6
Integração com o NCL Composer

Este capítulo tem como objetivo discutir a integração da ISB Designer com o NCL Composer. Essa integração acabou trazendo contribuições que não estavam previstas inicialmente, que também serão discutidas e detalhadas no decorrer do capítulo.

6.1
Motivação

Como mencionado no Capítulo 2, a nova versão do NCL Composer permite atender a diversos tipos de perfis de usuário nos mais diferentes ambientes de produção. A sua arquitetura extensível e personalizável possibilita que sejam instalados os mais diferentes conjuntos de *plugins*. Assim, o usuário pode personalizar o NCL Composer para trabalhar tanto em um ambiente de desenvolvimento, quanto em um ambiente de transmissão, bastando para isso apenas instalar os *plugins* que atendam às suas necessidades.

Para o desenvolvedor de *plugins*, o NCL Composer traz algumas vantagens em termos de projeto e implementação. De um modo geral, os *plugins* do NCL Composer possuem vários requisitos em comum. Alguns desses requisitos como o *parser* de documentos, a integração com o ambiente de execução, a organização de perspectivas, o gerenciamento de projeto etc. já são tratados pelo próprio NCL Composer. Isso permite ao desenvolvedor focar cada vez mais nas funcionalidades relevantes de sua ferramenta, deixando os detalhes comuns sobre responsabilidade do NCL Composer.

Além disso, o projeto de um novo *plugin* pode levar em consideração também as funcionalidades encontradas em outros *plugins* já instalados no NCL Composer. Dessa forma, um determinado *plugin* pode tirar proveito das funcionalidades de outros, como um meio de estender as suas funcionalidades ou resolver seus problemas. Especialmente para aqueles *plugins* interessados na autoria do documento hipermídia, esta é uma característica interessante. Na maioria dos casos, dependendo da abstração utilizada, determinadas entidades da

linguagem de especificação do documento são melhores representadas que outras. Assim, pode ser muito mais interessante para um *plugin* manipular somente algumas entidades. Isso representa para o desenvolvedor um esforço ainda menor para criação de um novo *plugin*, uma vez que a responsabilidade de especificação do documento pode ser dividida.

Para este trabalho em particular, a integração com o NCL Composer procurou resolver alguns problemas da ISB Designer. Entre esses problemas estão: a necessidade de um ambiente de pré-visualização, a preparação da aplicação produzida para transmissão e a pouca estruturação do código NCL gerado.

No caso da pré-visualização, apesar da ISB Designer possuir uma abstração bem próxima da aplicação final, o teste no ambiente de execução ainda se faz necessário, seja para fazer uma avaliação final ou para fazer uma apresentação para equipe de produção. Como visto, a integração com o ambiente de execução já é implementada pelo próprio NCL Composer. Assim, a simples integração da ferramenta com o NCL Composer já resolveria essa limitação da ISB Designer.

O mesmo acontece com a transmissão do conteúdo produzido pela ISB Designer. Dependendo do ambiente de destino, diferentes formas de preparação da aplicação são necessárias para transmitir esse conteúdo. Os diferentes sistemas de transmissão podem requerer diferentes sintaxes transferência utilizadas pelos sistemas de transmissão, que variam de acordo com o sistema de TV digital (e.g., no ambiente de TV digital terrestre, aplicações são geralmente transmitidas como *pushed data* em MPEG-2 TS (Zhou et al., 2011), no ambiente IPTV dependem de fluxos RTP (Simonson et al., 2011) e FLUTE (Chiao & Li, 2011), geralmente transmitidos sob demanda; etc.). Para lidar com transmissão do conteúdo, o NCL Composer utiliza os seus *plugins* de transmissão, que podem ser aproveitados também pela ISB Designer. Assim, para resolver a sua limitação quanto à transmissão do conteúdo, o *plugin* da ISB Designer deve ser distribuído junto com os *plugins* de transmissão do NCL Composer.

