5
Implementação

O *middleware* Ginga-NCL possui uma implementação de referência especificada nas linguagens C/C++ voltada para ambientes embarcados que utilizam a plataforma Linux. Contudo, como mencionado no Capitulo 4, o Ginga ainda não possuía uma solução voltada para ambientes *Desktop*. A adaptação do *middleware* na forma de um *plugin* reprodutor para Web impõe como um dos seus requisitos a ambientação do reprodutor de mídia ao ambiente de uma aplicação Desktop. Dessa forma, de acordo com o que foi apresentado nos capítulos anteriores, são propostas modificações e incrementos de funcionalidades a alguns módulos do *middleware* Ginga, conforme destacam as áreas marcadas na Figura 28.

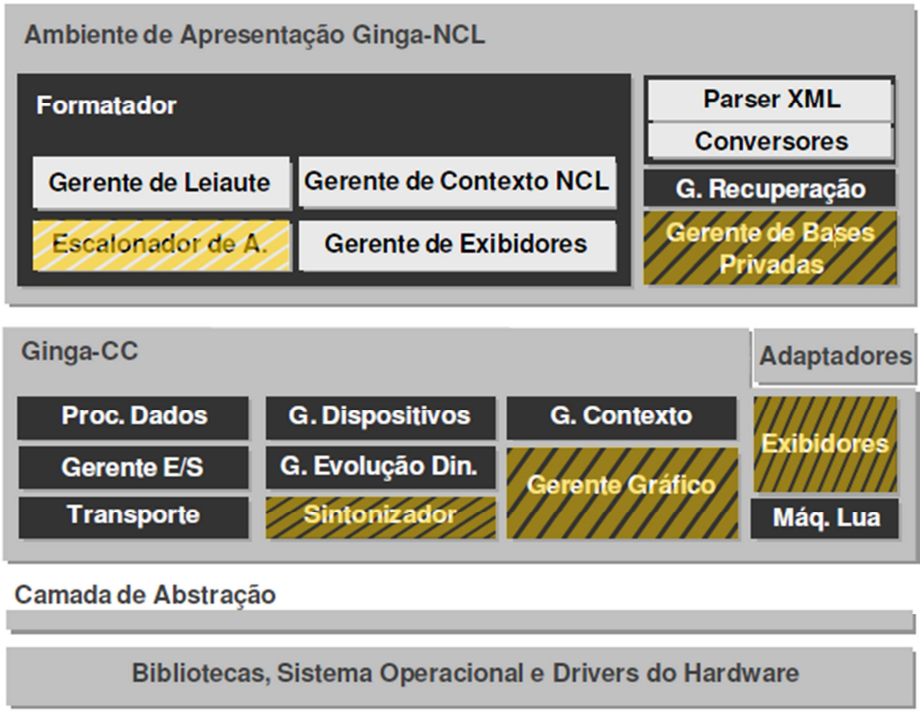

Figura 28 – Componentes e módulos afetados pela implementação proposta.

Conforme mencionado no Capitulo 4, neste capítulo também serão apresentados os detalhes da implementação na busca pela adaptação do *middleware* para que ele se torne um *plugin* para o navegador Web Mozilla Firefox e de forma decorrente um reprodutor hipermídia para plataforma *Desktop*

Windows. Para melhor divisão do trabalho, o capítulo está organizado em duas seções principais.

A primeira, a Seção 5.1, trata das principais adaptações necessárias à portabilidade do *middleware* para plataforma Windows, como um reprodutor hipermídia para Desktop. A Subseção 5.1.1 aborda a adaptação do módulo *Gerente Gráfico* ao novo *backend* gráfico, apresentado no Apêndice . A Subseção 5.1.2 detalha a adaptação do modulo *Exibidores* ao novo *backend* gráfico. Já a Subseção 5.1.4 abrange o suporte aos dispositivos de E/S de acordo com a nova plataforma.

A segunda, a Seção 5.2, enseja a incorporação de funcionalidades visando à integração com o navegador Mozilla. A Subseção 5.2.1 discorre sobre o reprodutor hipermídia obtido na Seção 5.1 e sua ambientação ao navegador Web. De forma complementar, a Subseção 5.2.2 apresenta a extensão das funcionalidades do *plugin* tal como discutidas na Seção 4.2.1.

5.1. Ginga para Plataforma Windows

A escolha do Windows ocorreu devido à facilidade de adaptação dos componentes do *middleware* declarativo Ginga-NCL a essa nova plataforma, ao baixo nível de conhecimento exigido para a instalação e configuração do ambiente, ao suporte amplo à aceleração gráfica e ao grande número de utilizadores desse Sistema Operacional.

A adaptação para a plataforma Windows exige a substituição de alguns componentes do núcleo comum do Ginga (Ginga-CC). Como apresentado na Seção 3.2.1, a arquitetura do *middleware* prevê a substituição de alguns componentes por outros customizados, inclusive os que dependem de alguma forma do *backend* gráfico do sistema. Com a substituição do *backend* gráfico, o *Gerente Gráfico* deve se adaptar ao novo modelo de apresentação, assim como os exibidores e o módulo *Sintonizador*, pois devem se adaptar aos novos provedores de conteúdo de mídia (como o DirectShow e as bibliotecas de terceiros). Além disso, o acesso e manipulação dos dispositivos de entrada (como mouse e teclado) devem ser fornecidos para que seja possível a interagir com o usuário.

5.1.1.
Gerente Gráfico

O módulo *Gerente Gráfico*, utilizado pelo *middleware* Ginga-NCL é responsável por gerenciar todo o processo de renderização de cada objeto de mídia que tenha representação visual. Na implementação de referência esse papel é feito pelo DirectFB, porém nesta abordagem foi escolhida a biblioteca proprietária DirectX em seu lugar.

A implementação de referência disponibiliza uma interface abstrata *IDeviceScreen* para a tratamento e gerência de telas fornecidas pelo motor gráfico. Para que o novo motor gráfico possa ser utilizado, foi criada uma nova classe *DXDeviceScreen* em substituição a classe *DFBDeviceScreeen* utilizada pela implementação de referência.

