elos que faziam o relacionamento entre os segmentos da mídia indicada. Todos os elos desempenhavam papel de *onEnd* para a condição e *start* para a ação, visto que, ao terminar a execução de um segmento seria preciso iniciar a execução do próximo, assim sucessivamente até o último segmento correspondente. Cada contexto, então, conteria segmentos de mídia de qualidades distintas e seria selecionado mediante validação de regras envolvendo a taxa de retorno do canal de interatividade.

A partir do momento em que a conversão do documento foi realizada, percebeu-se que a adaptabilidade se encontrava restrita apenas à primeira vez em que o *switch* era executado. Depois que o primeiro segmento fosse reproduzido, seria necessária uma ação que fizesse com que as regras fossem novamente avaliadas para selecionar o contexto mais adequado. No entanto, o número do segmento que deveria ser executado na próxima

iteração seria perdido e, resolver esse impasse seria dispendioso o bastante para que uma nova modelagem do documento de entrada e, consequentemente, da conversão se mostrasse mais atraente e elegante.

Assim, alterando de maneira bem sutil a estrutura da linguagem NCL 3.0, chegou-se à solução atual. Através da definição de um objeto de mídia especial e da criação de regras e um *switch* para avaliá-las, é possível contornar o problema da perda do número do segmento encontrado na solução inicial.

A definição de uma mídia especial de *streaming* conta com a criação do valor "*video/segmented*" para o atributo *type* do elemento *media*. A Figura 13 exemplifica como uma mídia especial deve ser definida no documento NCL estendido de entrada do programa conversor.

```
<media id="goodMedia" type="video/segmented" descriptor="dVideo"
                src="http://www.exemplo.br/1024x768/TheSixthSense.mp4"/>
```

Figura 13 – Mídia especial em um documento NCL estendido.

Seguindo esse raciocínio, quando o programador define um objeto de mídia que será apresentado através de *streaming* adaptativo, o conversor detecta que essa mídia especial foi inserida no documento e, automaticamente, gera todas as ligações para cada um dos segmentos da mídia indicada. Além disso, ainda cria os contextos alternativos, separados e selecionados através de elementos *switch* internos, contendo cada um deles um objeto de mídia (segmento do conteúdo) com determinada qualidade. Dessa forma, acompanha-se o norte de que o *streaming* adaptativo utilizando NCL conta com um elemento *switch* para fazer a seleção do componente a ser exibido no decorrer da apresentação.

No entanto, outros componentes precisam ser identificados, como por exemplo as regras que serão validadas pelo elemento *switch*. O programador é responsável por definir essas regras e deve inseri-las normalmente como em um documento NCL 3.0. Depois, criar o elemento *switch* utilizando as regras criadas e indicando qual mídia segmentada será reproduzida quando uma das regras for avaliada como verdadeira.

A Figura 14 ilustra como um elemento *switch* deve ser definido no documento de entrada.

```
<ruleBase>
    <rule id="rGood" var="system.returnBitRate(1)" comparator="gte" value="200"/>
    <rule id="rMedium" var="system.returnBitRate(1)" comparator="gte" value="100"/>
    <rule id="rBad" var="system.returnBitRate(1)" comparator="lt" value="100"/>
</ruleBase>
```

cabeçalho do documento

```
<switch id="selectMedia">
    <bindRule constituent="goodMedia" rule="rGood"/>
    <bindRule constituent="mediumMedia" rule="rMedium"/>
    <bindRule constituent="badMedia" rule="rBad"/>
    <media id="goodMedia" type="video/segmented" descriptor="dVideo"
                    src="http://www.exemplo.br/1024x768/TheSixthSense.mp4"/>
    <media id="mediumMedia" type="video/segmented" descriptor="dVideo"
                    src="http://www.exemplo.br/800x600/TheSixthSense.mp4"/>
    <media id="badMedia" type="video/segmented" descriptor="dVideo"
                    src="http://www.exemplo.br/640x480/TheSixthSense.mp4"/>
</switch>
```

corpo do documento

Figura 14 – Switch definido no documento de entrada.

Após a definição dos componentes presentes no documento de entrada, o conversor se encarrega de localizar, no diretório informado no cabeçalho do programa, todos os segmentos associados à mídia em questão. O diretório deve conter apenas os arquivos que representam os segmentos da mídia escolhida, sendo que cada segmento (exceto o último) deve ter a mesma duração, preferencialmente 10 segundos. Outra restrição cabe-se do fato de que os arquivos dos segmentos devem conter o mesmo nome e extensão do arquivo da mídia completa, terminando com a identificação numérica sequencial de 4 (quatro) dígitos, ou seja, se a mídia cujo arquivo é “TheSixthSense.mp4” for dividida em 10 segmentos de mesma duração, os arquivos dos segmentos deverão manter a mesma extensão do arquivo da mídia original, neste caso “.mp4”, e ser nomeados como “TheSixthSense0001”, “TheSixthSense0002”, assim por diante até o último segmento, que será “TheSixthSense0010”. A segmentação do conteúdo, dessa forma, é responsabilidade do programador.

No caso de realização de um *streaming* adaptativo ao vivo, como os segmentos do conteúdo ainda não foram gerados, o conversor não vai localizar quaisquer arquivos representando-os. Assim sendo, o programador deverá estimar antecipadamente uma quantia de segmentos que serão gerados com a produção do conteúdo. O valor dessa estimativa deve ser inserido no cabeçalho do programa conversor, atribuído à variável que contabiliza a quantidade de segmentos.

Depois de identificados todos os segmentos da mídia, devidamente codificado em diferentes qualidades, o programa conversor insere automaticamente todas as mídias no documento NCL 3.0 de saída, dentro de cada contexto específico. Então, se uma mídia foi dividida em 10 segmentos e codificada em 3 qualidades distintas, o documento NCL 3.0 de

saída conterá 30 mídias definidas, sendo que serão agrupadas de 10 em 10 e distribuídas nos 3 contextos criados.

Além disso, o documento de saída contará com um *switch* para cada contexto criado, avaliando o número do próximo segmento a ser reproduzido. Dessa forma, a seleção dos segmentos das mídias definidas irá oscilar entre os contextos gerados, respeitando as regras especificadas pelo programador. Garante-se, assim, a adaptabilidade da solução proposta.

É importante salientar que a solução proposta não suporta, na mídia especial, a criação de interfaces com o intuito de definir relacionamentos entre a mesma e outros nós de mídia. Se assim mesmo, no documento de entrada, existirem âncoras na mídia, as mesmas, juntamente com os relacionamentos correspondentes, serão ignoradas. Tal restrição reflete a decisão de projeto de não incluir interatividade ou sincronismo neste trabalho.

