4
Middleware Ginga-NCL como Plugin para Navegadores Web

De forma análoga ao que existe hoje ao se embutir um objeto de mídia HTML em uma aplicação NCL, a integração entre o *middleware* Ginga e um navegador Web, alcançada através da adaptação do Ginga como um *plugin* reprodutor, permite que páginas *Web* especifiquem aplicações NCL como seus objetos de mídia. Contudo, o suporte a um novo tipo de mídia pode representar uma tarefa complexa. Dessa forma, na hora de oferecer tal suporte alguns requisitos devem ser observados, como a adaptação do Ginga à arquitetura de *plugins* e a integração entre as máquinas de apresentação HTML e NCL.

A adaptação do Ginga à arquitetura de *plugins*, definida pelo navegador Web, envolve questões relacionadas à plataforma, pois o *plugin* utiliza recursos específicos da plataforma para a apresentação da mídia. Existe, nesse ponto, uma divergência na escolha do ambiente, uma vez que o *middleware* Ginga está voltado para sistemas embarcados e o navegador para sistemas *Desktop*. Além das questões ligadas a plataforma, a adaptação do Ginga à arquitetura de *plugins* envolve a especificação de procedimentos, em conformidade com a NPAPI, visando à inicialização, configuração e execução do *middleware* Ginga como um *plugin* para o navegador alvo, o Mozilla Firefox.

Uma vez atingida a adaptação do *middleware* Ginga como um *plugin* reprodutor NCL este trabalho direciona seus esforços para a integração, no que diz respeito ao controle de apresentação, comunicação e mecanismos de edição. A grande parte dos *plugins* reprodutores para *Web* oferece um grau limitado de integração para com seus respectivos navegadores, disponibilizando somente uma interface para *controle de apresentação* da mídia embutida. Por outro lado, se o objetivo está na integração completa com o *middleware* Ginga, não basta prover somente uma forma de apresentar aplicações NCL em um contexto Web, negligenciando todos os recursos oferecidos por ele. Mais do que isso, deve-se valer dessa integração como uma forma de coexistência e comunicação entre a máquina de apresentação NCL e a máquina de apresentação HTML (Gecko). Neste trabalho, a integração é feita através da extensão das funcionalidades por meio de um objeto *scriptable* vinculado ao

plugin, que permite o mapeamento de código nativo C/C++ em interfaces JavaScript, e das rotinas internas definidas pela API do navegador.

A linguagem NCL possui uma característica similar a linguagem HTML no que diz respeito ao suporte a novos tipos de objetos mídia e como tal suporte é tratado por suas respectivas máquinas de apresentação. A NCL não define nenhum tipo de mídia por si só. Ao invés, ela define a cola que relaciona objetos de mídia no tempo e no espaço durante a apresentação multimídia. O documento NCL somente referencia conteúdo de mídia. O *player* de NCL, componente do *Modulo Exibidores*, é incapaz de exibir esses objetos de mídia, sendo somente capaz de manter a sincronização, tal como especificada pela aplicação. Para exibir objetos de mídia, o *player* NCL recorre a exibidores de mídia de terceiros que são embutidos previamente. Como uma linguagem de cola, a NCL não restringe ou prescreve nenhum tipo de objeto de mídia. Os tipos de objetos de mídia que serão suportados dependem exclusivamente dos componentes *players* registrados junto ao *player* NCL.

Como visto na Subseção 3.2.1.1, os componentes *players* seguem um mesmo modelo de interface na adição de todos os exibidores de mídias. Como a NPAPI, o *middleware* implementa essa funcionalidade através do mecanismo de *plugin*. Como já mencionado, a NCL não restringe nenhum tipo de objeto de mídia e, além disso, as aplicações NCL podem ser também objetos de mídia. Como consequência, aplicações NCL podem ser embutidas em outras aplicações NCL. Portanto, para que isso seja possível, um *player* NCL também deve agir como um *plugin* de outro *player* NCL e, como tal, deve suportar o modelo de interface que o define. Como consequência, além de definir uma API comum para os *players* baseado no mecanismo de *plugin*, também é possível ter aplicações NCL embutidas em outras linguagens, como propõe esta dissertação. Contudo, de forma particular, este trabalho foca na interface com o *player* NCL no desenvolvimento da ponte de comunicação que permite a integração entre as duas máquinas de apresentação. Porém, o suporte a outros *players* pode ser facilmente estendido com algumas modificações.

No que concerne à integração buscando o *controle de apresentação* da aplicação NCL embutida. O *middleware* Ginga deve expor a interface do *Formatador* (*player* NCL) para controle de apresentação da aplicação. Dessa forma, a especificação de uma nova interface JavaScript, tendo o controle como fim, pode fazer uso da interface oferecida pelo *player*, permitido que as aplicações NCL embutidas na página Web tenham suas apresentações iniciadas, paralisadas, interrompidas ou resumidas.

Já no que diz respeito à integração vislumbrando a *comunicação* entre as máquinas de apresentação e de acordo com (SOARES, 2009), na inserção de objetos declarativos em aplicações NCL a comunicação entre um ou mais contextos pode ocorrer através da especificação de pontos de interface definidos por *âncoras de conteúdo* e *propriedade*. Assim, este trabalho define interfaces e estruturas JavaScript para o tratamento desses pontos de interfaces junto ao *Formatador*, com intuito de criar um canal de comunicação e sincronização entre um contexto HTML e um contexto NCL.