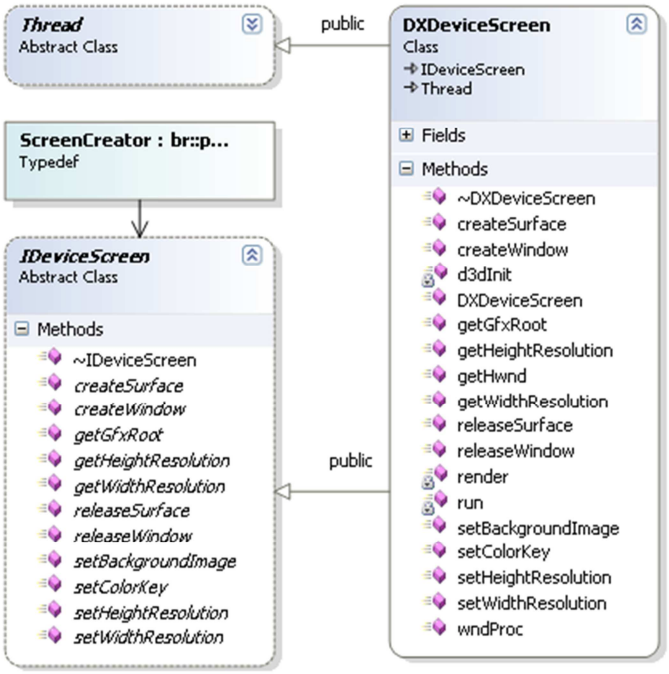

Figura 29 – Classe DXDeviceScreen.

Conforme ilustra o diagrama da Figura 29, a nova classe *DXDeviceScreen* implementa a interface *IDeviceScreen*. Como tal, ela deve oferecer os procedimentos para inicialização do motor gráfico, configuração de tela, instanciação de janelas e superfícies de exibição. Além disso, essa classe estende a classe *Thread*, a fim de criar um método concorrente para executar o *loop* de renderização dos artefatos gráficos (como janelas e superfícies).

Uma superfície, especificada pela interface *ISurface*, é uma abstração utilizada pelo *middleware* Ginga-NCL para a representação de uma região NCL. Uma região NCL pode ser representada geometricamente por um retângulo

disposto em um espaço bidimensional, definido pelo dispositivo associado ao motor gráfico ou pela geometria da sua região pai. Para que a superfície se adéque ao novo *backend* gráfico, foi criada a classe *DXSurface* em substituição à classe *DFBSurface*, utilizada na implementação de referência.

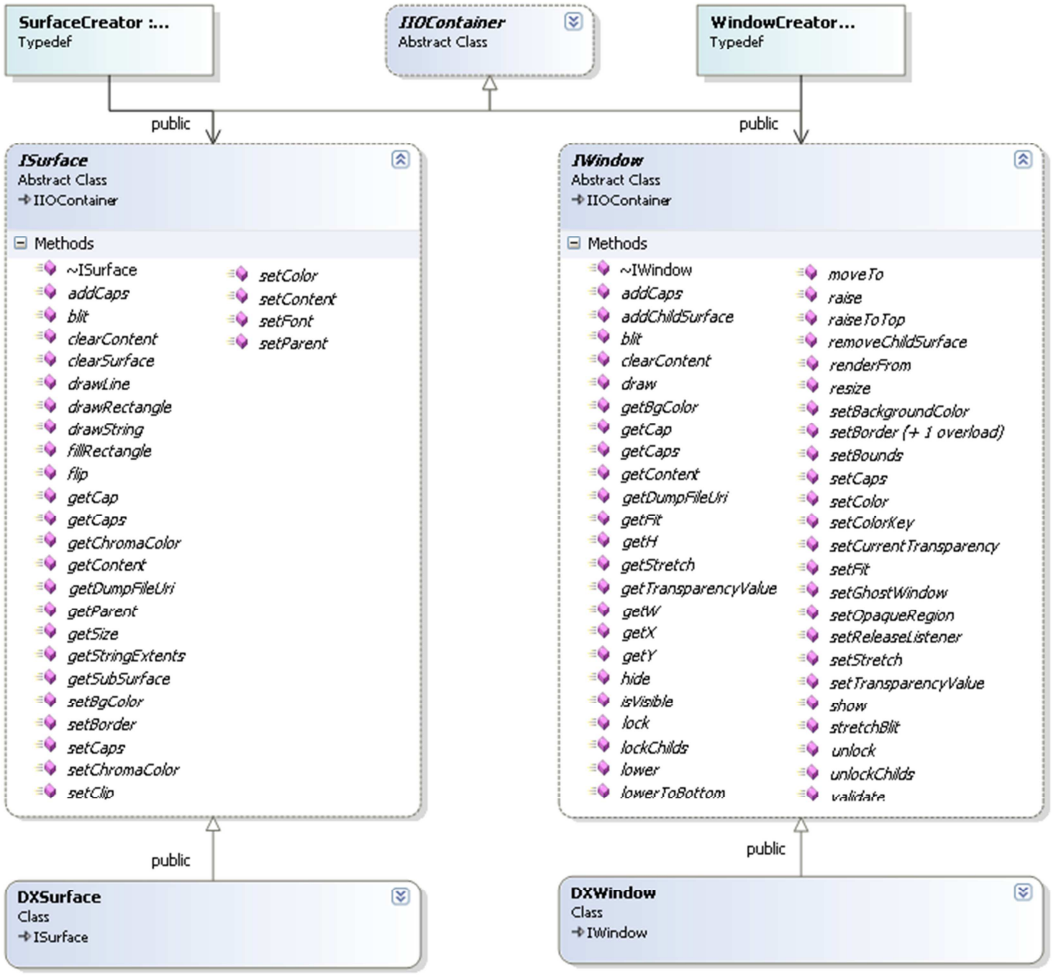

Figura 30 – Abstração de Superfícies e Janelas.