As Figuras 15 e 16 exemplificam, respectivamente, um documento NCL estendido de entrada e um documento NCL 3.0 correspondente de saída, após convertido.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <ncl id="exemplo_entrada" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
3        <head>
4           <ruleBase>
5              <rule id="rGood" var="system.returnBitRate (0)" comparator="gte" value="200"/>
6              <rule id="rMedium" var="system.returnBitRate (0)" comparator="gte" value="100"/>
7              <rule id="rBad" var="system.returnBitRate (0)" comparator="lt" value="100"/>
8           </ruleBase>
9           <regionBase>
10             <region id="displayArea" height="100%" width="100%"/>
11          </regionBase>
12          <descriptorBase>
13             <descriptor id="dVideo" region="displayArea"/>
14          </descriptorBase>
15       </head>
16
17       <body>
18          <port id="pStart" component="selectMedia"/>
19          <switch id="selectMedia">
20             <bindRule constituent="goodMedia" rule="rGood"/>
21             <bindRule constituent="mediumMedia" rule="rMedium"/>
22             <bindRule constituent="badMedia" rule="rBad"/>
23             <media id="goodMedia" type="video/segmented" descriptor="dVideo"
24                                    src="http://exemplo.br/1024x768/TheSixthSense.mp4"/>
25             <media id="mediumMedia" type="video/segmented" descriptor="dVideo"
26                                    src="http://exemplo.br/800x600/TheSixthSense.mp4"/>
27             <media id="badMedia" type="video/segmented" descriptor="dVideo"
28                                    src="http://exemplo.br/640x480/TheSixthSense.mp4"/>
29          </switch>
30       </body>
31 </ncl>
```

Figura 15 – Documento NCL estendido de entrada

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ncl id="exemplo_saida" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
    <head>
        <ruleBase>
            <rule id="rGood" var="system.returnBitRate (0)" comparator="gte" value="200" />
            <rule id="rMedium" var="system.returnBitRate (0)" comparator="gte" value="100" />
            <rule id="rBad" var="system.returnBitRate (0)" comparator="lt" value="100" />
            <rule id="rSegNum1" var="segnum" comparator="eq" value="1" />
            <rule id="rSegNum2" var="segnum" comparator="eq" value="2" />
            <rule id="rSegNum3" var="segnum" comparator="eq" value="3" />
        </ruleBase>
        <regionBase>
            <region id="displayArea" height="100%" width="100%" />
        </regionBase>
        <descriptorBase>
            <descriptor id="dVideo" region="displayArea" />
        </descriptorBase>
        <connectorBase>
            <causalConnector id="onEndSetStopStart">
                <connectorParam name="num" />
                <simpleCondition role="onEnd" />
                <compoundAction operator="seq">
                    <simpleAction role="set" value="$num" />
                    <simpleAction role="stop" />
                    <simpleAction role="start" />
                </compoundAction>
            </causalConnector>
        </connectorBase>
    </head>
    <body>
        <port id="pStart" component="selectMedia" />

        <media id="globalSegNum" type="application/x-ginga-settings">
            <property name="segnum" value="1" />
        </media>

        <switch id="selectMedia">
            <bindRule constituent="goodMediaSelection" rule="rGood" />
            <bindRule constituent="mediumMediaSelection" rule="rMedium" />
            <bindRule constituent="badMediaSelection" rule="rBad" />

            <switch id="goodMediaSelection">
                <bindRule constituent="goodMediaContext1" rule="rSegNum1" />
                <bindRule constituent="goodMediaContext2" rule="rSegNum2" />
                <bindRule constituent="goodMediaContext3" rule="rSegNum3" />

                <context id="goodMediaContext1">
                    <port id="goodMediaContextPort1" component="goodMedia1" />
                    <media id="goodMedia1" descriptor="dVideo"
                                src="http://exemplo.br/1024x768/TheSixthSense0001.mp4" />
                    <link xconnector="onEndSetStopStart">
                        <bind role="onEnd" component="goodMedia1" />
                        <bind role="set" component="globalSegNum">
                            <bindParam name="num" value="2" />
                        </bind>
                        <bind role="stop" component="selectMedia" />
                        <bind role="start" component="selectMedia" />
                    </link>
                </context>
```

```
60                  <context id="goodMediaContext2">
61                      <port id="goodMediaContextPort2" component="goodMedia2" />
62                      <media id="goodMedia2" descriptor="dVideo"
63                                  src="http://exemplo.br/1024x768/TheSixthSense0002.mp4" />
64                      <link xconnector="onEndSetStopStart">
65                          <bind role="onEnd" component="goodMedia2" />
66                          <bind role="set" component="globalSegNum">
67                              <bindParam name="num" value="3" />
68                          </bind>
69                          <bind role="stop" component="selectMedia" />
70                          <bind role="start" component="selectMedia" />
71                      </link>
72                  </context>
73
74                  <context id="goodMediaContext3">
75                      <port id="goodMediaContextPort3" component="goodMedia3" />
76                      <media id="goodMedia3" descriptor="dVideo"
77                                  src="http://exemplo.br/1024x768/TheSixthSense0003.mp4" />
78                      <link xconnector="onEndSetStopStart">
79                          <bind role="onEnd" component="goodMedia3" />
80                          <bind role="set" component="globalSegNum">
81                              <bindParam name="num" value="4" />
82                          </bind>
83                          <bind role="stop" component="selectMedia" />
84                      </link>
85                  </context>
86              </switch>
87
88              <switch id="mediumMediaSelection">
89                  <bindRule constituent="mediumMediaContext1" rule="rSegNum1" />
90                  <bindRule constituent="mediumMediaContext2" rule="rSegNum2" />
91                  <bindRule constituent="mediumMediaContext3" rule="rSegNum3" />
92
93                  <context id="mediumMediaContext1">
94                      <port id="mediumMediaContextPort1" component="mediumMedia1" />
95                      <media id="mediumMedia1" descriptor="dVideo"
96                                  src="http://exemplo.br/800x600/TheSixthSense0001.mp4" />
97                      <link xconnector="onEndSetStopStart">
98                          <bind role="onEnd" component="mediumMedia1" />
99                          <bind role="set" component="globalSegNum">
100                             <bindParam name="num" value="2" />
101                         </bind>
102                         <bind role="stop" component="selectMedia" />
103                         <bind role="start" component="selectMedia" />
104                     </link>
105                 </context>
106
107                 <context id="mediumMediaContext2">
108                     <port id="mediumMediaContextPort2" component="mediumMedia2" />
109                     <media id="mediumMedia2" descriptor="dVideo"
110                                 src="http://exemplo.br/800x600/TheSixthSense0002.mp4" />
111                     <link xconnector="onEndSetStopStart">
112                         <bind role="onEnd" component="mediumMedia2" />
113                         <bind role="set" component="globalSegNum">
114                             <bindParam name="num" value="3" />
115                         </bind>
116                         <bind role="stop" component="selectMedia" />
117                         <bind role="start" component="selectMedia" />
118                     </link>
119                 </context>
```

```
120                 <context id="mediumMediaContext3">
121                     <port id="mediumMediaContextPort3" component="mediumMedia3" />
122                     <media id="mediumMedia3" descriptor="dVideo"
123                                 src="http://exemplo.br/800x600/TheSixthSense0003.mp4" />
124                     <link xconnector="onEndSetStopStart">
125                         <bind role="onEnd" component="mediumMedia3" />
126                         <bind role="set" component="globalSegNum">
127                             <bindParam name="num" value="4" />
128                         </bind>
129                         <bind role="stop" component="selectMedia" />
130                     </link>
131                 </context>
132             </switch>
133 
134             <switch id="badMediaSelection">
135                 <bindRule constituent="badMediaContext1" rule="rSegNum1" />
136                 <bindRule constituent="badMediaContext2" rule="rSegNum2" />
137                 <bindRule constituent="badMediaContext3" rule="rSegNum3" />
138 
139                 <context id="badMediaContext1">
140                     <port id="badMediaContextPort1" component="badMedia1" />
141                     <media id="badMedia1" descriptor="dVideo"
142                                 src="http://exemplo.br/640x480/TheSixthSense0001.mp4" />
143                     <link xconnector="onEndSetStopStart">
144                         <bind role="onEnd" component="badMedia1" />
145                         <bind role="set" component="globalSegNum">
146                             <bindParam name="num" value="2" />
147                         </bind>
148                         <bind role="stop" component="selectMedia" />
149                         <bind role="start" component="selectMedia" />
150                     </link>
151                 </context>
152 
153                 <context id="badMediaContext2">
154                     <port id="badMediaContextPort2" component="badMedia2" />
155                     <media id="badMedia2" descriptor="dVideo"
156                                 src="http://exemplo.br/640x480/TheSixthSense0002.mp4" />
157                     <link xconnector="onEndSetStopStart">
158                         <bind role="onEnd" component="badMedia2" />
159                         <bind role="set" component="globalSegNum">
160                             <bindParam name="num" value="3" />
161                         </bind>
162                         <bind role="stop" component="selectMedia" />
163                         <bind role="start" component="selectMedia" />
164                     </link>
165                 </context>
166 
167                 <context id="badMediaContext3">
168                     <port id="badMediaContextPort3" component="badMedia3" />
169                     <media id="badMedia3" descriptor="dVideo"
170                                 src="http://exemplo.br/640x480/TheSixthSense0003.mp4" />
171                     <link xconnector="onEndSetStopStart">
172                         <bind role="onEnd" component="badMedia3" />
173                         <bind role="set" component="globalSegNum">
174                             <bindParam name="num" value="4" />
175                         </bind>
176                         <bind role="stop" component="selectMedia" />
177                     </link>
178                 </context>
179 
180             </switch>
181         </switch>
182     </body>
183 </ncl>
```