Quanto à integração visando o suporte a *mecanismos de edição*, a *edição em tempo de exibição* (COSTA, MORENO, *et al.*, 2006) é considerada, pois no contexto da TV Digital, esse recurso permite que aplicações NCL sejam editadas dinamicamente através de comandos específicos (Comandos de Edição). Dessa forma, com o mesmo objetivo, uma interface JavaScritpt para a submissão de comandos de edição junto ao módulo *Gerenciador de Bases Privadas* é contemplada, permitindo que documentos HTML, através de código JavaScript, editem o conteúdo da aplicação NCL embutida.

Resolvido os problemas mencionados, será possível embutir aplicações interativas NCL em uma página Web, permitindo que a página, através do acionamento do código JavasScript, controle a apresentação e altere o conteúdo da aplicação NCL embutida em tempo de exibição. Mais ainda, com a comunicação entre os ambientes de apresentação será possível sincronizar temporal e espacialmente os objetos de mídia especificados na aplicação NCL embutida com os objetos de mídia, de uma forma mais ampla, da página Web. Portanto, com o objetivo de tornar o *middleware* Ginga um reprodutor hipermídia para Web, esta dissertação propõe a adaptação do *middleware* Ginga à arquitetura de *plugins* do Mozilla Firefox, como discute a Seção 4.1, e a integração entre a máquina de apresentação HTML e a máquina de apresentação NCL, como discute a Seção 4.2.

Cabe alertar o leitor para o fato de que nas próximas seções será dada ênfase a alguns conceitos importantes da NCL, como a composicionalidade, interfaces entre contextos, elos e eventos. Caso o leitor julgue necessário, poderá recorrer ao Capitulo 2, onde o modelo conceitual NCM e a linguagem NCL são apresentados com maior detalhe.

4.1. Adaptação do Ginga à Arquitetura de Plugins do Mozilla Firefox

O navegador Mozilla Firefox, usado como base neste trabalho, está direcionado a plataformas *Desktop*. Por outro lado, o *middleware* Ginga-NCL, optou por se basear em uma solução voltada para plataformas embarcadas. A escolha da plataforma pode influenciar diretamente em componentes importantes, como o *backend* gráfico, que é a biblioteca responsável por efetuar todas as operações de renderização relacionadas às superfícies, janelas e aos exibidores. Decorrente disso, antes de ser adaptado como um *plugin*, o *middleware* deve se adequar à nova plataforma Desktop, compatível com a plataforma do navegador.

Assim, esta dissertação tem como contribuição indireta a adequação do *middleware* Ginga à plataforma Windows e ao seu motor gráfico DirectX, apresentado no Apêndice A. Por não introduzir nenhuma mudança significativa na arquitetura do *middleware* Ginga, além da substituição de componentes e módulos, esta dissertação deixa o detalhamento da adaptação do Ginga à nova plataforma para o Capítulo 5, referente à implementação, se concentrando no capítulo corrente apenas na sua integração Ginga ao ambiente Web, como *plugin*.

Supondo, assim, o *middleware* Ginga adequado à nova plataforma, a escolha do modo de apresentação do *plugin* recaiu no modo *windowed,* uma vez que ele se mostra o mais apropriado, pois todo o processo de renderização é feito de forma independente pelo novo *backend* gráfico. Dessa forma, cabe ao Ginga definir a inicialização do *Gerente Gráfico* a partir de parâmetros do novo *backend*, passados pelo navegador (como a referência para janela nativa e dimensão da área de renderização). Esses parâmetros são passados ao Ginga pela rotina de inicialização do *plugin*. Complementar a isso, o *plugin* deve especificar ainda, suas interfaces, tal como descrito na Subseção 3.3.2, com a finalidade de registro do *Mime type* e rotinas para controle da instância.

Após a especificação das interfaces exigidas pela NPAPI, o *middleware* passa a ser reconhecido pelo navegador Mozilla Firefox como um *plugin* reprodutor de objetos de mídia NCL. A partir de então, o navegador sabe o momento de instanciar o *plugin* através da associação entre a extensão do arquivo e o *Mime type* do objeto de mídia. Assim, arquivos com sufixo .ncl, ou com tipo, definido através do atributo *type*, igual a *application/x-ncl*, devem ser apresentados pelo *plugin*.

```
<? xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="eVid" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase id="RegionBaseId">
      <region id="reg" width="100%" height="100%"/>
    </regionBase>
    <descriptorBase>
      <descriptor id="dsVideo"  region="rgVideo" />
    </descriptorBase>
  </head>
  <body id="BodyId">
    <port id="ePoint" component="video"/>
    <media id="video" descriptor="dVideo"  src="video.mp4"/>
</body>
</ncl>
```

Figura 14 – Exemplo de uma aplicação NCL simples (main.ncl).

Segundo o padrão HTML, o elemento <embed> especifica o atributo *src,* como apontador para o local do objeto de mídia a ser inserido, e outros atributos (*width* e *height*), que determinam a área de renderização na qual o objeto de mídia deve ser apresentado. Além desses, ainda outros atributos destinados ao controle de apresentação (como *autostart*, *loop*, *playcount*, *volume*, *starttime* e *endtime*) também podem ser especificados.

```
<html>
  <head>
  </head>
  <body>
    <embed src="main.ncl" width="400" height="100"/>
  </body>
</html>
```

Figura 15 – Objeto de mídia NCL embutido em um documento HTML.

