6
Implementação

Visando testar as propostas apresentadas nos capítulos anteriores, foram realizadas modificações na Implementação de Referência do Ginga-NCL (COMUNIDADE GINGA-NCL, 2010) – a partir deste ponto, denominada apenas de Implementação de Referência. Este capítulo apresenta os principais módulos que foram modificados ou estendidos por este trabalho, conforme ressaltados na Figura 57.

Figura 57 Arquitetura da Implementação de Referência do Ginga-NCL evidenciando os pontos que foram modificados por esta dissertação. Adaptado de (MORENO, 2010).

A princípio, e visando testar as propostas do Capítulo 4, um novo adaptador e um novo exibidor para objeto de mídia declarativos em X3D foram adicionados, conforme será discutido na Seção 6.1, sem mudanças mais profundas na Implementação de Referência.

Como uma segunda etapa, e visando preparar o ambiente para permitir a implementação das propostas do Capítulo 5, todo o *backend* de renderização precisou ser modificado para trabalhar com OpenGL, resultando em alterações no

Gerente Gráfico e nos exibidores dos tipos de mídia, que já utilizavam DirectFB nativamente.

Por fim, adaptações no Gerente de Leiaute, no Parser XML e nos Conversores, permitiram a implementação das transformações 2D e 3D propostas no Capítulo 5.

O restante deste capítulo detalha o trabalho de implementação realizado no escopo desta dissertação e está dividido como segue. A Seção 6.1 discute a implementação do exibidor X3D que foi acoplado a Implementação de Referência. Enquanto a Seção 6.2 apresenta as modificações necessárias para permitir as transformações discutidas no Capítulo 5.

6.1.Exibidor X3D no Ginga-NCL

A Implementação de Referência é atualmente desenvolvida em C++. Ela foi desenvolvida de forma a facilmente integrar exibidores para novos tipos de mídia. Para isso, ela define componentes *Exibidores*. Conforme discutido na Seção 3.2, o *Exibidor* é responsável por apresentar um determinado tipo de objeto de mídia específico e deve implementar uma API padrão. Essa API permite a comunicação entre o Exibidor e a máquina de apresentação Ginga-NCL.

Os Exibidores são responsáveis por notificar a máquina de apresentação sobre eventos que ocorrem no objeto de mídia que ele está apresentando, tais como a ocorrência de alguma âncora (início ou fim da apresentação), a seleção de uma âncora, a alteração de alguma propriedade do nó de mídia, entre outros. Exibidores de objetos de mídia de terceiros que não seguem a especificação da API Ginga devem se utilizar de componentes *Adaptadores* para fazer essa mediação.

Visando testar a proposta apresentada no Capítulo 5, a Implementação de Referência do Ginga-NCL foi estendida neste trabalho e um novo Exibidor, capaz de exibir apropriadamente objetos de mídia definidos em X3D, foi acoplado a ela. Esse Exibidor é baseado no FreeWRL (COMUNIDADE FREEWRL, 2010), um *player* de documentos VRML/X3D implementado em ANSI C e disponível como código livre.

Um novo Adaptador também se fez necessário para manter a compatibilidade com a API Ginga-NCL e tratar a notificação dos novos eventos propostos acima, bem como a definição de âncoras nesses objetos de mídia.

Esta seção discute os principais pontos dessa implementação e apresenta as dificuldades encontradas.

6.1.1.Modelagem

Conforme discutido acima, o Ginga-NCL define uma API comum para todos os componentes Exibidores. Na Implementação de Referência, tais componentes devem implementar a interface *IPlayer*, disponível na Implementação de Referência do Ginga-NCL. Adicionalmente, também está disponível por meio da classe *Player* uma implementação de um Exibidor básico, que já possui a implementação de funções básicas como *addListener*, *removeListener*, *getPropertyValue* etc.

Neste trabalho, foi adicionado um novo componente, o *X3DPlayer*, que estende a classe *Player* e é responsável por instanciar, exibir e notificar eventos que acontecem durante a exibição de um documento X3D, para a máquina de apresentação do Ginga-NCL. A Figura 58 apresenta o diagrama de classes do componente X3DPlayer.