Como ilustra o diagrama da Figura 30, a classe *DXSurface* implementa a interface *ISurface* para que a superfície se adéque ao novo motor gráfico. Apesar de já existir o conceito de superfície no Direct3D, apresentado no Apêndice , ela não especifica todas as operações (como *drawLine*, *drawRectangle*, *drawString*, *blit* e *flip*) da mesma maneira que a interface *ISurface* determina. Além disso, a superfície Direct3D não se adéqua ao modelo tridimensional adotado. Dessa forma, o Direct3D deve oferecer a *DXSurface* outra abstração de superfície. Isso é feito através de um objeto geométrico primitivo na forma de retângulo no qual possa ser aplicada uma textura.

Como visto no Apêndice , um objeto primitivo é representado por um buffer de vértices, no qual são especificadas as coordenadas e informações pertinentes a textura. Para que um retângulo seja representado em um espaço tridimensional definido pelo Direct3D, esse trabalho utiliza a indexação de *buffers*, o que permite que o objeto seja especificado com quatro vértices e cinco arestas.

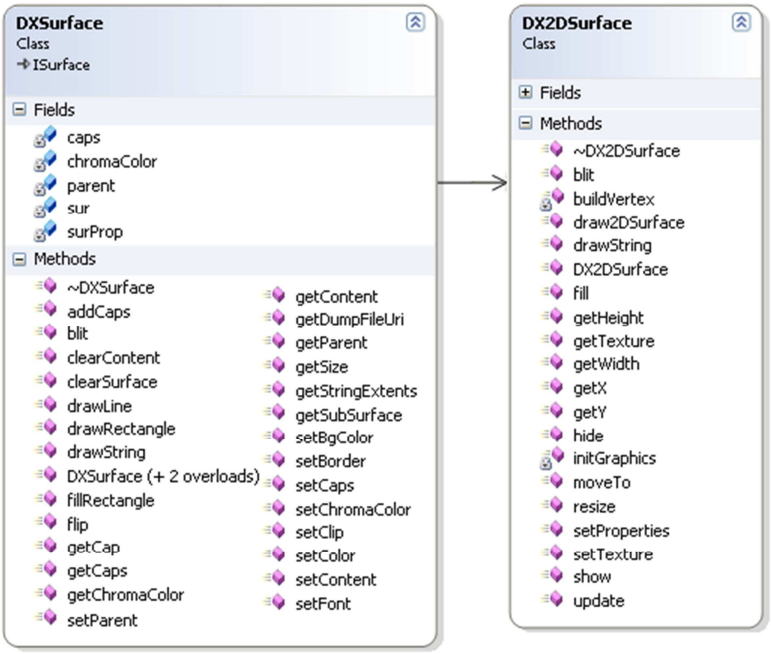

Figura 31 – Classe DX2Dsurface.

Como ilustra a Figura 31, a classe *DX2DSurface* foi criada com esse intuito. Ela especifica e encapsula o objeto geométrico primitivo que representa graficamente uma região NCL, além de oferecer as operações (como as especificadas pela interface *ISurface)* sobre a textura aplicada ao objeto. Como veremos mais adiante, essas serão as texturas onde, mais tarde, os exibidores farão a renderização do conteúdo referente ao objeto de mídia.

Para manter os objetos primitivos visíveis aos olhos do observador, a perspectiva principal é mantida perpendicular a cada objeto, fazendo com que somente a área de renderização da textura seja visível. Tal posicionamento tem implicação direta em um ponto importante da apresentação dos objetos de mídia: a sobreposição de regiões.

Em um documento NCL, uma região especifica sua ordem de precedência de sobreposição em relação às demais através do atributo *zIndex*. Assim, uma determinada região com maiores valores de *zIndex* deve ser obrigatoriamente empilhada no topo de regiões com valores de *zIndex* menores (ABNT, 2007).

De forma análoga ao mecanismo de *zIndex*, como visto no Apêndice , os objetos representados no plano em três dimensões podem estar mais próximos ou mais afastados da perspectiva do observador principal. Ao contrário do DirectFB, não existe uma pilha de superfícies em um ambiente em três dimensões, mas sim coordenadas espaciais, que devem ser respeitadas.

Dessa forma, a classe *DXWindow*, criada em substituição a classe *DFBWindow*, implementa um mecanismo de precedência baseado em um recurso chamado de *z-Buffer*, conforme apresentado na Seção 0,que permite realizar uma comparação pixel a pixel, para determinar qual janela deve preceder qual, independente da ordem de renderização atribuída durante o loop de renderização. O mecanismo se beneficia pelo fato de que o *z-Buffer* utiliza as coordenadas no eixo z com valores de 0.0 até 1.0 como parâmetros para medir a profundidade. Assim, ela mantém um valor da profundidade atual, atribuindo ao valor associado a janela o incremento do seu valor, armazenando-o em uma tabela associativa.

Como a coordenada *z* dos vértices da estrutura que representa a região (*DX2DSurface*) passa a assumir valores entre zero e um. A discrepância na dimensão entre regiões é imperceptível ao observador, ou seja, se, por exemplo, duas regiões de igual dimensão, porém de *zIndex* diferentes, forem renderizadas em um mesmo contexto gráfico, o observador não perceberá nenhuma diferença na dimensão entre as duas regiões.

5.1.2. Exibidores

Diante da adaptação do *Gerente Gráfico* ao novo *backend*, torna-se necessário a adequação do modulo *Exibidores*, para que esses garantam a correta apresentação dos objetos de mídia. Além da apresentação, os exibidores devem lidar com a obtenção do conteúdo dos objetos de mídia. Nesta seção será abordado o método de apresentação utilizado pelos exibidores, de acordo com o novo *backend* gráfico, além de mecanismos para obtenção de conteúdo dos objetos de mídia.

Invariavelmente, a apresentação do conteúdo dos objetos de mídia ocorre da mesma forma para todos os exibidores contemplados, através do mapeamento de texturas. Conforme apresentado no Apêndice , o mapeamento de texturas em objetos primitivos ocorre durante o *loop* de renderização, durante o qual, para cada objeto primitivo retangular, representando uma região NCL

(*DXSurface*), é aplicada uma textura contendo a unidade de informação do objeto de mídia. Apesar da generalização na apresentação, os exibidores podem obter seu conteúdo de forma direta ou através de biblioteca de terceiros (como o *DirectShow* ou uma outra biblioteca qualquer).