A Figura 15 mostra como exemplo a especificação da aplicação NCL da Figura 14 como objeto de mídia de um documento HTML. A página Web especifica a apresentação da aplicação (main.ncl), de acordo com a área de renderização, de dimensão (400 pixels de largura e 100 pixels de altura). Já a Figura 16 abaixo, ilustra a apresentação da página Web especificada anteriormente na Figura 15, bem como a apresentação da aplicação NCL embutida. Os controles de execução estão restritos aos especificados pela aplicação. Assim fazendo, tem-se o *middleware* Ginga adaptado como um *plugin* reprodutor para o Mozilla Firefox, chamado a partir de então de *Plugin Reprodutor Ginga-NCL*, ou simplesmente *Plugin Ginga-NCL*.

Figura 16 – Apresentação da página Web especificada na Figura 15.

4.2. Integração entre as Máquinas de Apresentação (HTML e NCL)

Conforme descrito na Seção 4.1, a adaptação do *middleware* Ginga na forma de um *plugin* possibilita a apresentação de aplicações NCL em ambiente Web. Porém, tal feito ainda o restringe a um *plugin* reprodutor hipermídia limitado, incapaz de agregar funcionalidades além do incremento no aspecto visual da página Web. Esta subseção tem como objetivo apresentar a expansão das capacidades do *Plugin Ginga-NCL*, beneficiando-o com outros recursos oferecidos pelo *middleware* Ginga e, por consequência, da NCL.

Com o intuito de dinamizar e padronizar o acesso e manipulação dos documentos HTML, o W3C especifica a interface de programação DOM (W3C, 2005), cujo objetivo principal é o de permitir que programas e scripts interfiram diretamente na apresentação de documentos HTML. Para atingir esse objetivo, a API fornece acesso e atualização dinâmica de estruturas, folhas de estilo e conteúdo de um documento HTML. Ademais, a API especifica um sistema genérico de eventos para os elementos que fazem parte do modelo. Apesar de existirem *bindings*[2] da API DOM para outras linguagens (W3C, 2004), o W3C só reconhece as linguagens Java e ECMAScript como certificadas.

[2] Bindings é o termo usado em programação de computadores para extensão de funcionalidades além da linguagem de programação na qual foi originalmente especificada.

O emprego do JavaScript em páginas Web, através da API DOM, proporciona maior flexibilidade ao HTML, permitindo estender as funcionalidades do elemento <embed>, por exemplo. Em contrapartida, no Mozilla Gecko, a cada entidade DOM correspondente a um elemento <embed> está relacionado um objeto *scriptable,* que oferece suporte para o mapeamento de código JavaScript à API de um dado sistema. Considerando tais funcionalidades, este trabalho promove a integração entre páginas Web e aplicações NCL embutidas permitindo que documentos HTML acessem as interfaces nativas do *middleware* Ginga a partir do código JavaScript.

O restante desse capítulo se organiza da seguinte forma, a Subseção 4.2.1 detalha a inserção de objetos de mídia NCL como objetos de mídia HTML, no que diz respeito ao aninhamento entre elementos das linguagens. Os Subseções 4.2.1.1 e 4.2.1.2 tratam da especificação de interfaces entre os dois ambientes (NCL e HTML), baseadas nas *âncoras de conteúdo* e nas *âncoras de propriedade*, respectivamente. Já a Subseção 4.2.2 trata da especificação de uma interface JavaScript visando o controle de apresentação e a obtenção de informações das *Base Privadas* e seus respectivos documentos. Por fim, a Subseção 4.2.3 trata da especificação de uma interface JavaScript focando a edição em tempo de exibição.

4.2.1. Objetos de Mídia NCL como Objetos de Mídia HTML

O aninhamento de objetos de mídia NCL em aplicações HTML se dá de forma semelhante ao aninhamento desses objetos em outras aplicações NCL (chamadas de aplicações pai). De uma forma geral, objetos de mídia do tipo declarativo podem ser inseridos em documentos NCL. Em particular, os *profiles* EDTV e BDTV da NCL 3.0 permitem que objetos declarativos do tipo *application/x-ncl-NCL* (Aplicações NCL) sejam embutidos em documentos NCL.

Para que seja feita a especificação de um objeto de mídia declarativo filho em uma aplicação NCL pai, é necessário inserir o elemento <media> e especificar em seu atributo *src* o caminho para o código declarativo a ser aninhado. Como exemplo, a Figura 17 ilustra um fragmento de código NCL, que especifica um objeto de mídia declarativo NCL filho (arquivo usado como exemplo na Figura 14)Figura **14 – Exemplo de uma aplicação NCL simples (main.ncl).**

```
<media id="mDecFilho" src="main.ncl">
  <propertyname="soundLevel"/>
  <area id="cAnc" clip="(chainId="ePoint",beginOffset="10s")"/>
</media>
```

Figura 17 – Exemplo especificação de um objeto de mídia NCL.