Como mencionado no Capítulo 5, o código NCL gerado pela ISB Designer aproxima-se do modelo interno da ferramenta, resultando em um código NCL com pouca estruturação, apesar de perfeitamente funcional. A estruturação lógica do documento potencializa o reúso e facilita o entendimento do código. Essa estruturação pode ser alcançada através de uma visão que permite ao usuário

visualizar e interagir com a estrutura lógica do documento, resolvendo o problema de organização do código NCL gerado pela ISB Designer.

Até o momento da integração da ISB Designer, a nova versão do NCL Composer ainda não possuía um *plugin* da visão estrutural da linguagem NCL. O desenvolvimento desse *plugin* acabou fazendo parte deste trabalho, beneficiando tanto a ISB Designer quanto o próprio NCL Composer. Os detalhes sobre a implementação da visão estrutural são discutidos na Seção 6.3.

6.2 Integração

A integração de um *software* com o NCL Composer ocorre por meio dos pontos de extensão definidos em sua arquitetura. No caso dos *plugins*, o NCL Composer define duas interfaces, *IPluginFactory* e a *IPlugin*, para que estes pontos de extensãos possam ser utilizados.

A responsabilidade de um *IPluginFactory* é criar novas instâncias de *IPlugin*. Uma nova instância do *IPlugin* é criada sempre que um novo projeto é iniciado no NCL Composer. A interface *IPluginFactory* também especifica métodos para recuperação de informações sobre versão, nome, licença e descrição do *plugin*.

Um *IPlugin* é o *plugin* do NCL Composer de fato. A instância de um *IPlugin* está sempre associada a uma instância do documento. Assim, toda e qualquer alteração ocorrida no documento será percebida por todas as instâncias de *IPlugin* associadas àquele documento. De forma semelhante, as instâncias do *IPlugin* também podem solicitar mudanças no documento. Todo esse processo de comunicação e sincronização é gerenciado pelo núcleo do NCL Composer.

Então, o primeiro passo para integrar a ISB Designer no NCL Composer foi implementar as duas interfaces definidas pelo ponto de extensão. Em seguida, para permitir que um *plugin* baseado na ISB Designer tenha controle sobre as estruturas internas da ferramenta, foi preciso implementar também as interfaces definidas pela API do controlador da ISB Designer. Assim, esse *plugin* passou a ter controle sobre as estruturas internas da ferramenta.

Para facilitar, a partir de agora chamaremos esse *plugin* baseado na ISB Designer de visão de *storyboard*. A Figura 6.1 a apresenta arquitetura da visão de *storyboard* para o NCL Composer.

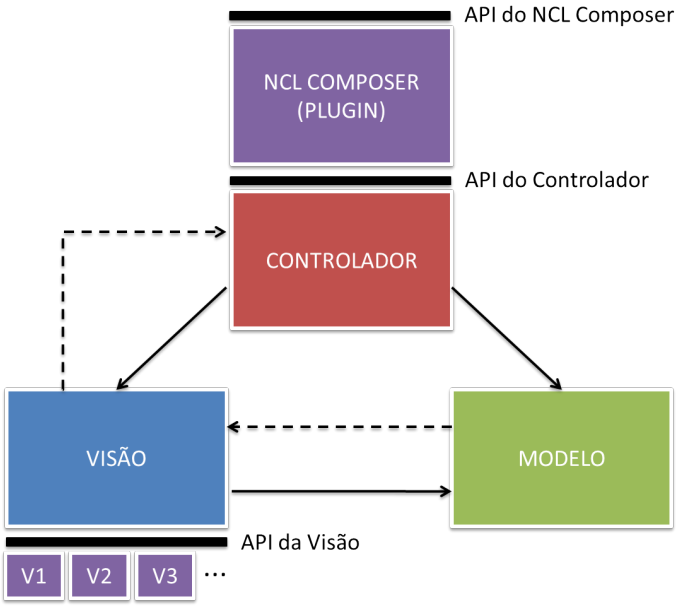

Figura 6.1 – Arquitetura da visão de *storyboard* do NCL Composer

Na implementação atual da visão de *storyboard*, a tradução entre os modelos ocorre em apenas um sentido: ISB Designer para NCL Composer. Isso porque as entidades do modelo interno do NCL Composer, que é baseado na linguagem NCL, não representam adequadamente as entidades do modelo interno da ISB Designer. A Figura 6.2 apresenta a visão de *storyboard*.