Como aprestado no Apêndice , na utilização do *DirectShow* como forma de obter o conteúdo de um objeto de mídia, o filtro VMR9 é utilizado para a renderização junto ao novo motor do *backend* gráfico, o Direct3D, como visto na Subseção 5.1.1. Portanto, de acordo com o processo de renderização descrito a cima, o filtro VMR9 deve renderizar o conteúdo do objeto de mídia em uma textura particular, que será aplicada posteriormente a um objeto primitivo particular dentro de um contexto tridimensional. Tais condições caracterizam o modo de apresentação *renderless*, como descrito no Apêndice , uma vez que não existem janelas, nem áreas de renderização em janelas da plataforma, mas sim a renderização direta em uma textura do Direct3D. De acordo com a configuração dos subcomponentes do VMR especificada no Apêndice , é necessário fornecer um filtro renderizador VMR com dois subcomponentes, o *Allocator* e o *Presenter* customizados.

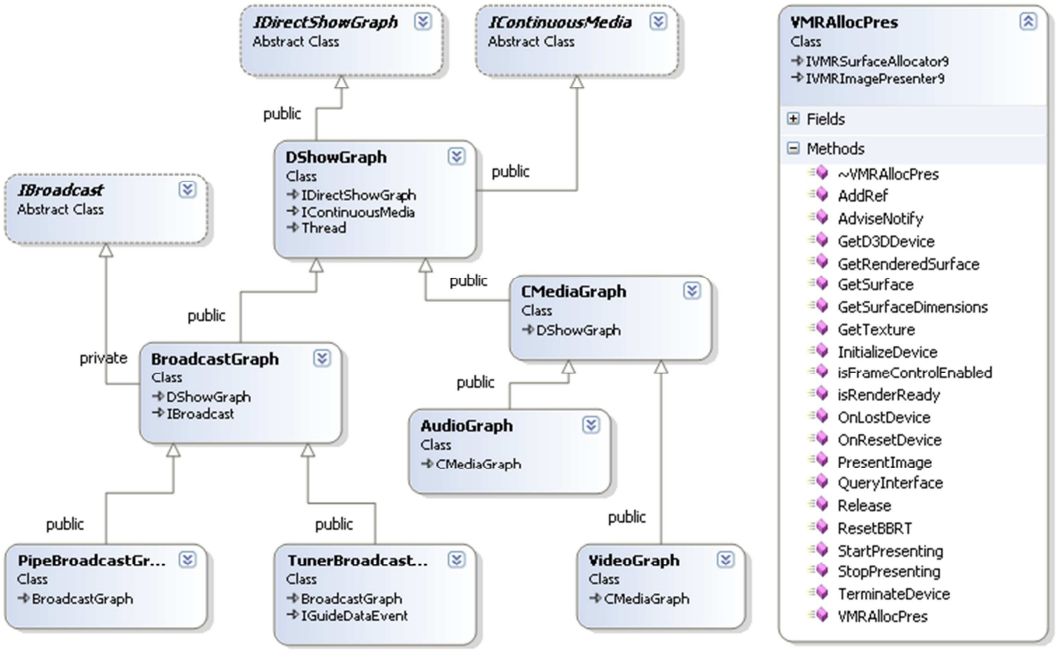

Figura 32 – Classes da biblioteca DshowGraphLib.

Como ilustra a Figura 32, a classe *VMRAllocPres* fornece a customização necessária ao filtro VMR9, através das interfaces *IVMRSufaceAllocator9* e *IVMRImagePresenter9*. Para facilitar a construção dos Grafos de Filtro, a biblioteca DshowGraphLib implementa dois tipos de grafo. O primeiro tipo especifica grafos para objetos de mídia continua (*CMediaGraph*), como áudio

(*AudioGraph*) e vídeo (*VideoGraph*). Já o segundo tipo especifica grafos para captura de fluxo de transporte, que podem ocorrer via sintonizador (*TunerBoradCastGraph*) ou via arquivo, através de um mecanismo para comunicação entre processos, o *pipe* (*PipeBroadCastGraph*).

A forma direta para obtenção de conteúdo dos objetos de mídia pode ser feita utilizando bibliotecas de terceiros, como mencionado. Todos os dados são obtidos dos objetos de mídia e copiados para região de memória da textura de renderização, que será aplicada ao objeto primitivo. Isso ocorre através de chamadas especificas do Direct3D para copia entre texturas. Neste trabalho alguns exibidores fizeram uso desse artifício para a apresentação dos objetos de mídia, como os exibidores de objeto de mídia para HTML, Lua e Imagem.

5.1.2.1. Áudio

O exibidor de áudio, através de seu provedor de conteúdo *DXAudioProvider*, utiliza o DirectShow para obter o conteúdo do objeto de mídia a ser apresentado. Para tal ele utiliza a biblioteca DshowGraphLib, através da classe *AudioGraph*, que constrói programaticamente o grafo de filtros correspondente. Tal como ilustrado a Figura 33.

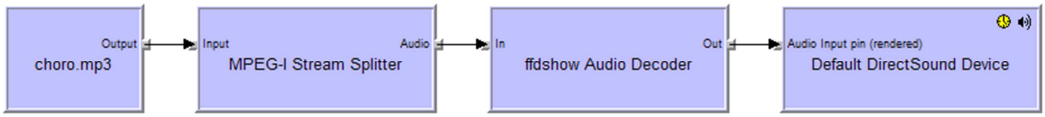

Figura 33 – Grafo de filtros para reprodução de áudio.

Como não existe uma representação visual para objeto de mídia do tipo áudio, o grafo construído não utiliza o filtro VMR9 para renderização, mas o faz diretamente através do dispositivo de áudio disponível.

5.1.2.2. Vídeo

O exibidor de vídeo, através de seu provedor de conteúdo *DXVideoProvider*, utiliza o DirectShow para obter o conteúdo do objeto de mídia a ser apresentado. Para tal, ele utiliza a biblioteca DshowGraphLib, através da classe *VideoGraph*, que constrói programaticamente o grafo de filtros correspondente. Tal como ilustrado a Figura 34.