Como outros objetos de mídia da NCL, o objeto declarativo pode utilizar o atributo *descriptor* para referenciar um elemento <descriptor>, que por sua vez é responsável por inicializar parâmetros relacionados à apresentação do objeto (como a posição na tela, na qual deve ser exibida). Além disso, a especificação do objeto declarativo pode conter definições de *âncoras de conteúdo* e *âncoras de propriedade*, como ilustra o exemplo da Figura 17.

Para a aplicação NCL pai, o objeto declarativo filho é visto como um conjunto de cadeias temporais. Cada cadeia temporal corresponde a uma sequência de eventos de apresentação no tempo, iniciadas a partir do evento que corresponde ao início da apresentação do objeto declarativo. Seções, dentro das cadeias temporais, podem ser associadas a *âncoras de conteúdo* através do atributo *clip*, cujo valor é representado por uma tripla "(*chainId*, *beginOffset*, *endOffset*)". O parâmetro *chainId* identifica uma das cadeias temporais definidas pelo objeto declarativo. Já os parâmetros *beginOffset* e *endOffset*, determinam, respectivamente, o instante inicial e final da *âncora de conteúdo* na cadeia. Dessa forma, como exemplo, ainda na Figura 17, a *âncora de conteúdo* "cAnc" inicia no instante em que a cadeia acessada através da porta "ePoint" (especificada na Figura 14) atingir 10 segundos, a partir do início da apresentação. Por fim, a âncora "cAnc" termina quando a cadeia chegar ao fim, uma vez que o parâmetro *endOffset* não foi especificado. Além disso, outra forma de definir *âncoras de conteúdo* é feita através do atributo *label,* que identifica um objeto individualmente.

Com as *âncoras de propriedades*, os objetos de mídia declarativos NCL filhos podem tornar externo propriedades definidas internamente a eles ou definir propriedades globais comuns a todo objeto, usando para isso o atributo *name*. Como exemplo, ainda na Figura 17, a *âncora de propriedade*, cujo atributo *name* tem valor igual a *soundLevel*, determina a visibilidade dessa propriedade, que é usada para definir o volume do objeto de mídia declarativo NCL filho.

Tanto as *âncoras de conteúdo* quanto as *âncoras de propriedades* podem ser utilizadas como condição para disparo de elos, através de elementos <link>, definidos externamente ao objeto de mídia declarativo NCL (filho). Propriedades podem também ter seus valores alterados, decorrente de ações de um elo

definido externamente ao objeto de mídia declarativo NCL. *Âncoras de conteúdo* também podem sofrer ações referentes à sua exibição, provenientes de elos.

Como consequência, a especificação de *âncoras de conteúdo* e *propriedade*, através dos elementos <area> e <property> , quando utilizadas como pontos de interface em elementos <link>, estabelecem uma ponte de comunicação em dois sentidos, entre o *player* NCL e o *player* do objeto declarativo, que executa o objeto de mídia declarativo filho.

A inserção de objetos declarativos NCL em documentos HTML como objetos de mídia pode ser pensada da mesma forma que a inserção de objetos de mídia declarativos em documentos NCL. Nesse sentido, uma vez que a aplicação NCL embutida é encapsulada pelo código JavaScript inserido na página Web, como mencionado na seção anterior, toda a interface com o reprodutor NCL (*Formatador*) é feita por intermédio de *scripts* estendidos por interfaces nativas do objeto *scriptable*.

A especificação de elos que utilizem as *âncoras de conteúdo*, com será visto na Subseção 4.2.1.1 e as *âncoras de propriedade,* que será vista na Subseção 4.2.1.2, são feitas através de rotinas definidas por *scripts* JavaScript. Essas rotinas (chamadas de *handlers)* são usadas para o tratamento de eventos gerados pelas âncoras e exercem papel equivalente aos elos e conectores especificados no aninhamento de contextos NCL.

4.2.1.1. Definição de Elos a partir de Âncoras de Conteúdo

Tal como acontece no aninhamento de aplicações NCL, a definição de *âncoras de conteúdo* para o objeto imperativo NCL embutido no documento HTML também deve ser possível. Assim como na NCL, o objeto de mídia declarativo embutido é tratado pelo documento HTML como um conjunto de cadeias temporais. No entanto, para que uma página Web consiga especificar *âncoras de conteúdo* em seu objeto de mídia declarativo NCL filho, ela deverá fazê-lo de forma imperativa, usando para isso, a interface Javascript fornecida pelo *Plugin Ginga-NCL*.

Como exemplifica a Figura 18, a aplicação (*main.ncl*) da Figura 14 possui somente uma cadeia temporal, que é identificada por uma de suas interfaces de entrada, mais especificamente pela porta de id “ePoint”, filha do elemento <body>. Assim, a *âncora de conteúdo* definida pela estrutura JavaScript *cAnchor* especifica que, quando a cadeia temporal identificada pelo id “ePoint” atingir 10

segundos a partir do início de sua apresentação, a âncora “ePointAnchor” iniciará e, quando atingir 30 segundos, terminará.

```
<html>
<head>
</head>
<body>
  <embed id="gPlugId" src="main.ncl" width="400" height="100"/>
  <button type="button" onclick="ePost("presentation","stop")"/>
  <script language = "JavaScript">
      var gPlugin;
      var cAnchor;

      gPlugin = getElementById("gPlugId");

      if(gPlugin){
        cAnchor.id = "ePointAnchor";
        cAnchor.chainId = "ePoint";
        cAnchor.beginOffset = "10s";
        cAnchor.beginOffset = "30s";
        cAnchor.recvEvtHandler = cEventRecv;
        gPlugin.addContentAchor(cAnchor);
      }
  </script>
</body>
</html>
```

Figura 18 – Definição de elo a partir de uma âncora de conteúdo na aplicação NCL embutida.