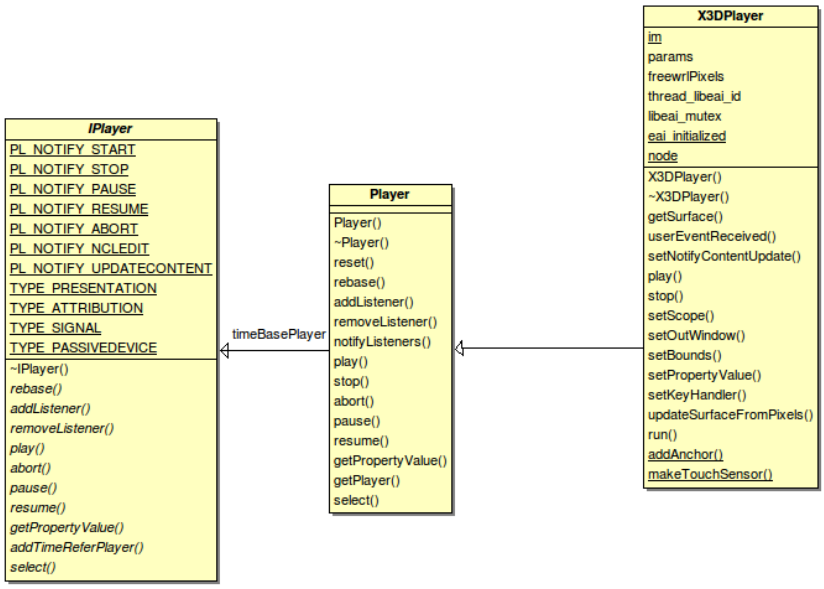

Figura 58 Diagrama de Classe Exibidor X3D adicionado à Implementação de Referência do Ginga-NCL.

A implementação da classe *X3DPlayer* tira proveito das funcionalidades da EAI (*External Access Interface*), disponibilizada pelo FreeWRL. A SAI não é utilizada porque o FreeWRL não possui uma implementação em C/C++ para ela. Algumas mudanças internas no código fonte desse *player* também foram necessárias, como é discutido na subseção seguinte.

Um novo Adaptador também foi adicionado por este trabalho, denominado *X3DPlayerAdapter*. Adaptadores devem implementar a API *IPlayerAdapter*. Da mesma forma que a classe *Player*, também existe uma classe padrão que implementa a API de Adaptadores implementando as funções básicas: o *FormatterPlayerAdapter*. O *ApplicationPlayerAdapter* é uma classe que agrupa as principais funcionalidades para tipos de mídia representados por códigos imperativos ou declarativos, tais como HTML, Lua, Java etc., estendendo a classe *FormatterPlayerAdapter*. Objetos de mídia X3D são objetos de mídia declarativos. Sendo assim, o *X3DPlayerAdapter*, adicionado por este trabalho, estende a classe *ApplicationPlayerAdapter*, conforme apresentado na Figura 59.