Figura 16 – Documento NCL 3.0 de saída

Como pode ser percebido na Figura 15, o criador da aplicação NCL com *streaming* adaptativo apenas irá se preocupar com a definição das regras e do elemento *switch* que irá avaliá-las, bem como a definição das mídias segmentadas que serão utilizadas no processo de conversão. Entre as linhas 5 e 7 e na linha 19 do documento de entrada exemplificado, são definidas as regras e o elemento *switch*, respectivamente. Dentro do último, a identificação das mídias especiais (com o valor "*vídeo/segmented*" para o atributo *type* do elemento *media*) que farão parte da apresentação da aplicação, como observado nas linhas 23, 25 e 27.

Através da Figura 16, percebe-se que os elementos da linguagem presentes no documento de saída correspondem aos componentes especificados na NCL 3.0, não contendo nenhum valor, atributo ou outro elemento qualquer que irá impedir que o formatador NCL reproduza o conteúdo da aplicação. Da linha 5 à linha 10, encontram-se as regras que serão avaliadas, sendo que aquelas especificadas pelo programador no documento de entrada foram preservadas. Na linha 19, observa-se a definição do conector que será utilizado para atualizar a variável de contagem dos segmentos e, além disso, reiniciar o *switch* mais externo para que as regras sejam novamente analisadas. Na linha 33, percebe-se a criação de um elemento *media*, que corresponde à variável que armazena a informação sobre o número do segmento a ser reproduzido. A mesma é inicializada com o valor "1" e incrementada à medida que os segmentos de mídia são reproduzidos. Já na linha 37, encontra-se o *switch* externo, definido pelo próprio programador no documento NCL de entrada. Nas linhas 42, 88 e 134, podem ser observados os *switches* internos, responsáveis pela seleção do contexto mais adequado para a apresentação da mídia, considerando o número do segmento a ser executado. Assim, dentro de cada elemento *switch* existe um contexto envolvendo cada segmento da mídia. Ao final de cada contexto, após a mídia correspondente ter sido reproduzida, o valor da variável que armazena o número do segmento será incrementado e, depois, o *switch* externo será parado e iniciado novamente, como observado entre as linhas 51 e 58 do documento de saída, por exemplo. Com essa estratégia, garante-se que a cada segmento reproduzido ocorrerá uma avaliação acerca da melhor alternativa para a reprodução do próximo, conforme as regras definidas pelo programador no documento estendido.

5.3 PERFIL DE NCL NECESSÁRIO AO HTTP STREAMING

Seguindo listagem apresentada em SOARES E BARBOSA (2012, p. 142), o perfil da linguagem NCL estendida correspondente ao documento de entrada é composto pelos elementos: *ncl*, *head* e *body*, da área funcional *Structure*; *region* e *regionBase*, da área funcional *Layout*; *descriptor* e *descriptorBase*, da área funcional *Presentation Specification*;

media, da área funcional *Components*; *rule*, *ruleBase*, *bindRule* e *switch*, da área funcional *Presentation Control*; e *port*, da área funcional *Interfaces*.

A área funcional *Structure* é responsável por definir a estrutura básica de um documento NCL. A área funcional *Layout* especifica elementos e atributos que definem como os objetos serão apresentados em regiões na tela de um dispositivo de exibição. A área funcional *Presentation Specification* objetiva especificar as informações temporais e espaciais necessárias para apresentar cada componente de um documento, modeladas por objetos chamados descritores. A área funcional *Components* é responsável pela definição dos tipos básicos de objetos de mídia ou dos nós de composição que um documento NCL pode conter. A área funcional *Presentation Control* tem a função de especificar alternativas de conteúdo e alternativas de exibição para um documento. E, por fim, a área funcional *Interfaces* permite a definição de pontos de interface de nós que serão usados para a criação de elos (SAADE, 2003).

É importante lembrar que o elemento *media* utilizado no perfil NCL do documento de entrada é diferente daquele que compõe a NCL 3.0. O tipo MIME "*video/segmented*" definido através do atributo *type* faz com que o elemento *media* utilizado na solução proposta não seja encontrado sob normas da ABNT.

A partir do resultado do processo de conversão realizado pelo programa criado, pode-se elaborar uma aplicação contendo *streaming* adaptativo de conteúdo multimídia sem que seja preciso especificar outra versão da linguagem NCL ou criar *players* que reconheçam o documento de entrada proposto.

Os *Schemas XML* das áreas funcionais que compõem o perfil da linguagem NCL observada no documento de entrada, bem como o código-fonte do programa conversor criado, podem ser encontrados em anexo, ao final do trabalho.