A notificação de eventos oriundos das transições da máquina de estados associada às *âncoras de conteúdo* deve ser tratada pelo *script*. Nesse sentido, o objeto *scriptable,* associado ao *Plugin Ginga-NCL*, define uma rotina *handle* para o tratamento de eventos, a *recvEvtHandler* para o recebimento dos eventos e outra rotina interna, a *postEvent*, para notificação dos eventos. Cabe ao autor da página Web especificar como os eventos oriundos do *Formatador* refletirão na página. Em contrapartida, cabe a aplicação NCL especificar como os eventos DOM “mapeados” refletirão na aplicação NCL embutida, como feito pelos elos e conectores quando no aninhamento de aplicações NCL.

```
<html>
  <head>
    <script language = "JavaScript">
       function cEventRecv(evtType, action, anchorType, value){
        if(anchorType == "area"){
          switch(evtType){
            case "presentation":
              switch (action){
                case "start":
                  Alert ("Inicio da âncora de conteúdo");
                  break;
                case "stop":
                  Alert ("Fim da âncora de conteúdo");
                  break;
```

```
                }
                break;
            }
          }
        }
        function ePost(evtType, action, anchorType, value){
          gPlugin.postEvent(evtType, action, anchorType, value);
        }
      </script>
    </head>
    <body>
    <!-- Código omitido, veja Figura 18 -->
    </body>
</html>
```

Figura 19 – Especificação de *handlers* para envio e recebimento de eventos da âncora de conteúdo.

A Figura 19 ilustra a definição de dois *handlers* para a âncora de conteúdo “ePointAnchor” definida na Figura 18. Quando atingir os 10 segundos de apresentação da cadeia temporal o player declarativo deve notificar o início da apresentação da âncora, chamando a função *handle cEventRecv* e passando as variáveis (*evtType= presentation, action =start, anchorType = area*) como parâmetro. Da mesma forma que, ao atingir os 30 segundos de apresentação, o *player* declarativo deve notificar o término da apresentação da âncora, chamando a rotina *handle cEventRecv*, passando as variáveis (*evtType= presentation, action=stop, anchorType = area*) como parâmetro. Nesse exemplo, em ambos os casos, a página Web reage exibindo uma janela de alerta notificando sobre a mudança de estado da *âncora de conteúdo*. A função *ePost* é invocada a qualquer instante, ao pressionar do botão. O clique no botão especifica o término da âncora de conteúdo, enviado um evento que passa (*evtType= presentation, action= stop, anchorType = area*) como parâmetro.

A Figura 20 ilustra a *visão estrutural* do exemplo especificado pela Figura 18 e Figura 19. Na figura o retângulo na cor vermelha representa a interface NCM e o retângulo na cor preta o *handler* JavaScript equivalente.

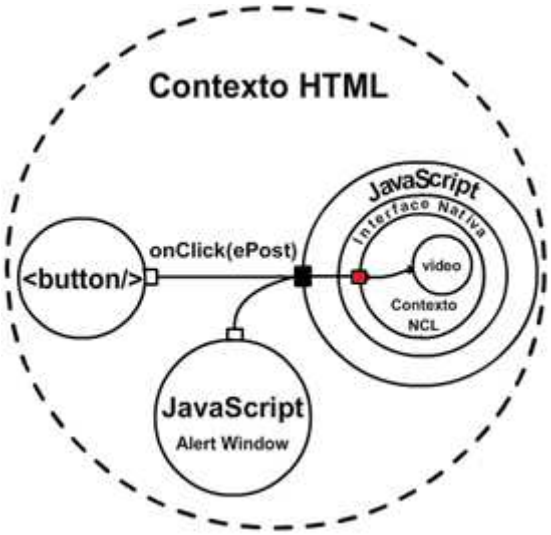

Figura 20 – Visão estrutural do exemplo da Figura 18 e da Figura 19.

Dessa forma, como mostrado ao longo dessa subseção, a especificação de *âncoras de conteúdo* como pontos de interface para o código de ligação *script*, representado pelos *handlers* de recebimento e envio de eventos, contribui para o sincronismo, de uma forma geral, entre os objetos de mídia pertencentes aos domínios envolvidos (HTML e NCL).