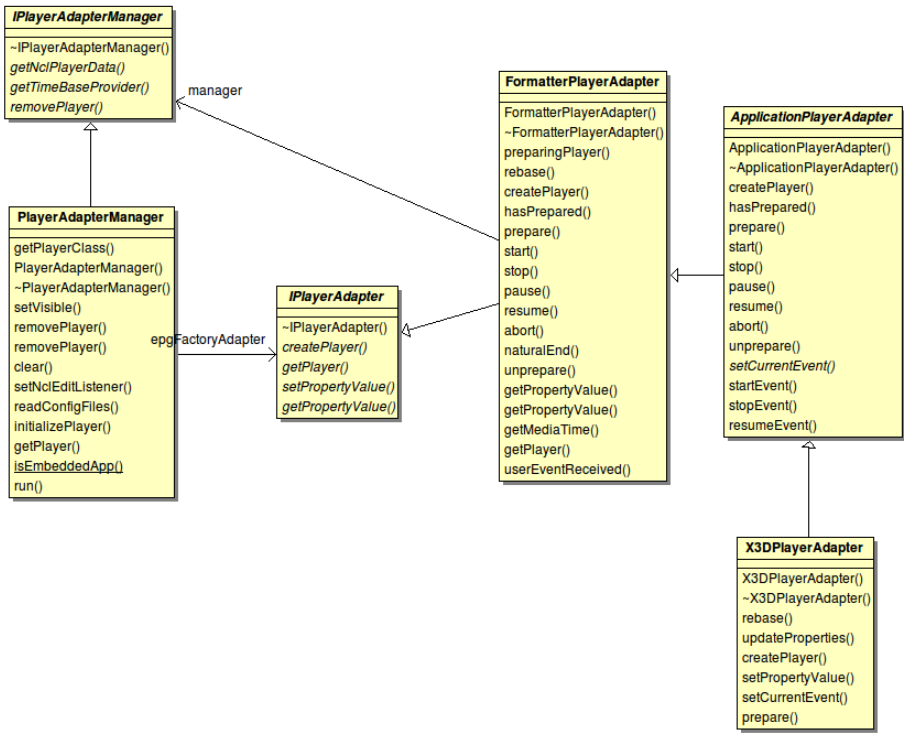

Figura 59 Diagrama de Classe Adaptador X3D adicionado à Implementação de Referência do Ginga-NCL.

Além de permitir a criação e manipulação do grafo de cena, a EAI possibilita também o registro de *callbacks* para eventos que acontecem enquanto o

grafo de cena está em execução. Em especial, neste trabalho, estamos interessados nos eventos reconhecidos por NCL: *apresentação*, *seleção*, *atribuição*, *colisão*, *visibilidade* e *proximidade* (esses três últimos adicionados por esta dissertação). Para serem notificados, entretanto, alguns desses eventos necessitam de nós de sensores. Com o objetivo de tirar essa carga cognitiva dos autores de documentos NCL com objetos X3D embutidos, o X3DPlayer, baseado em informações enviadas pelo X3DPlayerAdapter, cria os sensores necessários em tempo de execução.

Quais os sensores necessários podem ser deduzidos a partir dos elos que fazem referência a suas âncoras de conteúdo. Eventualmente, caso o autor defina uma âncora de conteúdo diretamente para um nó de sensor (como discutido na Subseção 4.1.1) não existe a necessidade da criação dinâmica desses nós de sensores.

A Figura 60 apresenta um exemplo de como é possível criar dinamicamente nós de sensores baseado nos elos existentes no documento. Neste exemplo, existe um elo que utiliza o evento de seleção (uma condição *onSelection*). Por isso, deve-se criar um nó de sensor de toque (o nó *TouchSensor* de X3D) nesse objeto (conforme exemplifica a Figura 60). O mesmo também é válido para os eventos de colisão (*CollisionSensor*), visibilidade (*VisibilitySensor*) e proximidade (*ProximitySensor*). Os eventos de apresentação e atribuição não necessitam de sensores, já que podem ser realizados apenas por meio do controle dos atributos dos nós X3D.

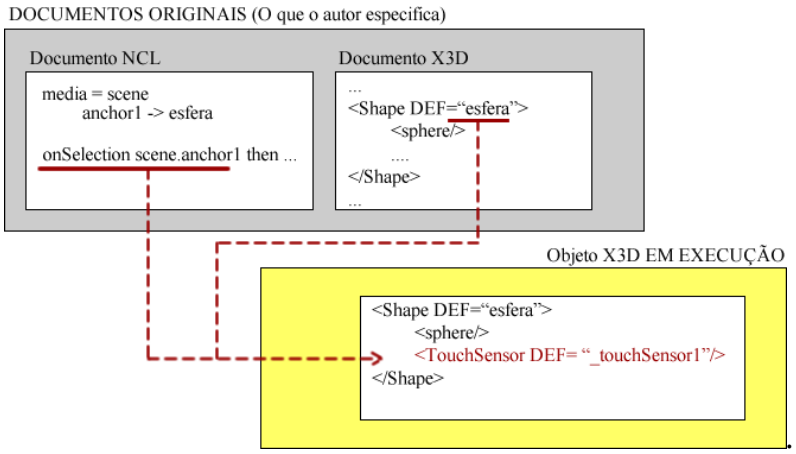

Figura 60 Geração de nós de sensores automaticamente pelo X3DPlayer.

A Figura 61 apresenta o digrama de classe dos eventos de colisão, proximidade e visibilidade adicionados nesta dissertação.