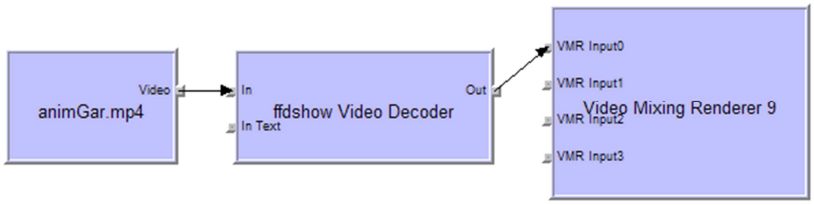

Figura 34 – Grafo de filtros para reprodução de vídeo.

Ao contrário do exibidor de áudio, existe uma representação visual para o objeto de mídia do tipo vídeo. Dessa forma, o grafo construído utiliza o filtro VMR9 para realizar a renderização, preenchendo a textura, a ser mapeada pelo motor gráfico, com o conteúdo obtido do objeto de mídia.

5.1.2.3. HTML

O exibidor de objetos do tipo HTML não apresenta um provedor especificado pela implementação de referência, utilizando uma biblioteca de terceiro, a *Awesomium* (KHRONA, 2010) para obter o conteúdo do objeto de mídia, como ilustra a Figura 35.

Figura 35 – Provedor HTML baseado no Awesomium.

Ao contrário das demais implementações dos exibidores apresentados até o momento, o exibidor HTML não possui um Grafo especifico para obter o conteúdo do objeto de mídia. Nele, o preenchimento da textura ocorre de forma direta. Uma vez a extraída a amostra de conteúdo, ela é copiada diretamente para a textura, através das funções de cópia do Direct3D (como a *D3DtextureCopy*).

5.1.2.4. Texto

O exibidor de texto, através de seu provedor de conteúdo (*DXFontProvider*), utiliza a biblioteca de baixo nível, chamada GDI (*Graphics Device Interface*), para gerar o texto referente ao objeto de mídia, como ilustra a Figura 36.

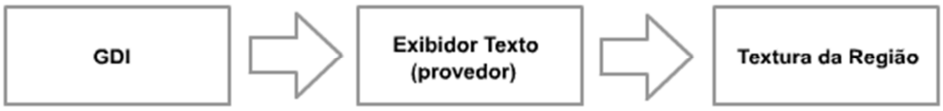

Figura 36 – Provedor Texto baseado no GDI.

Tal como acontece com o exibidor de HTML, esse exibidor não possui um filtro do DirectShow para obter e renderizar o conteúdo do objeto de mídia. Dessa forma, é realizada uma chamada a função *DrawTextW* da API especificada pelo GDI para realizar a renderização do texto diretamente para a textura associada a região.

5.1.2.5. Lua

O exibidor Lua não apresenta um provedor especificado pela implementação de referência, utilizando as próprias rotinas para desenho fornecidas pela abstração de superfícies e janelas (*DXSurface* e *DXWindow*), tal qual foi apresentado na Subseção 5.1.1.

Figura 37 – Renderização do Lua *Canvas*.

Diferente do acontece em todos os exibidores, esse exibidor gera o conteúdo a ser apresentado em tempo de apresentação. Isso ocorre devido à presença de um componente *Canvas* para desenho em superfícies. Dessa forma, as operações de desenho são realizadas diretamente na textura associada à superfície.

5.1.2.6. Imagem

O exibidor de imagens, através de seu provedor de conteúdo (*DXImageProvider*),utiliza uma biblioteca de terceiro, a *FreeImage* (FREEIMAGE, 2010) para obter o conteúdo do objeto de mídia, como ilustra a Figura 38.

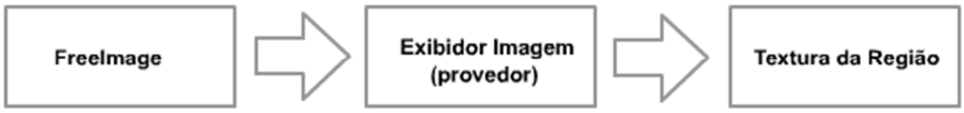

Figura 38 – Renderização do FreeImage.

Tal como acontece com o exibidor de HTML, esse exibidor não possui um filtro do DirectShow para obter e renderizar o conteúdo do objeto de mídia. Uma vez gerada a imagem contendo o texto, ela é copiada para a textura através das funções de cópia do Direct3D (como a *DXLoadTextureFromImage* e função *D3DTextureCopy*).

5.1.3. Sintonizador

O módulo sintonizador ISDB-T utiliza o DirectShow para capturar o fluxo de transporte e extrair o fluxo elementar de áudio e vídeo, além das aplicações interativas. Para tal ele utiliza a biblioteca DshowGraphLib, com a qual é possível criar grafos para obter o fluxo de transporte do dispositivo sintonizador (*TunerBroadcastGraph*) como de um arquivo (*PipeBroadcastGraph*), como ilustra a Figura 39.

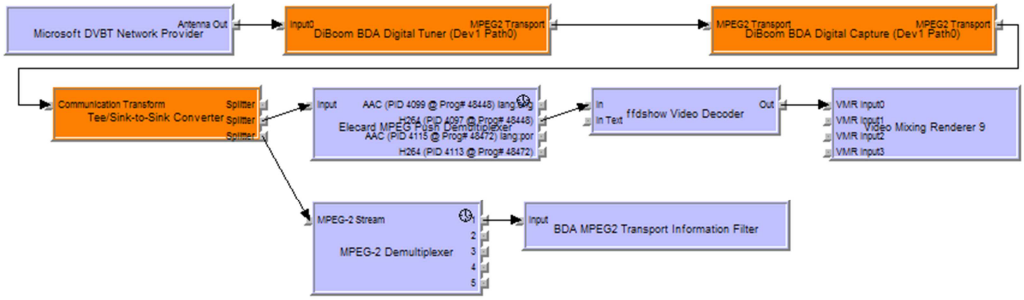

Figura 39 – Grafo de Filtros para sintonização ISDB-T.