5.4 COMPARATIVO COM A MPD MPEG-DASH

Traçando um comparativo entre o documento NCL 3.0 de saída e a MPD, utilizada na solução MPEG-DASH, as Figuras 17 e 18 mostram como seriam os documentos (equivalentes) de cada uma das abordagens. O exemplo contextualizado consta-se de um vídeo de 20 segundos, dividido em dois segmentos de 10 segundos cada um. Esse vídeo, ainda, possui duas versões, de qualidades distintas, que são alternativas entre si.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" type="static" mediaPresentationDuration="PT20S"
minBufferTime="PT1.5S" profiles="urn:mpeg:dash:profile:full:2011">

  <BaseURL>http://exemplo.br/</BaseURL>

  <Period start="PT0S" duration="PT20S">
    <AdaptationSet mimeType="video/mp4" codecs="avc1.4d0228">
      <ContentProtection schemeIdUri="urn:uuid:706D6953-656C-5244-4D48-656164657221"/>
      <Representation bandwidth="204800" id="1">
        <SegmentList timescale="1000" duration="10000">
            <SegmentURL media="800x600/TheSixthSense0001.mp4"/>
            <SegmentURL media="800x600/TheSixthSense0002.mp4"/>
        </SegmentList>
      </Representation>
      <Representation bandwidth="102400" id="2">
        <SegmentList timescale="1000" duration="10000">
            <SegmentURL media="640x480/TheSixthSense0001.mp4"/>
            <SegmentURL media="640x480/TheSixthSense0002.mp4"/>
        </SegmentList>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>
```

Figura 17 – MPD para HTTP Streaming adaptativo

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ncl id="exemplo_saida" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
    <head>
        <ruleBase>
            <rule id="rGood" var="system.returnBitRate (0)" comparator="gte" value="100" />
            <rule id="rBad" var="system.returnBitRate (0)" comparator="lt" value="100" />
            <rule id="rSegNum1" var="segnum" comparator="eq" value="1" />
            <rule id="rSegNum2" var="segnum" comparator="eq" value="2" />
        </ruleBase>
        <connectorBase>
            <causalConnector id="onEndSetStopStart">
                <connectorParam name="num" />
                <simpleCondition role="onEnd" />
                <compoundAction operator="seq">
                    <simpleAction role="set" value="$num" />
                    <simpleAction role="stop" />
                    <simpleAction role="start" />
                </compoundAction>
            </causalConnector>
        </connectorBase>
    </head>
    <body>
        <port id="pStart" component="selectMedia" />
        <media id="globalSegNum" type="application/x-ginga-settings">
            <property name="segnum" value="1" />
        </media>
        <switch id="selectMedia">
            <bindRule constituent="goodMediaSelection" rule="rGood" />
            <bindRule constituent="badMediaSelection" rule="rBad" />
            <switch id="goodMediaSelection">
                <bindRule constituent="goodMediaContext1" rule="rSegNum1" />
                <bindRule constituent="goodMediaContext2" rule="rSegNum2" />
                <context id="goodMediaContext1">
                    <port id="goodMediaContextPort1" component="goodMedia1" />
                    <media id="goodMedia1" src="http://exemplo.br/800x600/TheSixthSense0001.mp4" />
                    <link xconnector="onEndSetStopStart">
                        <bind role="onEnd" component="goodMedia1" />
                        <bind role="set" component="globalSegNum">
                            <bindParam name="num" value="2" />
                        </bind>
```

```
                        <bind role="stop" component="selectMedia" />
                        <bind role="start" component="selectMedia" />
                    </link>
                </context>
                <context id="goodMediaContext2">
                    <port id="goodMediaContextPort2" component="goodMedia2" />
                    <media id="goodMedia2" src="http://exemplo.br/800x600/TheSixthSense0002.mp4" />
                    <link xconnector="onEndSetStopStart">
                        <bind role="onEnd" component="goodMedia2" />
                        <bind role="set" component="globalSegNum">
                            <bindParam name="num" value="3" />
                        </bind>
                        <bind role="stop" component="selectMedia" />
                        <bind role="start" component="selectMedia" />
                    </link>
                </context>
            </switch>
            <switch id="badMediaSelection">
                <bindRule constituent="badMediaContext1" rule="rSegNum1" />
                <bindRule constituent="badMediaContext2" rule="rSegNum2" />
                <context id="badMediaContext1">
                    <port id="badMediaContextPort1" component="badMedia1" />
                    <media id="badMedia1" src="http://exemplo.br/640x480/TheSixthSense0001.mp4" />
                    <link xconnector="onEndSetStopStart">
                        <bind role="onEnd" component="badMedia1" />
                        <bind role="set" component="globalSegNum">
                            <bindParam name="num" value="2" />
                        </bind>
                        <bind role="stop" component="selectMedia" />
                        <bind role="start" component="selectMedia" />
                    </link>
                </context>
                <context id="badMediaContext2">
                    <port id="badMediaContextPort2" component="badMedia2" />
                    <media id="badMedia2" src="http://exemplo.br/640x480/TheSixthSense0002.mp4" />
                    <link xconnector="onEndSetStopStart">
                        <bind role="onEnd" component="badMedia2" />
                        <bind role="set" component="globalSegNum">
                            <bindParam name="num" value="3" />
                        </bind>
                        <bind role="stop" component="selectMedia" />
                        <bind role="start" component="selectMedia" />
                    </link>
                </context>
            </switch>
        </switch>
    </body>
</ncl>
```

Figura 18 – NCL para HTTP Streaming adaptativo

5.5 QUESTÕES DE PROJETO DO GINGA-NCL

Para que a solução proposta seja, de fato, uma contribuição para a melhoria das técnicas utilizadas atualmente para apresentação de conteúdo multimídia em TV Digital e IPTV, é necessário que algumas condições sejam atendidas. O mecanismo de pré-busca do Ginga mostra-se essencial para corroborar a eficiência e a qualidade do serviço oferecido através da solução indicada.

De acordo com RODRIGUES (2003), para que o usuário perceba uma boa qualidade em uma apresentação hipermídia, diversos parâmetros precisam ser controlados

pelo formatador. Dentre os parâmetros temporais destacam-se a latência para exibição dos objetos e a taxa para apresentação dos objetos de mídia contínua. A latência de exibição corresponde ao intervalo de tempo entre o instante no qual o conteúdo de um objeto deve ser exibido e o instante no qual o mesmo é efetivamente apresentado. As taxas de apresentação são características dos objetos de mídia contínua, e normalmente envolvem uma quantidade de amostras por segundo e uma resolução para cada uma das amostras.

A principal responsabilidade do mecanismo de pré-busca é procurar garantir que o conteúdo de cada objeto (não necessariamente o objeto inteiro) esteja preparado para a ferramenta de exibição no momento em que a mesma necessite exibi-lo, fazendo com que as latências de apresentação durante a execução dos documentos encontrem-se dentro dos limites aceitáveis.

Ainda segundo RODRIGUES (2003), o ponto de partida do compilador de pré-busca é percorrer a cadeia temporal principal do plano de execução e estimar o tempo despendido na preparação de cada um dos objetos a serem executados. Assim, o compilador deve consultar o conteúdo de cada objeto para tentar descobrir sua duração, além de tentar encontrar nos descritores correspondentes o percentual mínimo do conteúdo total que precisa estar disponível antes do início da apresentação e a latência máxima permitida para a exibição do objeto. O compilador de pré-busca deve dispor, ainda, de pelo menos três informações adicionais do contexto de exibição para estimar o tempo de busca, tais como: o retardo de propagação dos dados do local onde estão armazenados até o *buffer* da ferramenta, a taxa de transmissão disponível para transferência dos dados e o próprio retardo imposto pela ferramenta de exibição para sua criação e iniciação.

A outra função do compilador de pré-busca é determinar a ordem das preparações para os objetos de execução, para com isso realizar a construção efetiva do plano de pré-busca. Alguns aspectos importantes devem ser levados em consideração na construção do plano de pré-busca, visto que é importante que haja um planejamento na escolha dos tempos e na ordenação das requisições. Caso os tempos não sejam bem escolhidos, podem ser observadas duas situações: a busca pode ser iniciada tardiamente e o objeto não estar preparado no momento da sua exibição, ou a busca pode iniciar cedo demais e o objeto ter de permanecer armazenado na memória *cache* durante muito tempo, podendo, inclusive, ser descartado até o momento efetivo de sua exibição, gerando uma pré-busca inútil (RODRIGUES, 2003).