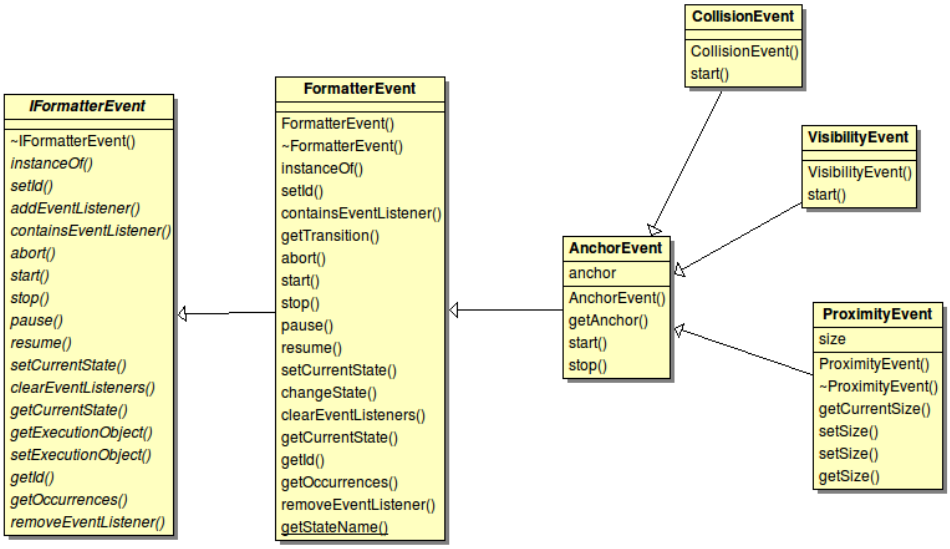

Figura 61 Diagrama de classe dos novos eventos adicionados nesta dissertação.

6.1.2.FreeWRL

Além das mudanças discutidas acima, na Implementação de Referência do Ginga-NCL, também foram necessárias modificações internas no código fonte do FreeWRL. Essas alterações resultaram em uma nova biblioteca, denominada *telemidia-freewrl* e que também está disponível como código livre. As alterações no código fonte do FreeWRL também foram encaminhadas à sua comunidade de desenvolvimento, a qual pode absorvê-las no código principal. As principais mudanças necessárias foram:

- **Camada de abstração para a notificação de eventos de interação do usuário**

Existem versões do FreeWRL para Windows, Linux e MAC OS. Cada uma delas utiliza código nativo do sistema operacional para recebimento de eventos do usuário. Neste trabalho, foi desenvolvida uma camada intermediária entre o *player* e o sistema operacional que permite enviar eventos do usuário para a máquina de exibição X3D. Dessa forma, os eventos de interação recebidos pelo Ginga-NCL podem ser enviados para o FreeWRL, sem dependência do sistema operacional. Eventualmente, agora também é possível simular a interação do usuário através do envio desses eventos de forma programática, o que facilita a criação de testes automáticos.

- **Renderização *OffScreen***

Antes deste trabalho, o FreeWRL não permitia a renderização da cena em memória (*offscreen*). Contudo, essa funcionalidade é vital para embuti-lo como exibidor de um novo tipo de mídia em NCL, já que o Ginga-NCL necessita do resultado da renderização do grafo de cena, para só então compor a apresentação NCL. Por esse motivo, o FreeWRL foi adaptado para permitir a renderização em memória. A implementação realizada baseou-se na API de renderização do X11, que tem a desvantagem de não ser multiplataforma. Porém, como a própria Implementação de Referência do Ginga-NCL atual é dependente do sistema operacional Linux, tal desvantagem é tolerada.

Outro problema identificado com a API X11 foi o seu baixo desempenho, devido principalmente a constante mudança de contexto OpenGL entre o FreeWRL e o Ginga-NCL. Uma nova implementação baseada em objetos de *Framebuffers* (FBO), extensão de OpenGL que permite dentre outras coisas a renderização offscreen em texturas, deve ser realizada no futuro.

- **Adaptações na implementação da EAI**

Algumas adaptações na implementação da EAI também foram necessárias durante este trabalho. Em especial, as alterações focaram em permitir a correta identificação de qual sensor, adicionado pelo Ginga-NCL, gera os eventos durante a execução do grafo de cena.