Para obter o fluxo de transporte de um arquivo do tipo *Pipe*, foi criado um filtro cliente, da categoria origem. Independente da origem do conteúdo, a renderização do fluxo elementar obtido ocorre de forma análoga à apresentada no exibidor de áudio e vídeo, apresentados anteriormente.

5.1.4. Dispositivos de entrada

A implementação de referência do Ginga-NCL especifica interfaces para a abstração do tratamento de eventos emitidos pelos dispositivos de entrada (como controle remoto, mouse e teclado).

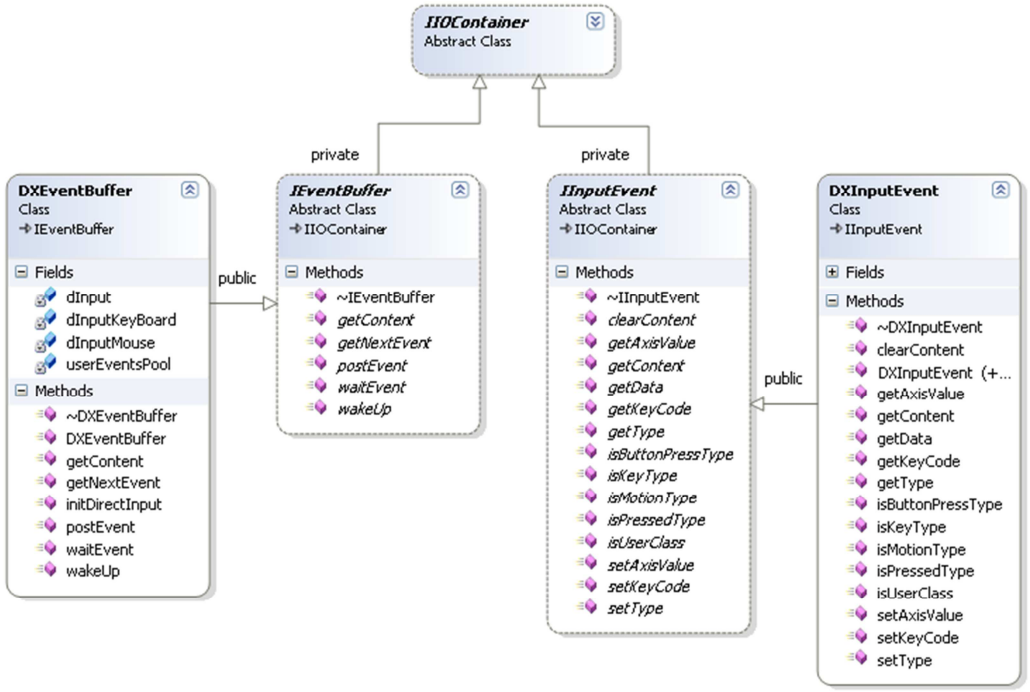

Figura 40 – Classes dos dispositivos de entrada.

A Figura 40 apresenta o diagrama de classes que define as interfaces que especificam a captura (*IEventBuffer*) e o encapsulamento (*IInputEvent*) de eventos gerados pelos dispositivos de entrada. A biblioteca Directinput, como apresentado no Apêndice , foi utilizada para a captura de eventos no contexto do novo *backend* gráfico. Portanto, em substituição as classes *DFBEventBuffer* e *DFBInputEvent*, foram implementadas duas classes *DXEventBuffer* e *DXInputEvent*, respectivamente.

A classe *DXEventBuffer* realiza a captura de eventos através do *pooling* dos dispositivos de entrada. Na ocorrência de um evento a classe DXInputEvent é utilizada para encapsular o evento do DirectInput (*DIDEVICEOBJECTDATA*). Existe ainda um tipo especial de evento utilizado pelo exibidor lua em sua máquina de eventos, o evento do tipo usuário. Para tal tipo de evento, foi criado um *buffer* de eventos simples (*userEventPool*), que contém estruturas de evento particular ao DirectInput, porém, customizadas.

5.2. Middleware Ginga-NCL como Plugin para o Navegador Mozilla Firefox

De acordo com o que foi apresentado na Seção 4.2.1, o nível de integração entre o *middleware* Ginga-NCL e o navegador Web abrange desde a adaptação do *middleware* como um *plugin* reprodutor hipermídia até a

comunicação entre ambos. Assim, a Subseção 5.2.1 apresenta o Desenvolvimento necessário para torná-lo um reprodutor hipermídia para Web e a Subseção 5.2.2 apresenta o desenvolvimento necessário para a integração, no que concerne o *controle de apresentação, a sincronização* e a *edição em tempo de exibição*.

5.2.1. Adaptação à Arquitetura de Plugins

O primeiro passo visando à integração consiste em adaptar o *backend* gráfico ao contexto de apresentação do navegador, permitindo a inserção de objetos de mídia do tipo NCL em uma página Web, como descrito na Seção 4.1. Inicialmente, como especificado em 3.3.1, a NPAPI define pontos de interface com os quais é possível instanciar o *plugin*.

A classe *GingaNCLPluginInstance*, como ilustra a Figura 41 abaixo, foi criada para permitir o encapsulamento da instância do *middleware* Ginga-NCL. A classe em questão é responsável pela configuração, inicialização, instanciação e término da máquina de apresentação do *middleware* Ginga-NCL. A rotina NPP_New, definida pelo navegador, é encarregada de instanciar a classe *GingaNCLPluginInstance*, passando como parâmetros o *Mime type* associado, a referência para a instância do *plugin*, o modo e os atributos especificados pelo elemento <embed>. Após a inicialização do *plugin*, e consequentemente a inicialização do *middleware* Ginga-NCL, o navegador realiza a inicialização gráfica.