Para o caso da solução de *streaming* adaptativo proposta, a técnica de pré-busca deve ser ainda mais perspicaz. Como ao final de cada segmento de mídia reproduzido as regras definidas no documento NCL são novamente avaliadas, existe a possibilidade de, a cada avaliação, a execução estar relacionada a um *switch* interno diferente, isto é, pode

acontecer a seleção de uma qualidade de mídia diferente para a reprodução do próximo segmento a todo término da execução de outro.

Seria desperdício, por exemplo, iniciar requisições para preparação do conteúdo do quinto segmento de uma mídia de determinada qualidade no momento em que o terceiro segmento da mídia correspondente (segmentos de um mesmo *switch* interno) esteja em reprodução. Pode ser que, no momento da reprodução efetiva, o quinto segmento do conteúdo que será exibido corresponda a uma mídia com outra qualidade e, portanto, faça parte de outro agrupamento de mídias (*switch* interno). Dessa forma, o plano de pré-busca deve agir de maneira tal a abranger e tratar os diversos caminhos de execução possíveis.

Para contornar esse problema, conforme menciona RODRIGUES (2003), a estratégia de compilação da pré-busca deve levar em conta a escolha do momento para iniciar a preparação dos objetos contidos em cadeias temporais auxiliares, já que é importante que a latência seja minimizada, mesmo para objetos com exibições imprevisíveis. O ideal é que a estratégia consiga obter diretamente do plano de execução, ou junto ao contexto de exibição, a probabilidade para que cada uma das cadeias auxiliares tenha sua execução iniciada.

A partir do momento em que o ambiente no qual a aplicação NCL será executada atende aos requisitos mencionados, pode-se garantir que a solução proposta será utilizada com maior eficiência e, por conseguinte, proverá melhor qualidade de serviço aos telespectadores.

6 CONCLUSÃO

Técnicas de difusão de conteúdo multimídia na Internet vêm sendo cada vez mais estudadas e aprimoradas mediante a importância que esse tipo de serviço passou a incorporar no decorrer dos últimos anos. A escalabilidade dessas técnicas teve um aumento diretamente proporcional às exigências dos usuários e à qualidade do serviço que se procura oferecer.

A utilização de infraestruturas mais simples e mais baratas também impulsionou positivamente o surgimento de novas abordagens acerca da maneira pela qual o conteúdo multimídia é distribuído na rede. A adoção do protocolo HTTP, por exemplo, fez com que a arquitetura da rede de transmissão de conteúdo ficasse mais simples e, não obstante, ainda acarretou em uma notável melhora no serviço prestado. O *download progressivo*, ao que parece, representou um marco para as aplicações envolvidas nesse contexto. Não é por acaso que encontramos essa técnica atualmente bastante difundida na *Web*, com grandes adeptos como *YouTube*, *Vimeo*, entre outros.

Mas com a qualidade do serviço ainda não suficiente e/ou com o aumento das exigências dos usuários, percebeu-se a necessidade de criar soluções de transmissões de conteúdo multimídia que se adaptassem às alterações do ambiente de execução. Agora, mesmo que as condições da rede ou a capacidade de processamento sejam limitadores para a prestação de um serviço bem qualificado, um usuário pode assistir a um vídeo ou escutar uma música com fluidez, mesmo que para isso a qualidade da mídia oscile.

É fácil notar, assim, que as pesquisas evoluíram bastante no decorrer do tempo e, de acordo com a tendência dos acontecimentos, muitas outras surgirão e aprimorarão as técnicas que envolvem o contexto multimídia na Internet.

Outra característica perceptível cabe-se do surgimento de novos dispositivos que passaram a contar com recursos já presentes na *Web*. Isso faz com que, além de se fazer necessário o aprofundamento nas técnicas hoje já encontradas, ocorra a necessidade de fornecer esses serviços para usuários desses outros dispositivos, e com qualidade relativamente compatível com a que é encontrada nos setores onde a pesquisa se mostra mais avançada.

Em face disso, agregar valor às novas tecnologias inserindo funcionalidades que já são desenvolvidas em outros meios pode ser bastante interessante no ponto de vista de atração aos usuários. O *streaming* adaptativo para terminais de TV Digital e IPTV, por exemplo, possibilita que telespectadores possam assistir a conteúdos provenientes de servidores *Web* a qualquer instante sendo levados em consideração todos os gargalos infraestruturais que possam prejudicar a qualidade do serviço, fazendo com que esses problemas se tornem cada vez mais transparentes aos usuários.

A partir da varredura bibliográfica que foi feita e da proposta de solução apresentada para *streaming* adaptativo utilizando o protocolo HTTP, consegue-se notar que a TV Digital e o IPTV possuem grande potencial para agregar novas técnicas que estão sendo utilizadas em outros ambientes, principalmente a *Web*.

O objetivo do presente trabalho foi, de fato, apresentar uma alternativa de se fazer uma aplicação com a utilização do protocolo HTTP para *streaming* adaptativo utilizando os próprios recursos que a linguagem NCL e o *middleware* Ginga oferecem.

Com a solução proposta, o programador de aplicações NCL não precisa se preocupar com os relacionamentos entre os segmentos da mídia e com a criação dos contextos para seleção do conteúdo mais adequado. Isso é feito automaticamente através do conversor elaborado.

Sendo assim, pode-se considerar que o objetivo do trabalho foi suficientemente alcançado, apesar das limitações de recursos encontradas para desenvolvimento de uma solução mais eficiente e abrangente.

6.1 TRABALHOS FUTUROS

Para dar continuidade ao presente trabalho, pretende-se aprimorar a técnica de adaptabilidade adotada, passando a analisar mais variáveis que possam influenciar nas condições da rede e a capacidade de processamento do ambiente de execução. Um exemplo de trabalho futuro cabe-se da análise do *buffer* de recebimento dos segmentos à medida que são descarregados no receptor. Pode-se, através da quantidade de segmentos contidos no *buffer* em um momento específico, determinar que uma adaptação deve ser feita e, consequentemente, realizar uma troca de contexto, isto é, se a qualidade do conteúdo pode ser aumentada ou deve ser diminuída para o *download* dos próximos segmentos.

Além disso, pretende-se inserir a possibilidade de se criar interatividade ou sincronismo na construção de aplicações NCL que seguem o modelo proposto e melhorar a qualidade e eficiência da implementação, fazendo com que o processo de conversão fique mais rápido, mais flexível e que consiga agregar novas funcionalidades que possam vir a ser necessárias.

REFERÊNCIAS

3GPP. 3GPP Adaptive HTTP Streaming. Disponível em: https://labs.ericsson.com/apis/streaming-media/documentation-Adaptive-HTTPStreaming-DASH. Acesso em: 29 de novembro de 2012.

ADOBE SYSTEMS INC. Adobe Dynamic HTTP Streaming. Disponível em: http://www.adobe.com/products/httpdynamicstreaming. Acesso em: 26 de novembro de 2012.

ANTONACCI, Meire Juliana; RODRIGUES, Rogério Ferreira; SAADE, Débora C. Muchaluat; SOARES, Luiz Fernando Gomes. NCL: Uma linguagem declarativa para especificação de documentos hipermídia na Web. Rio de Janeiro, 2000.

APPLE INC. Apple HTTP Live Streaming. Disponível em: http://tools.ietf.org/id/draft-pantos-httplive-streaming-04.txt. Acesso em: 26 de novembro de 2012.