Anteriormente, o FreeWRL permitia registrar uma *callback* em um evento de um nó do grafo de cena, por meio da função *X3DAdvise(EventOut *event, void *callback)*. Essa função recebia como parâmetro qual o evento de saída de um nó de mídia que se deseja "escutar" e um ponteiro para uma função que deve ter a interface: *void callback(void*)*. A função *callback* é chamada pelo FreeWRL sempre que uma alteração no evento *event* ocorrer, passando como parâmetro para essa callback o novo valor. Dessa forma, se uma mesma função fosse registrada para mais de um evento, não existia nenhum mecanismo para identificar qual dos eventos estaria chamando essa *callback*. Identificar qual o evento que realmente chamou a função de *callback* é extremament útil quando geramos novos sensores dinamicamente. Isso porque, não é possível criar novas funções – observe que estamos falando de funções e *não* ponteiros para funções – dinamicamente em C.

Neste trabalho, foi adicionado um valor de retorno à função X3DAdvise que retorna um inteiro que identifica univocamente o registro de uma *callback*. Cada vez que a função *X3DAdvise* é chamada, ela deve retornar um valor diferente. Adicionalmente, a função *callback* agora deve implementar a interface: "*void callback(int, void*)*". Sendo que o primeiro parâmetro informa qual o identificador do registro daquela *callback*. Dessa forma, é possível que uma mesma função seja registrada para mais de um evento e a correta identificação quando aquela *callback* for chamada.

6.2.Implementação de Referência do Ginga-NCL baseada em SDL e OpenGL

Visando possibilitar a definição de regiões NCL como superfície de objetos 3D, um importante esforço demandado por este trabalho foi a modificação do *backend* da Implementação de Referência. Antes desta dissertação, a Implementação de Referência era baseada exclusivamente no DirectFB (DIRECTFB, 2010), que, atualmente, não dá suporte satisfatório para gráficos 3D. No escopo deste trabalho, foram realizadas diversas mudanças que permitem escolher, baseado no modelo de componentes definido pela própria Implementação de Referência (MORENO, 2010), entre a execução utilizando DirectFB (que não dá suporte às transformações nas regiões aqui apresentadas) ou utilizando SDL (*Simple DirectMedia Layer*) (SDL, 2010).

SDL é uma biblioteca multiplataforma desenvolvida inicialmente para a criação de jogos 2D. Entretanto, permite a utilização de comandos OpenGL diretamente. Nas mudanças introduzidas por este trabalho, SDL é utilizada principalmente para criar a janela principal e tratar os eventos do usuário, enquanto toda a parte de desenhos gráficos fica a cargo de OpenGL. Todas as transformações de regiões discutidas no Capítulo 5 tornam-se bem mais simples de se desenvolver usando OpenGL diretamente. Isso porque, tais transformações podem ser implementadas apenas com chamadas simples à API OpenGL, tais como *glTranslatef* e *glRotatef*.

A Implementação de Referência do Ginga-NCL já possui interfaces para a abstração do tratamento de eventos do usuário, sendo que antes desse trabalho, só existia implementação dessas classes para o DirectFB. A Figura 62 apresenta o

diagrama de classes que define as classes que fazem o mapeamento entre os eventos gerados pelo usuário via SDL em eventos que são reconhecidos pela Implementação de Referência, por meio da interface *IInputEvent*.

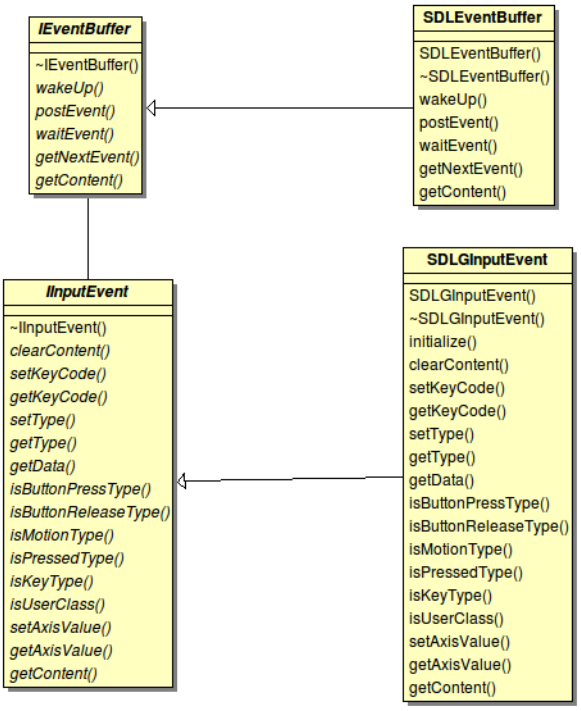

Figura 62 Diagrama de classes evidenciando a interface de eventos do usuário, disponível pela Implementação de Referência, e as novas implementações dessas interfaces baseadas em SDL.