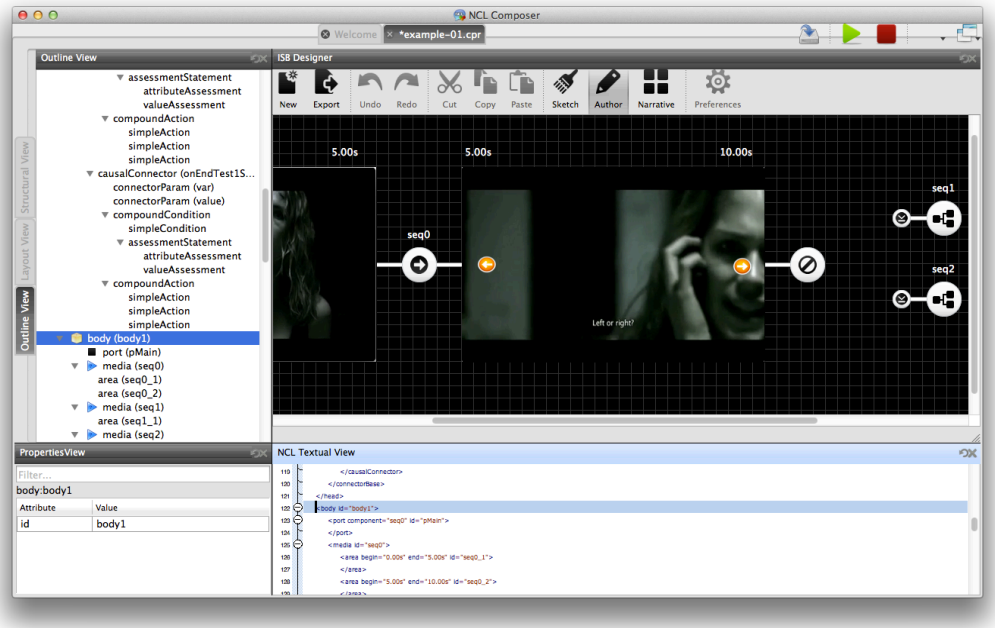

Figura 6.2 – Visão de *storyboard* do NCL Composer

6.3
Outros *plugins*

Como mencionado, a nova versão do NCL Composer ainda não possuía uma visão para representar a estrutura de um documento NCL. Essa visão acabou sendo desenvolvida durante este trabalho.

A visão estrutural permite ao autor da aplicação visualizar e interagir com a estrutura lógica do documento. Em NCL, a estrutural lógica é representada pelos nós e por relacionamentos entre eles. Os nós são representados por vértices de um grafo, que podem ser compostos, no caso dos nós de composição, ou atômicos, no caso dos nós de conteúdo. Já os relacionamentos, são representados por arestas entre vértices do grafo.

A nova versão da visão estrutural considera alguns novos detalhes em relação às versões anteriores. A representação das portas é o primeiro deles. Em NCL, as portas são pontos de interface de uma composição. As portas permitem que sejam especificados, mesmo que de forma indireta, relacionamentos entre objetos externos e internos a uma composição. Ao mapear um dos objetos internos, as portas ainda carregam outra semântica. Elas determinam também qual objeto será disparado quando a composição em que a porta é definida for iniciada. Esse último detalhe em especial foi tratado também pela versão anterior da visão estrutural, a do antigo NCL Composer. No entanto, essa informação ficava no nó de mídia.

Nessa nova versão da visão estrutural, as portas são representadas como entidades que sempre estão posicionadas nas bordas das composições. Na verdade, essa é uma característica de todos os tipos de pontos de interface representados nesta visão. Os pontos de interface da visão estão diretamente relacionadas com o elementos <*port*>, <*area*>, <*property*> e <*switchPort*> da linguagem NCL.