AZAMBUJA, Marcello de Lima; PEREIRA, Rafael Silva. Arquiteturas de Distribuição de Vídeos na Internet. Revista de Radiodifusão, v. 3, n. 03, setembro 2009.

BOLAÑO, César; VIEIRA, Vinícius Rodrigues. TV digital no Brasil e no mundo: estado da arte. Revista de Economía Política de las Tecnologías de la Información y Comunicación, v. 6, n. 2, maio/agosto 2004.

CAMPOS, Fernanda Cláudia Alves; CASTRO, Maria Clicia Stelling de. Manual de Monografia: Curso de Bacharelado em Ciência da Computação.

CLEMENTE, Ricardo Gomes. Uma solução de streaming de vídeo para celulares: conceitos, protocolos e aplicativo. 2006. Trabalho de Conclusão de Curso (Bacharelado em Computação) – Departamento de Eletrônica e de Computação, Universidade Federal do Rio de Janeiro.

FERNANDES, Jocimar. TV Digital Interativa. Vitória, 2006. Trabalho de Conclusão de Curso (Pós-graduação em Engenharia de Sistemas) – Escola Superior Aberta do Brasil.

KUROSE, James F.; ROSS, Keith W. Redes de computadores e a Internet: uma abordagem top-down. 5 ed. S.l.: Pearson, 2010.

MARQUES, André; BETTENCOURT, Raquel; FALCÃO, Joana. Internet Live Streaming. Lisboa, 2012.

MICROSOFT CORP. Microsoft IIS Smooth Streaming. Disponível em: http://www.iis.net/download/smoothstreaming. Acesso em: 26 de novembro de 2012.

MONTEIRO, Estêvão Chaves; SANTOS, Lucas Alberto S. Modernização dos Sistemas Audiovisuais do Serpro. 2012.

MONTEZ, Carlos; BECKER, Valdecir. TV Digital Interativa: Conceitos e Tecnologias. In: WebMidia e LA-Web 2004 – Joint Conference. Ribeirão Preto, 2004.

OPEN IPTV FORUM. OIPF HTTP Adaptive Streaming. Disponível em: http://wenku.baidu.com/view/4fe12e8a680203d8ce2f240c.html. Acesso em: 26 de novembro de 2012.

RATAMERO, Erick Martins. Tutorial sobre a linguagem de programação NCL (Nested Context Language). Niterói, 2007.

RIBEIRO, Cesar Henrique Pereira. Novas Propostas para Protocolos de Streaming. Niterói, 2006.

RODRIGUES, Rogério Ferreira. Formatação e Controle de Apresentações Hipermídia com Mecanismos de Adaptação Temporal. Rio de Janeiro, 2003. Tese (Doutorado em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

SAADE, Débora Christina Muchaluat. Relações em Linguagens de Autoria Hipermídia: Aumentando Reuso e Expressividade. Rio de Janeiro, 2003. Tese (Doutorado em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

SANTOS, José Fernando Pereira dos. Mobile Streaming: Qualidade de experiência. Porto, 2010. Dissertação (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) – Faculdade de Engenharia - Universidade do Porto.

SOARES, Luiz Fernando Gomes; BARBOSA, Simone Diniz Junqueira. Programando em NCL 3.0: Desenvolvimento de aplicações para o middleware Ginga. 2 ed. Rio de Janeiro, 2012.

STOCKHAMMER, Thomas. Dynamic Adaptive Streaming over HTTP: Standards and design principles. 2011.

TANENBAUM, Andrew S. Redes de computadores. 4 ed. S.l.: Campus, 2003.

TSCHÖKE, Clodoaldo. Criação de Streaming de Vídeo para Transmissão de Sinais de Vídeo em Tempo Real pela Internet. Blumenau, 2001. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau.

ZANCANARO, Airton; SANTOS, Paloma Maria; TODESCO, José Leomar. Ginga-J ou Ginga-NCL: características das linguagens de desenvolvimento de recursos interativos para a TV Digital. In: I Simpósio Internacional de Televisão Digital. Bauru, 2009.