4.2.1.2. Definição de Elos a partir de Âncoras de Propriedade

Tal como acontece no aninhamento de aplicações NCL, a definição de *âncoras de propriedades* para o objeto imperativo NCL embutido no documento HTML também deve ser possível. Dessa forma, para que uma página Web consiga especificar *âncoras de propriedades* em seu objeto de mídia declarativo NCL filho, ela deverá fazê-lo de forma imperativa, usando também para isso, a interface JavaScript fornecida pelo *Plugin Ginga-NCL*.

```
<html>
  <head>
  </head>
  <body>
    <embed id="gPlugId" src="main.ncl" width="400"
                                       height="100"/>
    <input name="gVar" type="TEXT"
          onChange="ePost(attribution,start,property,this)"/>
    <script language = "JavaScript">
      var gPlugin;
      var pBoundsAnchor;
      var pGlobalNameAnchor;

      gPlugin = getElementById("gPlugId");
```

```
      if(gPlugin){
        pBoundsAnchor.id = "pBoundsAnchor";
        pBoundsAnchor.name = "Bounds";

        pGlobalNameAnchor.id = "pGNameAnchor";
        pGlobalNameAnchor.name = "globalName";

        gPlugin.addPropertyAchor(pBoundsAnchor);
        gPlugin.addPropertyAchor(pGlobalNameAnchor);
      }
    </script>
  </body>
</html>
```

Figura 21 – Definição de elo a partir âncora de propriedade na aplicação NCL embutida.

Como exemplificado na Figura 21, duas *âncoras de propriedade* são definidas para o objeto de mídia declarativo NCL inserido na página Web. A primeira se refere a um atributo do objeto declarativo, que especifica a posição na tela. Já a segunda referencia a interface porta, cujo id é igual à *globalName* definida como filha do elemento <body>, no objeto de mídia declarativo NCL filho embutido.

```
<html>
  <head>
    <script language = ”JavaScript”>
     function cEventRecv(evtType, action, anchorType, value){
       var gVar = getElementById("gVar");
         if(anchorType == "property"){
           switch(evtType){
             case "attribution":
               switch (action){
                 case "start":
                 break;

                 case "stop":
                 gVar.value = value;
                 break;
               }
               break;
           }
         }
      }
      function ePost(evtType, action, anchorType, value){
        Var propVal = value.value;
        gPlugin.postEvent(evtType, action, anchorType, propVal);
      }
    </script>
  </head>
  <body>
    <!-- Código omitido, veja Figura 21 -->
  </body>
</html>
```

Figura 22 – Especificação de *handlers* para envio e recebimento de eventos da âncora de propriedade.

A Figura 22 apresenta um exemplo de definição *handlers* para tratamento de eventos relacionados a *âncoras de propriedades* em um objeto de mídia declarativo. Quando houver mudança no valor da propriedade dentro do ambiente da aplicação NCL, o *player* declarativo deve notificar a mudança de valor da âncora, chamando a função handle *cEventRecv* passando as variáveis (*evtType= attribution, action=start, anchorType = property, value = X*) como parâmetros.

Por outro lado, quando a página Web modificar o valor da variável mapeada na *âncora de propriedades*, o script deve notificar o player declarativo sobre a mudança. Nesse exemplo em particular, um evento DOM (*onChange*) associado ao elemento INPUT é usado para disparar a chamada a função ePost, passando as variáveis (*evtType= attribution, action = stop,anchorType = area*) como parâmetros. A Figura 23 ilustra a *visão estrutural* do exemplo especificado pela Figura 21 e a Figura 22. Na figura o retângulo na cor vermelha representa a interface NCM e o retângulo na cor preta o *handler* JavaScript equivalente.

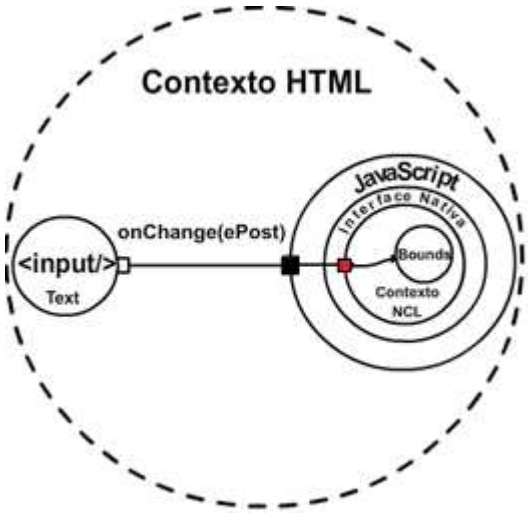

Figura 23 – Visão estrutural do exemplo da Figura 21 e da Figura 22.

Como mostrado ao longo dessa subseção, a especificação de *âncoras de propriedades* como pontos de interface para o código de ligação *script*, representado pelos *handlers* de recebimento e envio de eventos, permite que variáveis definidas no ambiente do objeto de mídia declarativo embutido sejam mapeadas para variáveis ou funções JavaScript. E, por sua vez, que variáveis ou funções JavaScript sejam mapeadas em variáveis definidas dentro do ambiente do objeto de mídia declarativo embutido. Isso, assim como na subsecção anterior, contribui para o sincronismo, de uma forma geral, entre os objetos de mídia pertencentes aos domínios envolvidos (HTML e NCL).

4.2.2.
Interface para Controle de Apresentação

Como apresentado na Subseção 3.2.1.2, os documentos NCL associados a uma *Base Privada* podem ser iniciados, pausados, retomados, posicionados e interrompidos. O *Gerente da Máquina de Apresentação* é responsável por controlar a apresentação das aplicações contidas nas bases privadas. É responsabilidade do *Plugin Ginga-NCL* oferecer uma interface para controle de apresentação da aplicação embutida.

```
<html>
  <head>
    <script language = "JavaScript">
      function play(curTPos){
        gPlugin.play();
      }
      function pause(curTPos){
        gPlugin.seek(curTimePos);
      }
    </script>
  </head>
  <body>
   <embed id="gPlgId" src="main.ncl" width="400" height="100"/>
   <button onclick="play()">Play</button>
   <button onclick="stop()">Stop</button>
   <script language = "JavaScript">
     var gPlugin;
     gPlugin = getElementById("gPlgId");
   </script>
</body>
</html>
```

Figura 24 – Especificação do controle de apresentação da aplicação NCL embutida em uma página Web.