A Implementação de Referência define uma única interface para tratar todas as telas de dispositivos, inclusive dispositivos secundários: *IDeviceScreen*. As classes que implementam essa interface são responsáveis, dentre outras coisas, por instanciar a janela de exibição, definir a resolução do dispositivo, entre outras. Em especial, quando estamos trabalhando com OpenGL, essa classe também é responsável por iniciar o contexto OpenGL, habilitando o tratamento de texturas, definindo parâmetros de iluminação e câmera etc.

A Figura 63 apresenta o diagrama de classes da classe *SDLDeviceScreen*, adicionada por este trabalho, como uma alternativa de execução à classe *DFBDeviceScreen* utilizada anteriormente.

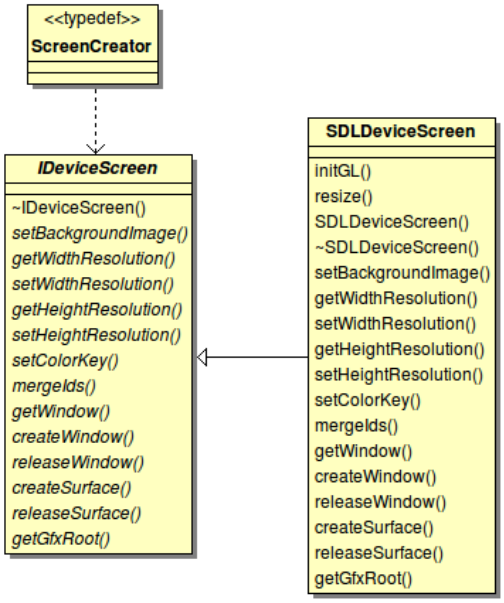

Figura 63 Diagrama de classes evidenciando a interface responsável por abstrair as funções da tela do dispositivo e implementação em SDL.

Uma Superfície (interface *ISurface*) é a abstração utilizada pela Implementação de Referência do Ginga-NCL para representar uma região NCL – o conceito de região aqui é o mesmo discutido no Capítulo 5, o conjunto de atributos relacionados à dimensão e ao posicionamento dos objetos de mídia. A própria SDL também provê uma abstração de superfície, a qual já possui boa parte dos métodos necessários para desenhos definidos na interface *ISurface* da Implementação de Referência. Adicionalmente, para dar suporte à implementação das transformações discutidas no Capítulo 5 a superfície SDL foi mapeada em uma textura OpenGL a qual pode ser aplicada em qualquer objeto 3D OpenGL.

A classe *SDLSurface*, apresentada no diagrama de classes da Figura 64, foi adicionada à Implementação de Referência como uma nova implementação da interface *ISurface*. Essa classe é responsável por tratar todas as transformações discutidas no Capítulo 5, inclusive mapeando o conteúdo a ser exibido na superfície de objetos 3D.

Adicionalmente, também foram necessárias modificações nos componentes XML Parser e Converter (discutidos na Seção 3.2) que possibilitam extrair essas informações do documento NCL.