APÊNDICE 1 – CÓDIGO-FONTE DO PROGRAMA CONVERSOR

```
/*****************************************
    UNIVERSIDADE FEDERAL DE JUIZ DE FORA
    INSTITUTO DE CIÊNCIAS EXATAS
    GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
    MONOGRAFIA FINAL EM COMPUTAÇÃO
    --- CONVERSOR NCL ---
    AUTOR: RAPHAEL NEVES ANDRÉ
*****************************************/

#include <stdio.h>
#include <tinyxml.h>
#include <stdlib.h>
#include <dirent.h>

#define DOCUMENTO_ENTRADA "(inserir nome e extensão do arquivo de entrada. Ex.
entrada.ncl)"

#define DOCUMENTO_SAIDA "(inserir nome e extensão do arquivo de saída. Ex.
saida.ncl)"

#define NUM_SEGMENTOS (inserir número de segmentos, caso seja conteúdo ao vivo)

#define DIRETORIO_MIDIAS "(inserir caminho do diretório onde estão os segmentos)"

int numeroSegmentos = NUM_SEGMENTOS;

TiXmlElement *mainSwitch = new TiXmlElement("switch");


/** FUNÇÃO QUE CONTA O NÚMERO DE SEGMENTOS PRESENTES NO DIRETÓRIO INFORMADO **/

void segmentsCounter(){
    int numSegmentos = 0;
    DIR *dir;
    struct dirent *lsdir;

    dir = opendir(DIRETORIO_MIDIAS);

    while ((lsdir = readdir(dir)) != NULL){
        numSegmentos++;
    }

    closedir(dir);

    numeroSegmentos = numSegmentos - 2;

    //se não tem segmentos no diretório, o conteúdo é ao vivo
    if (numeroSegmentos == 0){
        numeroSegmentos = NUM_SEGMENTOS;
    }
}
```

```
/** FUNÇÃO QUE MONTA O DOCUMENTO DE SAÍDA **/
void createConvertedDocument(TiXmlDocument *doc){
    TiXmlDocument convertedDocument;

    TiXmlElement *head =doc->FirstChildElement("ncl")->FirstChildElement("head");
    TiXmlElement *ruleBase = head->FirstChildElement("ruleBase");

    for (int i=0; i < numeroSegmentos; i++){
        char ruleName[8] = "rSegNum";
        char count[numeroSegmentos];
        sprintf(count,"%d",i+1);

        TiXmlElement *rule = new TiXmlElement("rule");
        rule->SetAttribute("id", strcat(ruleName, count));
        rule->SetAttribute("var", "segnum");
        rule->SetAttribute("comparator", "eq");
        rule->SetAttribute("value", count);

        ruleBase->InsertEndChild(*rule);
    }

    TiXmlElement *connectorBase = head->FirstChildElement("connectorBase");
    if (!connectorBase){
        connectorBase = new TiXmlElement("connectorBase");
    }

    TiXmlElement *causalConnector = new TiXmlElement("causalConnector");
    causalConnector->SetAttribute("id", "onEndSetStopStart");

    TiXmlElement *connectorParam = new TiXmlElement("connectorParam");
    connectorParam->SetAttribute("name", "num");

    TiXmlElement *simpleCondition = new TiXmlElement("simpleCondition");
    simpleCondition->SetAttribute("role", "onEnd");

    TiXmlElement *compoundAction = new TiXmlElement("compoundAction");
    compoundAction->SetAttribute("operator", "seq");

    TiXmlElement *simpleActionSet = new TiXmlElement("simpleAction");
    simpleActionSet->SetAttribute("role", "set");
    simpleActionSet->SetAttribute("value", "$num");

    TiXmlElement *simpleActionStop = new TiXmlElement("simpleAction");
    simpleActionStop->SetAttribute("role", "stop");

    TiXmlElement *simpleActionStart = new TiXmlElement("simpleAction");
    simpleActionStart->SetAttribute("role", "start");

    compoundAction->InsertEndChild(*simpleActionSet);
    compoundAction->InsertEndChild(*simpleActionStop);
    compoundAction->InsertEndChild(*simpleActionStart);

    causalConnector->InsertEndChild(*connectorParam);
    causalConnector->InsertEndChild(*simpleCondition);
    causalConnector->InsertEndChild(*compoundAction);

    connectorBase->InsertEndChild(*causalConnector);
```

```
    TiXmlElement *connectors = head->FirstChildElement("connectorBase");
    if (!connectors){
        head->InsertEndChild(*connectorBase);
    }

    TiXmlElement *body =doc->FirstChildElement("ncl")->FirstChildElement("body");

    TiXmlElement *globalVarMedia = new TiXmlElement("media");
    globalVarMedia->SetAttribute("id", "globalSegNum");
    globalVarMedia->SetAttribute("type", "application/x-ginga-settings");

    TiXmlElement *property = new TiXmlElement("property");
    property->SetAttribute("name", "segnum");
    property->SetAttribute("value", "1");

    globalVarMedia->InsertEndChild(*property);

    TiXmlElement *port = body->FirstChildElement("port");

    TiXmlElement *newBody = new TiXmlElement("body");
    newBody->InsertEndChild(*port);
    newBody->InsertEndChild(*globalVarMedia);
    newBody->InsertEndChild(*mainSwitch);

    TiXmlElement *ncl = new TiXmlElement("ncl");
    int size = strlen(DOCUMENTO_SAIDA);
    char *documentID = (char*)malloc(sizeof(char) * (256));
    strcpy(documentID, DOCUMENTO_SAIDA);
    documentID[size-4] = *"\0";
    ncl->SetAttribute("id", documentID);
    ncl->SetAttribute("xmlns", "http://www.ncl.org.br/NCL3.0/EDTVProfile");
    ncl->InsertEndChild(*head);
    ncl->InsertEndChild(*newBody);

    TiXmlDeclaration * decl = new TiXmlDeclaration("1.0", "ISO-8859-1", "");

    convertedDocument.LinkEndChild(decl);
    convertedDocument.LinkEndChild(ncl);
    convertedDocument.SaveFile(DOCUMENTO_SAIDA);

    printf("\n\nDocumento convertido com sucesso!\n\n");

}

/** FUNÇÃO QUE CRIA OS SWITCHES INTERNOS AO SWITCH DO DOCUMENTO DE ENTRADA **/
void createInternalSwitches(TiXmlElement *media){
    TiXmlElement *internalSwitch = new TiXmlElement("switch");

    char *mediaID = (char*)malloc(sizeof(char) * (256));
    strcpy(mediaID, media->Attribute("id"));

    internalSwitch->SetAttribute("id", strcat(mediaID, "Selection"));
    strcpy(mediaID, media->Attribute("id"));

    for (int i=0; i < numeroSegmentos; i++){

        char counter[numeroSegmentos];
        sprintf(counter,"%d",i+1);
```

```
        TiXmlElement *context = new TiXmlElement("context");
        context->SetAttribute("id", strcat(strcat(mediaID, "Context"), counter));
        strcpy(mediaID, media->Attribute("id"));

        char *contextID = (char*)malloc(sizeof(char) * (256));
        strcpy(contextID, strcat(mediaID, "Context"));
        strcpy(mediaID, media->Attribute("id"));

        TiXmlElement *port = new TiXmlElement("port");
        port->SetAttribute("id", strcat(strcat(contextID, "Port"), counter));
        port->SetAttribute("component", strcat(mediaID, counter));
        strcpy(contextID, context->Attribute("id"));
        strcpy(mediaID, media->Attribute("id"));

        TiXmlElement *segmentMedia = new TiXmlElement("media");
        segmentMedia->SetAttribute("id", port->Attribute("component"));
        segmentMedia->SetAttribute("descriptor", media->Attribute("descriptor"));
        /* Lógica para nomear os segmentos */
        int size = strlen(media->Attribute("src"));
        char extension[4];
        char *mediaSource = (char*)malloc(sizeof(char) * (256));
        strcpy(mediaSource, media->Attribute("src"));
        char *sentinel = (char*)"\0";
        extension[0] = mediaSource[size-4];
        extension[1] = mediaSource[size-3];
        extension[2] = mediaSource[size-2];
        extension[3] = mediaSource[size-1];
        char count[numeroSegmentos];
        mediaSource[size-4] = *sentinel;
        strcat(mediaSource, "0");
        if (i+1 <= 9){
            strcat(mediaSource, "0");
        }
        if (i+1 <= 99){
            strcat(mediaSource, "0");
        }
        sprintf(count,"%d",i+1);
        strcat(mediaSource, count);
        strcat(mediaSource, extension);
        mediaSource[size+4] = *sentinel;
        /* Fim da lógica para nomear os segmentos */
        segmentMedia->SetAttribute("src", mediaSource);

        TiXmlElement *link = new TiXmlElement("link");
        link->SetAttribute("xconnector", "onEndSetStopStart");

        TiXmlElement *onEndBind = new TiXmlElement("bind");
        onEndBind->SetAttribute("role", "onEnd");
        onEndBind->SetAttribute("component", segmentMedia->Attribute("id"));

        TiXmlElement *setBind = new TiXmlElement("bind");
        setBind->SetAttribute("role", "set");
        setBind->SetAttribute("component", "globalSegNum");
        TiXmlElement *bindParam = new TiXmlElement("bindParam");
        bindParam->SetAttribute("name", "num");
        sprintf(counter,"%d",i+2);
        bindParam->SetAttribute("value", counter);
        sprintf(counter,"%d",i+1);
        setBind->InsertEndChild(*bindParam);
```

```
        TiXmlElement *stopBind = new TiXmlElement("bind");
        stopBind->SetAttribute("role", "stop");
        stopBind->SetAttribute("component", mainSwitch->Attribute("id"));

        link->InsertEndChild(*onEndBind);
        link->InsertEndChild(*setBind);
        link->InsertEndChild(*stopBind);

        if (i+1 < numeroSegmentos){
            TiXmlElement *startBind = new TiXmlElement("bind");
            startBind->SetAttribute("role", "start");
            startBind->SetAttribute("component", mainSwitch->Attribute("id"));
            link->InsertEndChild(*startBind);
        }

        context->InsertEndChild(*port);
        context->InsertEndChild(*segmentMedia);
        context->InsertEndChild(*link);

        TiXmlElement *internalBindRule = new TiXmlElement("bindRule");
        internalBindRule->SetAttribute("constituent", contextID);
        char ruleName[8] = "rSegNum";
        internalBindRule->SetAttribute("rule", strcat(ruleName, counter));

        internalSwitch->InsertEndChild(*internalBindRule);
        internalSwitch->InsertEndChild(*context);
    }

    TiXmlElement *bindRuleElement = mainSwitch->FirstChildElement("bindRule");
    do{
        if(strcmp(bindRuleElement->Attribute("constituent"), mediaID) == 0){
            bindRuleElement->SetAttribute("constituent", internalSwitch->
Attribute("id"));
            break;
        }
        bindRuleElement = bindRuleElement->NextSiblingElement("bindRule");
    } while(bindRuleElement);

    mainSwitch->InsertEndChild(*internalSwitch);
}

/** FUNÇÃO QUE LOCALIZA O SWITCH PRINCIPAL NO DOCUMENTO DE ENTRADA **/
void getSwitch(TiXmlDocument *doc){
    doc->LoadFile();

    TiXmlElement *body = doc->FirstChildElement("ncl")->
FirstChildElement("body");
    TiXmlElement *switchElement = body->FirstChildElement("switch");

    //verifica se existe switch declarado no corpo do documento
    if (switchElement){
        //se existe media declarada dentro do switch
        if (switchElement->FirstChildElement("media")){

            mainSwitch->SetAttribute("id", switchElement->Attribute("id"));
            TiXmlElement *bindRuleElement = switchElement->
FirstChildElement("bindRule");
```

```
            do{
                //copia os elementos bindRule do switch do documento de entrada
para o mainSwitch
                mainSwitch->InsertEndChild(*bindRuleElement);
                bindRuleElement = bindRuleElement->
NextSiblingElement("bindRule");
            } while(bindRuleElement);

            TiXmlElement *media = switchElement->FirstChildElement("media");

            do{
               //se a midia declarada é a midia especial
               if (strcmp(media->Attribute("type"), "video/segmented") == 0){
                   /* chama função que vai montar os switches internos */
                   createInternalSwitches(media);
               }

               media = media->NextSiblingElement("media");
            } while(media);

        }
        else{
            //não existe mídia declarada dentro do switch
            printf("Nao foi encontrada nenhuma midia dentro do switch do
documento de entrada! \n");
            exit(0);
        }
    }
    else{
        //não existe switch declarado no corpo do documento
        printf("Nao foi encontrado nenhum switch no documento de entrada! \n");
        exit(0);
    }

}

/** FUNÇÃO PRINCIPAL **/
int main(){
    TiXmlDocument *doc = new TiXmlDocument(DOCUMENTO_ENTRADA);
    segmentsCounter();
    getSwitch(doc);
    createConvertedDocument(doc);
}
```