Com a presença das interfaces, é possível estabelecer e visualizar relacionamentos de forma bem mais precisa. Basta lembrar, por exemplo, que em NCL um relacionamento pode ser definido entre objetos ou entre trechos dos objetos. Com a presença das interfaces na visão estrutural, essa diferença fica bem clara. Na versão anterior, essa informação só era acessível por meio de formulários, escondendo esse detalhe ao representar a estrutura do documento.

Outra novidade é a representação explícita dos tipos dos relacionamentos. A visão utiliza a abordagem para representação dos relacionamentos definida em (Guimarães, Neto & Soares, 2008). Em (Guimarães, Neto & Soares, 2008) é especificada uma representação gráfica para cada tipo de condição e ação de um relacionamento. Por fim, sempre que o novo relacionamento é criado, caso não exista um conector associado a ele, esse conector é criado automaticamente. Em NCL, os conectores são responsáveis por definir os papéis e a relação de uma relação.

Para implementar a visão estrutural, procurou-se utilizar alguma biblioteca de grafo. No entanto, nenhuma das pesquisadas, como por exemplo (Graphviz, 2011), (JGraph, 2011), (Goblin, 2011) e (Qanava, 2011), dá suporte adequado para o desenho da estrutura da linguagem NCL. Normalmente, nessas APIs, a noção de aninhamento é inserida sem a semântica do encapsulamento presente na linguagem. Em outras palavras, as APIs não dão suporte ao desenho de entidade com características semelhantes às das interfaces. Dessa forma, foi preciso implementar uma nova biblioteca para desenho de grafo aninhados. Esta biblioteca, denominada libqncg[7] (*Nested Context Graph Library for Qt*), foi desenvolvida e disponibilizada como *software livre*.

A versão atual da biblioteca ainda não possui algumas funcionalidades interessantes para essa visão. Nessa versão, por exemplo, a biblioteca não permite colapsar e expandir uma determinada composição. Isso interfere diretamente na presença de mecanismos de filtragem automática como o baseado na técnica de "olho-de-peixe" (Muchaluat, Rodrigues & Soares, 1998).

Como ponto mais crítico, a biblioteca ainda não implementa algoritmos para o desenho automático do grafo composto. A solução adotada atualmente simplesmente evita que os nós sejam sobrepostos, mas não considera o cruzamento entre as arestas. Outra solução que poderia ser adotada é aplicar os algoritmos já conhecidos para desenho de grafo recursivamente dentro de cada composição. Apesar de ser uma solução aceitável, não é ótima. Dada a possibilidade de se especificar relacionamentos tanto de dentro de uma composição quanto de fora com uma interface, esses algoritmos quando aplicados recursivamente vão acabar considerando uma solução que favoreça somente um

[7] Disponível em http://www.telemidia.puc-rio.br/~edcaraujo/libqncg/

dos casos, em outras palavras, ou as interfaces serão posicionadas de maneira ótima com relação aos relacionamentos estabelecidos internamente ou as interfaces serão posicionadas de maneira ótima com relação aos relacionamentos estabelecidos com entidades externas. Assim é preciso modificar alguns desses algoritmos para considerar o comportamento particular das interfaces.