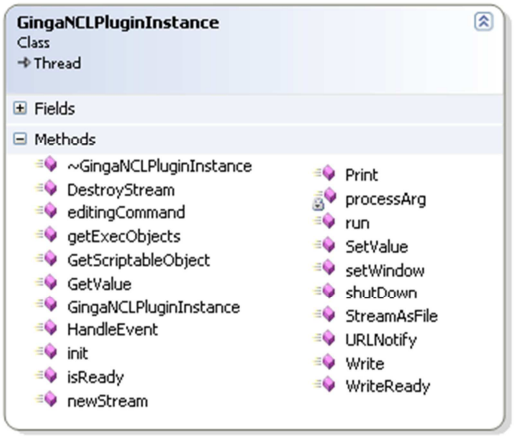

Figura 41 – Classe *GingaNCLPluginInstance*.

Ainda de acordo com a Figura 41, como visto na Subseção 3.3.3, o *plugin* especifica o modo de apresentação *windowed* de modo que tenha total controle

sobre os eventos e o processo de renderização. Dessa forma, o navegador invoca, através da interface *NPP_SetWindow*, o método *setWindow* da classe *GingaNCLPluginInstance*, passando como parâmetro o *handler* da janela onde ocorrerá a renderização.

Devido ao fato que no âmbito de um *plugin* Mozilla, cujo modo de apresentação adotado é o *windowed*, a criação da tela para renderização é feita pelo navegador, para que o modulo *Gerente Gráfico* inicialize o motor gráfico, de acordo com a área de renderização especificada, foi previsto a inicialização da classe *DXDeviceScreen*, recebendo como parâmetros a dimensão e a referência para *handle* nativo da área de renderização.

5.2.2. Integração entre o Navegador e o Middleware Ginga

Após a adequação do *middleware* como um *plugin* Mozilla para reprodução de uma aplicação NCL, um novo passo é tomado em direção à integração, tal qual especificada na Seção 4.2.1. Desse modo, o *plugin* implementa um mecanismo de comunicação nos dois sentidos, tanto do navegador em direção ao *middleware*, quanto do *middleware* em direção ao navegador.

A comunicação no sentido *middleware* para navegador é alcançada através da API disponibilizada pelo navegador, que permite ao código nativo enxergar o código JavaScript, como apresentado na Subseção 3.3.2. Através das funções *NPN_GetProperty* e *NPN_GetValue*, o *plugin* obtém a referência para o Documento DOM da página Web, o que lhe permite manipular todo o documento HTML no qual está inserido. Cabe ressaltar que a comunicação nesse sentido ocorrerá somente mediante autorização do navegador (usuário).

Como apresentado na Subseção 3.3.2, a cada instância do *plugin* está associado um objeto para especificação de interfaces nativas em JavaScript. Para que ocorra a comunicação no sentido navegador para *middleware* é necessário que o *middleware* torne externo a interface a ser mapeada. Nesse ponto é interessante destacar que todos os exibidores possuem uma mesma interface, o que permite fazer uma generalização no que diz respeito a interface a que deve ser transparente a pagina Web que utilize o *plugin*.

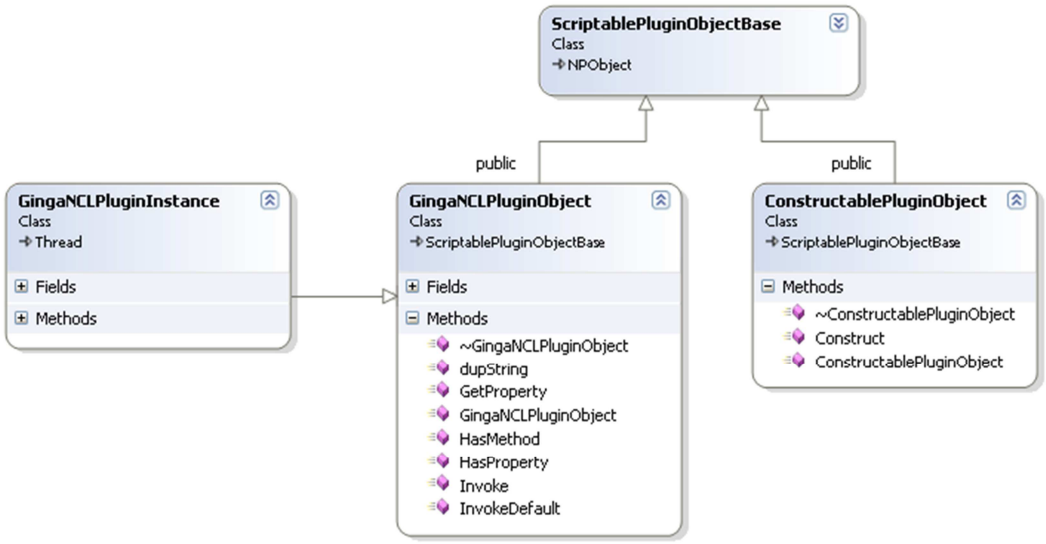

Figura 42 – Classe *GingaNCLPluginObject* associada ao objeto *scriptable*.

De acordo com o diagrama da Figura 42, com o propósito de tornar externo as interfaces do *middleware* Ginga-NCL, foi criada a classe *GingaNCLPluginObject*. Quando forem especificados métodos ou variáveis dentro do escopo definido pelo objeto DOM, que está vinculado ao elemento <embed>, a classe *GingaNCLPluginObject* estará encarregada de fazer o mapeamento entre as linguagens.

Como mencionado anteriormente, o *middleware* Ginga-NCL especifica uma interface (*IPlayer*) comum a todos os componentes exibidores. Uma aplicação NCL, quando inserida como objeto de mídia declarativo em outra aplicação NCL, utiliza um exibidor próprio para a apresentação. Tal exibidor é especificado através da interface (*INCLPlayer*), que por sua vez estende a interface (*IPlayer*).