ANEXO 1 – XML SCHEMAS DO PERFIL DA LINGUAGEM NCL ESTENDIDA

ÁREA FUNCIONAL *STRUCTURE* (módulo *Structure*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Structure.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the Structure module namespace,
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Structure"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

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

  <complexType name="nclPrototype">
    <sequence>
      <element ref="structure:head" minOccurs="0" maxOccurs="1"/>
      <element ref="structure:body" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
  </complexType>

  <complexType name="headPrototype">
  </complexType>

  <complexType name="bodyPrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="ncl" type="structure:nclPrototype"/>
  <element name="head" type="structure:headPrototype"/>
  <element name="body" type="structure:bodyPrototype"/>

</schema>
```

ÁREA FUNCIONAL *LAYOUT* (módulo *Layout*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Layout.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Layout module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Layout"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="regionBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="device" type="string" use="optional"/>
        <attribute name="region" type="string" use="optional"/>
  </complexType>

  <complexType name="regionPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="layout:region" />
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
    <attribute name="height" type="string" use="optional"/>
    <attribute name="left" type="string" use="optional"/>
    <attribute name="right" type="string" use="optional"/>
    <attribute name="top" type="string" use="optional"/>
    <attribute name="bottom" type="string" use="optional"/>
    <attribute name="width" type="string" use="optional"/>
    <attribute name="zIndex" type="integer" use="optional"/>
  </complexType>


  <!-- declare global attributes in this module -->

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

  <!-- declare global elements in this module -->
  <element name="regionBase" type="layout:regionBasePrototype"/>
  <element name="region" type="layout:regionPrototype"/>

</schema>
```

ÁREA FUNCIONAL *PRESENTATION SPECIFICATION* (módulo *Descriptor*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Descriptor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Descriptor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Descriptor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

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

  <complexType name="descriptorPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="descriptor:descriptorParam"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="player" type="string" use="optional"/>
  </complexType>

  <complexType name="descriptorBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="descriptorParam"
type="descriptor:descriptorParamPrototype"/>
  <element name="descriptor" type="descriptor:descriptorPrototype"/>
  <element name="descriptorBase"
type="descriptor:descriptorBasePrototype"/>

  <!-- declare global attributes in this module -->
  <attributeGroup name="descriptorAttrs">
    <attribute name="descriptor" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

ÁREA FUNCIONAL *COMPONENTS* (módulo *Media*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Media.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Media module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Media"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="mediaPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="src" type="anyURI" use="optional"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="media" type="media:mediaPrototype"/>

</schema>
```

ÁREA FUNCIONAL *PRESENTATION CONTROL* (módulo *TestRule*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TestRule.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL TestRule module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TestRule"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >
```

```
<complexType name="ruleProtoype">
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="var" type="string" use="required"/>
    <attribute name="value" type="string" use="required"/>
    <attribute name="comparator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="eq"/>
          <enumeration value="ne"/>
          <enumeration value="gt"/>
          <enumeration value="gte"/>
          <enumeration value="lt"/>
          <enumeration value="lte"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="compositeRulePrototype">
    <choice minOccurs="2" maxOccurs="unbounded">
      <element ref="testRule:rule"/>
      <element ref="testRule:compositeRule"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="operator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="and"/>
          <enumeration value="or"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="ruleBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="rule" type="testRule:ruleProtoype"/>
  <element name="compositeRule" type="testRule:compositeRulePrototype"/>
  <element name="ruleBase" type="testRule:ruleBasePrototype"/>

</schema>
```

ÁREA FUNCIONAL *PRESENTATION CONTROL* (módulo *TestRuleUse*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TestRuleUse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL TestRuleUse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="bindRulePrototype">
    <attribute name="constituent" type="IDREF" use="required" />
    <attribute name="rule" type="string" use="required" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="bindRule" type="testRule:bindRulePrototype"/>

</schema>
```

ÁREA FUNCIONAL *PRESENTATION CONTROL* (módulo *ContentControl*):

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ContentControl.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL ContentControl module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ContentControl"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >
```

```
<complexType name="defaultComponentPrototype">
    <attribute name="component" type="IDREF" use="required" />
  </complexType>

  <!-- define the switch element prototype -->

  <complexType name="switchPrototype">
    <choice>
    <element ref="contentControl:defaultComponent" minOccurs="0"
maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="defaultComponent"
type="contentControl:defaultComponentPrototype"/>
  <element name="switch" type="contentControl:switchPrototype"/>

</schema>
```

ÁREA FUNCIONAL *INTERFACES* (módulo *CompositeNodeInterface*):

```
<!--
XML Schema for the NCL modules

Public URI:
http://www.ncl.org.br/NCL3.0/modules/NCL30CompositeNodeInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Composite Node Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"

xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/CompositeNodeInte
rface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="compositeNodePortPrototype">
    <attribute name="id" type="ID" use="required" />
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="port"
type="compositeInterface:compositeNodePortPrototype" />

</schema>
```