A Figura 24 exemplifica o uso da interface de controle de apresentação da aplicação NCL (*main.ncl*) exemplificada na Figura 14. A aplicação Web especifica dois botões, um para (início e pausa) e o outro para interrupção da apresentação da aplicação.

Além da interface para controle de apresentação da aplicação, o *Plugin Ginga-NCL* deve oferecer um conjunto de rotinas para obtenção de informações relativas às bases privadas e seus respectivos documentos. De acordo com SBTVD, cada base privada está associada a um canal de TV, por outro lado no ambiente da Web não existe um canal TV, mas sim a cada instância do *Plugin Ginga-NCL* está relacionado uma base privada, que pode conter um ou mais documentos. Uma página Web pode conter mais de uma referência a instância do plugin, o que permite a referencia entre documentos contidos.

Além disso, como será visto mais adiante, para que seja possível realizar alguns procedimentos relacionados à edição em tempo de exibição, é necessário conhecer algumas informações referentes ao documento sendo executado pela máquina de apresentação. Dessa forma, o *Plugin Ginga-NCL* também deve oferecer uma interface de programação para obtenção de informações sobre o documento.

```
<html>
  <head>
  </head>
  <body>
    <embed id="gPlgId" src="main.ncl" width="400" height="100"/>
    <script language = "JavaScript">

      var gPlugin;
      gPlugin = getElementById("gPlugId");

      if(gPlugin){
        array pBases = gPlugin.getPrivateBases();
        array nclDoc = pBases[0].getCurrentNcl();

        for(var interface in nclDoc["interfaces"])
          alert("Interface" + interface.id + "encontrada");
        }
      }
    </script>
  </body>
</html>
```

Figura 25 – Exemplo de uso da interface para obter informações sobre Base Privada e seus documentos.

A Figura 25 exemplifica o uso da interface JavaScript para obter informações sobre as bases privadas e posteriormente sobre o documento corrente em execução. O exemplo obtém a lista das bases privadas disponíveis e a armazena em uma estrutura *Array* chamada *pBases*. Em seguida, obtém a referência para o documento corrente da primeira *Base Privada* listada e os elementos do tipo interface NCM presentes.

4.2.3. Interface para Edição em Tempo de Exibição

A edição em tempo de exibição é um recurso que possibilita a autoria da aplicação NCL no momento da apresentação. Como mencionado na Subseção 3.2.1.2, o *Gerente de Bases Privadas* está encarregado de receber os comandos utilizados para edição (comandos de edição) e repassá-los ao *Formatador*. Por sua vez, cabe ao *Formatador* decidir aplicá-los, garantindo a correta apresentação da aplicação.

Os comandos de edição são divididos em três subconjuntos (GOMES, RODRIGUES, *et al.*, 2006). O primeiro especifica funções para a gerência de bases privadas (como os comandos *openBase*, *activeBase* e *closeBase*). Já o segundo está focado na gerência e controle de execução dos documentos (como os comandos *addDocument*, *startDocument* e *stopDocument*). Por fim, o terceiro subconjunto especifica comandos para a manipulação dos documentos (como os comandos *addRegion*, *addMedia* e *removeConnector*).

Segundo a abordagem proposta nesta dissertação e conforme discutido na Seção 4.2.1 a manipulação direta do modelo de dados referente a aplicação NCL embutida usando apenas a interface DOM não é desejada. Como alternativa, a edição em tempo de exibição permite a aplicação cliente delegar o processo de edição á máquina de apresentação do Ginga. A edição em tempo de exibição pela máquina de apresentação do Ginga é apresentada nesta dissertação como uma alternativa a interface de edição DOM para edição da estrutura do documento.

Os comandos de edição podem ser transportados por vários meios. Em particular no SBTVD, eles são transportados pelo protocolo DSMCC junto ao fluxo de transporte elementar de áudio e vídeo, que é transmitido via radiodifusão. Outra forma possível de transporte é através do canal de interatividade. Além dessas duas formas, ainda existe, no SBTVD, a possibilidade de execução dos *Comandos de Edição* de forma direta pelo telespectador, através de um nó de mídia imperativo NCLua. Os objetos de mídia imperativos devem oferecer uma API que provenha um conjunto de métodos para suporte aos Comandos de Edição NCL e comandos do *Gerenciador de Bases Privada* (SANT'ANNA, SOARES e CERQUEIRA, 2008).