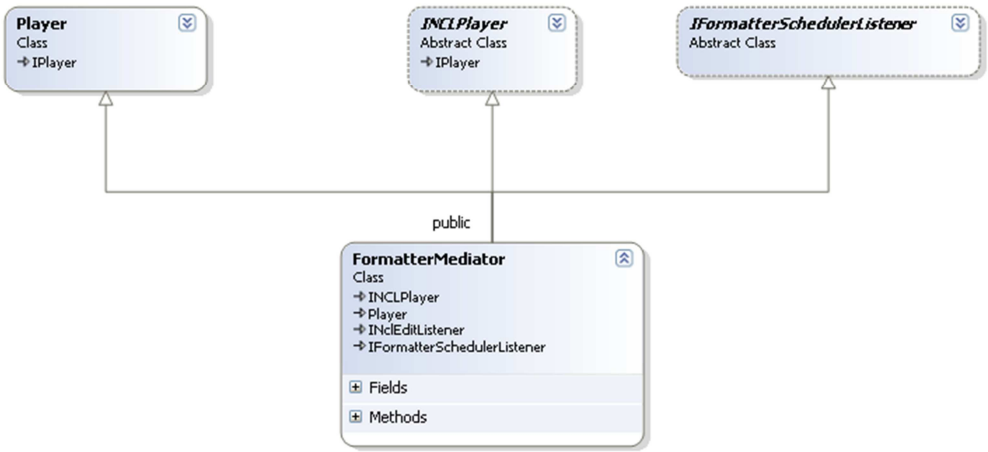

Figura 43 – Classe FormatterMediator.

De acordo com o diagrama da Figura 43, o módulo *Formatador*, representado pela classe *FormatterMediator*, nada mais é que um exibidor de aplicações NCL. Como tal, ele implementa a interface *INCLPlayer* como os outros exibidores. Ademais, estende a classe *Player*, que implementa um exibidor básico.

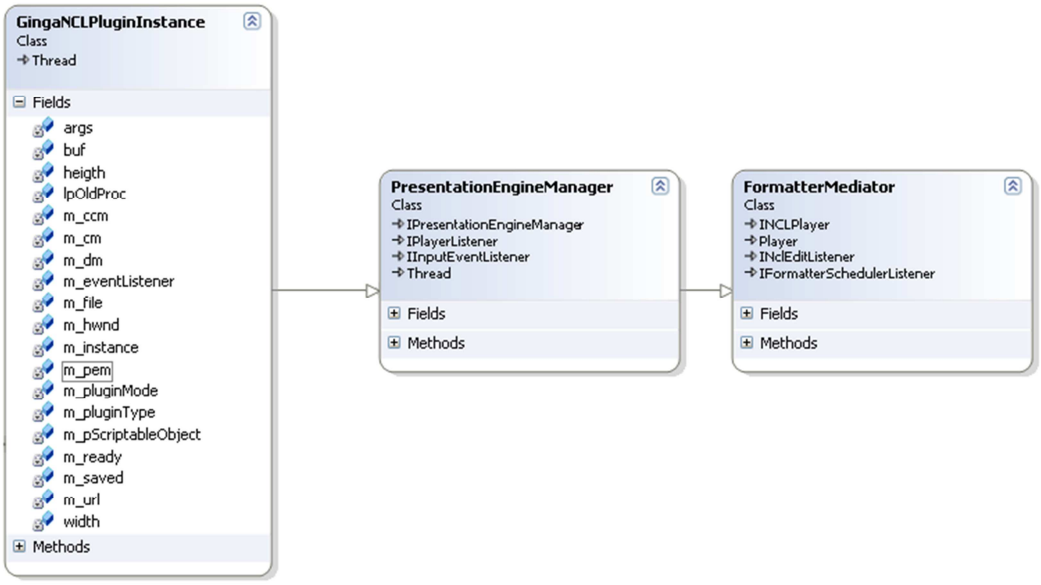

Figura 44 – Classe FormatterMediator como Interface externa do *plugin*.

Como ilustra o diagrama da Figura 44, a classe *FormatterMediator* não representa um ponto de interface direto com o *plugin*. Porém, a classe *PresentationEngineManager* é utilizada com esse propósito, permitindo que o *plugin* tenha acesso ao exibidor da aplicação indiretamente. Essa interface é crucial para o suporte a especificação de *âncoras de conteúdo* e *âncoras de propriedades*, controle de apresentação da aplicação e edição em tempo de exibição.

Para a especificação de *âncoras de conteúdo* e *âncoras de propriedades* a classe *PresentationEngineManager* foi modificada para que possa registrar funções *listeners* para a notificação de eventos NCL (*apresentação, seleção e atribuição*) ao contexto mais externo. Uma vez visível, a interface para a criação de âncoras, representada pelo método *invoke* da classe *GingaNCLPluginObject,* é chamada, tendo o parâmetro *name* com valor igual ao nome da interface NCM e os demais com informações sobre a âncora. Entre esses parâmetros, está a função para tratamento de eventos provenientes daquela âncora em particular, que é registrada junto *PresentationEngineManager*, através de um método *listener*. Em contrapartida, para a notificação de eventos ocorridos no contexto

Web, é disponibilizada uma rotina para notificação direta de eventos relacionados à âncora em particular.

A especificação de uma interface para o controle na apresentação da aplicação é feita por meio dos métodos *startPresentation*, *pausePresentation* e *stopPresentation*, que pertencem a classe *PresentationEngineManager*. Assim, quando o método *invoke,* especificado pela classe *GingaNCLPluginObject,* é chamado, tendo o parâmetro *name* com valor igual a uma dessas interfaces, o método equivalente da classe *PresentationEngineManager* é chamado e o valor de retorno atribuído.

Para a especificação da interface de edição ao vivo, é necessário a tornar transparente o método *editingCommand*, que pertence à classe *PresentationEngineManager*. Portanto, os comandos de edição especificados em código JavaScript podem ser traduzidos em comandos de edição NCL. Os comandos de edição são, originalmente, transportados, no SBTVD, através de seções DSMCC, contudo, para a submissão de comandos de edição segundo o contexto deste trabalho, não é necessário encapsulá-los dessa forma. Assim, foi realizada uma modificação no método *editingCommand* da classe *PresentationEngineManager* para que os comandos de edição possam ser recebidos independente do meio de transporte adotado. O que permite até mesmo vislumbrar o transporte através de serviços assíncronos para Web, como o Ajax (*Asynchronous JavaScript and XML*).