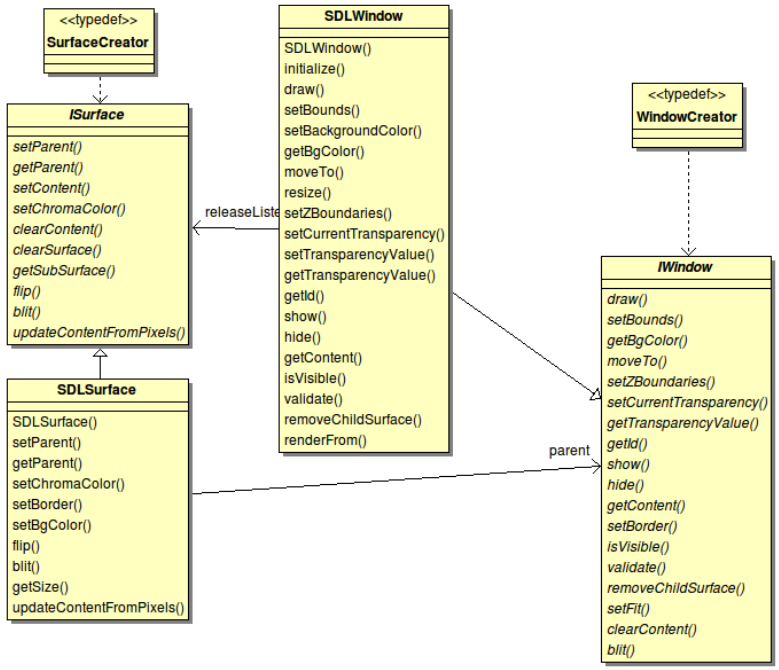

Figura 64 Diagrama de classe evidenciando a implementação de superfícies de renderização em SDL.

6.2.1.Configuração dos Componentes do Ginga-NCL para utilizar SDL e OpenGL como *backend*

A própria arquitetura de componentes (MORENO, 2010) da Implementação de Referência ajudou bastante o trabalho de implementar as mudanças discutidas acima. Isso porque, o trabalho teve foco principalmente em reimplementar, baseado em SDL e OpenGL, as interfaces definidas e em configurar o modelo de componentes para utilizar essas novas implementações. Como resultado, hoje é possível escolher, tanto no momento da compilação como no da execução, qual o *backend* de renderização que deve ser utilizado: DirectFB ou SDL+OpenGL.

O modelo de componentes da Implementação de Referência é baseado em um arquivo XML que descreve quais os componentes disponíveis, quais são suas interfaces e quais os métodos responsáveis pela criação e destruição de novos objetos baseados naqueles componentes. O arquivo de configuração é utilizado como base enquanto o Ginga-NCL está executando, para permitir a criação de novos objetos, e permite inclusive que os componentes sejam atualizados em tempo de execução, sem a necessidade de seu reinício. Adicionalmente, esse arquivo também descreve as dependências entre os componentes.

Como exemplo, a Figura 65 apresenta uma parte do arquivo de configuração de componentes que permite à Implementação de Referência carregar objetos baseados na implementação baseada em SDL.

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <middleware platform="pc" system="linux" version="2.6.27.8">
3: ...
4:   <component package="gingacc-system"
          name="libgingaccsystemiosdl.so" version="0.12.1"
          type="dynamic">
5:     <dependency componentName="libSDL.so" version="1.2.0"/>
6:     <dependency componentName="libtelemidiautil.so"
                   version="0.12.1"/>
7:     <dependency componentName="libgingaccsystemio.so"
                   version="0.12.1"/>
8:     <dependency componentName="libgingaccsystemiocodemap.so"
                   version="0.12.1"/>
9:     <dependency componentName="libgingaccsystemthread.so"
                   version="0.12.1"/>
10:    <symbol object="DeviceScreen" creator="createSDLScreen"
               destroyer="destroySDLScreen"
               interface="IDeviceScreen"/>
11:    <symbol object="EventBuffer"
               creator="createSDLEventBuffer"
               destroyer="destroySDLEventBuffer"
               interface="IEventBuffer"/>
12:    <symbol object="InputEvent" creator="createSDLInputEvent"
               destroyer="destroySDLInputEvent"
               interface="IInputEvent"/>
13:    <symbol object="UserEvent" creator="createSDLUserEvent"
               destroyer="destroySDLUserEvent"
               interface="IInputEvent"/>
14:    <symbol object="Surface" creator="createSDLSurface"
               destroyer="destroySDLSurface"
               interface="ISurface"/>
15:    <symbol object="Window" creator="createSDLWindow"
               destroyer="destroySDLWindow" interface="IWindow"/>
16:    <location type="local" uri="/usr/local/lib/ginga/"/>
17:    <repository
        uri="http://www.gingancl.org.br/releases/referenceimp/"/>
18:  </component>
19:
20:...
21:
22:</middleware>
```

Figura 65 Exemplo da definição dos novos componentes baseados em SDL no modelo de componentes da Implementação de Referência.