O ambiente de uma página Web é caracterizado pelo dinamismo e sua consequente adaptabilidade. Além do suporte à edição em tempo de exibição oferecido na inserção de objetos imperativos NCLua esta dissertação oferece à página Web uma interface para edição ao vivo. Tal interface permite que a aplicação Web altere a aplicação NCL embutida de acordo com a interação do usuário ou conforme estabelecido pelo autor da pagina Web e da aplicação NCL. Para isso, os comandos de edição passados pela página Web são traduzidos pelas chamadas em comandos de edição NCL, que, por sua vez, são entendidos pela Base Privada.

```
<html>
  <head>
    <script language = "JavaScript">
      function geraProp(){
```

```
        var rgCmd, dsCmd;
         var mdCmd, lkCmd;
         array pBases;
         array nclDoc;

         pBases = gPlugin.getPrivateBases();
         nclDoc = pBases[0].getCurrentNcl();

         // add region
         rgCmd.Tag   = NCL_ADD_REGION ; // 0xB
         rgCmd.pBase = pBases[0].id;
         rgCmd.docId = nclDoc["id"];
         rgCmd.rBaseId = nclDoc["regionBase"].id;
         rgCmd.region.id = "rgVideo2";
         rgCmd.region.width = "30%";
         rgCmd.region.height = "30%";

         gPlugin.liveEditing(rgCmd);

         // add descriptor
         dsCmd.Tag   = NCL_ADD_DESCRIPTOR; // 0x17
         dsCmd.pBase = -1;
         dsCmd.docId = "NCLPluginSample";
         dsCmd.descriptor.id     = "dsVideo2";
         dsCmd.descriptor.region  = "rgVideo2";

         gPlugin.liveEditing(dsCmd);

         // add media
         mdCmd.Tag   = NCL_ADD_MEDIA; // 0x27
         mdCmd.pBase = pBases[0].id;
         mdCmd.docId = nclDoc["id"];
         mdCmd.bodyId = nclDoc["body"].id;
         mdCmd.media.id = "dsVideo2";
         mdCmd.media.descriptor = "dsVideo2";
         mdCmd.media.src = "ppgd.mp4";

         gPlugin.liveEditing(mdCmd);

         // add link
         lkCmd.Tag   = NCL_ADD_LINK; // 0x2B
         lkCmd.pBase = pBases[0].id;
         lkCmd.docId = nclDoc["id"];
         lkCmd.link.id = "linkId";
         lkCmd.link.xconnector = "onEndStart";
         lkCmd.link.bind[0].role = "onEnd";
         lkCmd.link.bind[0].component = "video";
         lkCmd.link.bind[1].role = "start";
         lkCmd.link.bind[1].component = "video2";

         gPlugin.liveEditing(lkCmd);

      }
    </script>
  </head>
  <body>
    <embed id="gPlgId" src="main.ncl" width="400" height="100"/>
    <input name="gProp" value="[Gerar]" onclick="geraProp()"/>
    <script language = "JavaScript">

      var gPlugin;
      gPlugin = getElementById("gPlgId");
```

```
    </script>
  </body>
</html>
```

Figura 26 – Exemplo de edição em tempo de exibição local de uma aplicação NCL embutida em uma página Web.

A Figura 26 ilustra como exemplo, a construção dinâmica de uma aplicação a partir de comandos de edição invocados localmente. No exemplo, é considerada a aplicação NCL (*main.ncl*), que foi especificada na Figura 14. Segundo o exemplo, ao final da apresentação, a aplicação deve exibir um comercial criado dinamicamente pela submissão de um conjunto de dos comandos de edição NCL.

Ainda na Figura 26, no início da execução da aplicação NCL, é obtida a referência para a instância do *Plugin Ginga-NCL*. Em um segundo passo, em resposta ao clique no botão *gProp*, a função *geraProp* é invocada. A função *geraProp* específica a requisição dados sobre a base privada em execução, mais especificamente, são obtidas as informações sobre os elementos que compõe o documento vinculado à base (como id do documento e o id da Base de Regiões). Ainda na função *geraProp*, são criados os comandos de edição, que são posteriormente aplicados a aplicação embutida. Os comandos especificam a criação de uma região, de um descritor associado à região criada, do objeto de mídia (*ppgd.mp4*), referente à propaganda, e do elo entre o objeto de mídia principal e objeto de mídia da propaganda. O elo especifica uma relação causal entre os dois objetos, especificando que após o término do objeto de mídia principal (*video.mp4*) deve ser iniciada a apresentação do objeto de mídia da propaganda. Tal relação é definida pelo conector *onEndStart*, contido na base de conectores previamente importada.

```
<? xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="eVid" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase id="RegionBaseId">
      <region id="regVid1" width="100%" height="100%"/>
      <region id="regVid2" width="100%" height="100%"/>
    </regionBase>
    <descriptorBase>
      <descriptor id="dsVid1"  region="rgVid1" />
      <descriptor id="dsVid2"  region="rgVid2" />
    </descriptorBase>
  </head>
  <body id="BodyId">
    <port id="ePoint" component="video"/>
    <media id="video" descriptor="dVid1"  src="video.mp4"/>
    <media id="video2" descriptor="dVid2"  src="ppgd.mp4"/>
    <link id="linkId" xConnector="onEndStart" >
```

```
      <bind role="onEnd" component="video"/>
      <bind role="start" component="video2"/>
    </link>
  </body>
</ncl>
```

Figura 27 – Aplicação *main.ncl* resultante, após os comandos de edição.

Após execução dos comandos de edição, a aplicação main.ncl ficará tal qual a Figura 27 ilustra. As modificações ocasionadas pela submissão dos comandos de edição estão em destaque na figura.