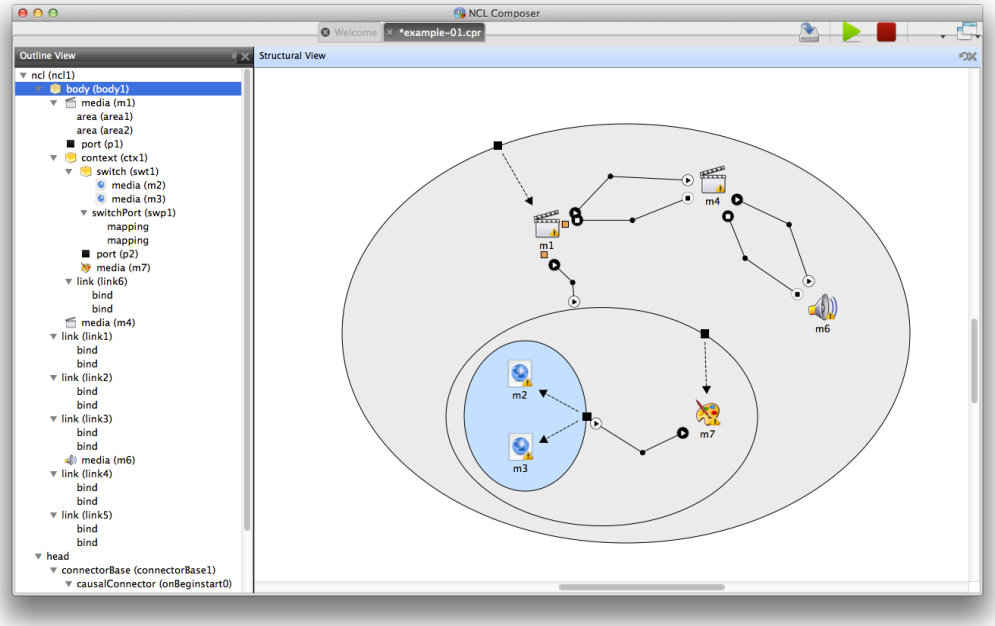

Figura 6.3 – Visão estrutural do NCL Composer

A Figura 6.3 apresenta a visão estrutural desenvolvida neste trabalho.

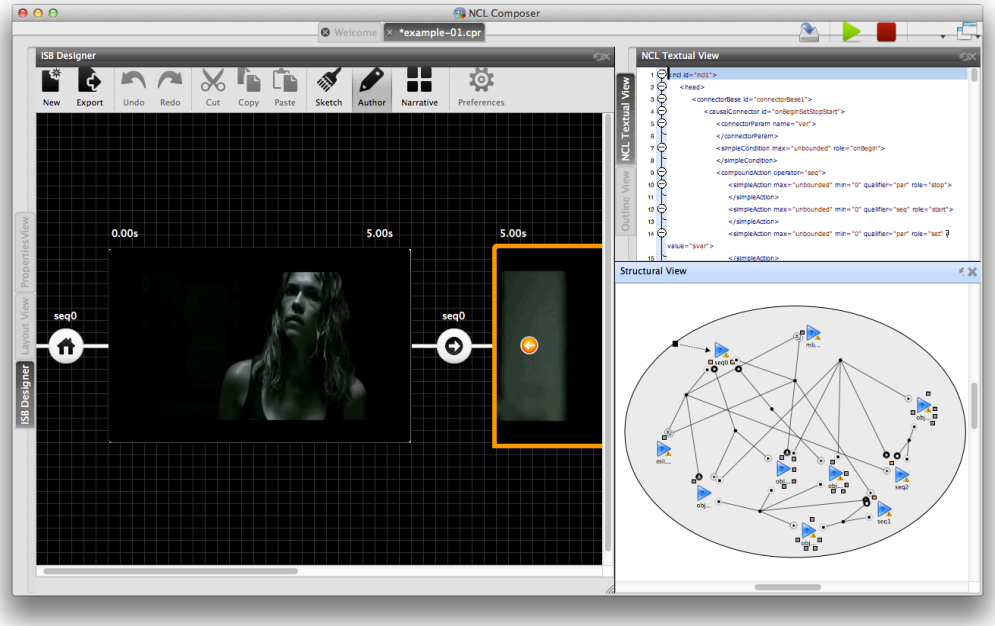

Figura 6.4 – Estrutura de um documento gerado a partir da visão de *storyboard*

A Figura 6.4 apresenta a visão estrutural do código gerado pela visão de *storyboard* do NCL Composer. Através da visão estrutural é possível notar que não existe nenhuma composição, consequência da pouca estruturação do código gerado pela visão de *storyboard*.