UNIVERSIDADE FEDERAL FLUMINENSE

Júlia Varanda da Silva

NEXT – Editor Gráfico para Programas NCL com Suporte a Templates

NITERÓI

2012

UNIVERSIDADE FEDERAL FLUMINENSE

Júlia Varanda da Silva

NEXT – Editor Gráfico para Programas NCL com Suporte a Templates

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Redes e Sistemas Distribuídos e Paralelos.

Orientadora: Profa. Dra. Débora Christina Muchaluat Saade

Niterói

2012

NEXT – Editor Gráfico para Programas NCL com Suporte a Templates

Júlia Varanda da Silva

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Redes e Sistemas Distribuídos e Paralelos.

Aprovada em Agosto de 2012.

BANCA EXAMINADORA

_____

Profa. Dra. Débora Christina Muchaluat Saade – Orientadora

UFF

_____

Prof. Dr. Esteban Walter Gonzalez Clua

UFF

_____

Profa. Dra. Cláudia Maria Lima Werner

UFRJ

Niterói

2012

Dedico esta dissertação aos meus pais, Gilda Varanda da Silva e Dinei Dias da Silva, por sempre estarem ao meu lado, me apoiando durante esta jornada;
à minha irmã, Silvia Varanda da Silva, pelas palavras de incentivo nos momentos mais difíceis;
e ao meu noivo, Igor Oldrini de Souza, pelo companheirismo e compreensão.

AGRADECIMENTOS

Agradeço, primeiramente, à minha família, meus pais e irmã, por me apoiarem e ajudarem durante a realização deste trabalho. Ao meu noivo, por sempre estar ao meu lado, sempre disposto a ajudar no que fosse preciso.

À Professora Dra. Débora Christina Muchaluat Saade, pela excelente orientação e dedicação.

Aos amigos da faculdade e ao pessoal do Laboratório MídiaCom, permitindo uma boa convivência, através de momentos descontraídos e de trocas de ideias.

A todos aqueles que, de alguma forma, contribuíram para a concretização deste trabalho. Ao grupo de TV Digital do MídiaCom, que auxiliou realizando testes e sugerindo melhoras na ferramenta e, principalmente, ao Joel André Ferreira dos Santos e ao Douglas Paulo de Mattos, que sempre me auxiliaram e contribuíram para obtenção de um melhor resultado.

É impossível haver progresso sem mudança e, quem não consegue mudar a si mesmo, não muda coisa alguma.
(George Bernard Shaw)

RESUMO

Aplicações para TV digital podem ser representadas como documentos multimídia, os quais disponibilizam vários tipos de conteúdo, além do vídeo e áudio tradicionais em um sistema de televisão. A fim de facilitar o desenvolvimento dessas aplicações são oferecidos sistemas de autoria multimídia. Estes sistemas podem oferecer diferentes ambientes para o desenvolvimento de aplicações. Para auxiliar autores que não possuem conhecimento da linguagem de autoria, é importante que estes sistemas ofereçam o uso de templates, uma vez que os templates possibilitam a criação de aplicações multimídia sem que o autor precise conhecer a especificação detalhada do documento final.

No Sistema Brasileiro de TV Digital, a linguagem NCL (Nested Context Language) é usada para especificação de aplicações seguindo o paradigma declarativo. A fim de possibilitar que autores sem conhecimento da linguagem NCL possam contribuir na geração de aplicações interativas para a TV digital, um sistema de autoria que permita o uso de templates é desejável. Este trabalho propõe um editor gráfico para autoria de documentos NCL com suporte a templates, chamado NEXT – NCL *Editor supporting XTemplate*. O NEXT oferece suporte a templates de composição especificados com a linguagem XTemplate 3.0 para autoria de documentos na linguagem NCL 3.0.

O editor é baseado no uso de plugins, os quais podem oferecer diversas funcionalidades ao autor. Além do editor, este trabalho também apresenta quatro plugins que foram desenvolvidos para uso no NEXT: editor de conectores, visão de leiaute, visão estrutural e plugin para uso de templates de composição. Através destes, é possível a construção de um documento NCL por diferentes perfis de autores.

Ao final do desenvolvimento do documento, o editor também é capaz de realizar sua validação e verificação através do uso da API aNaa, garantindo a consistência do documento de acordo com a linguagem NCL 3.0.

Palavras-chave: editor gráfico multimídia, autoria multimídia, NEXT, NCL, XTemplate, templates de composição.

ABSTRACT

Digital TV applications can be represented as multimedia documents, which provide different kinds of content in addition to traditional audio and video in a television system. In order to facilitate the development of these applications, authoring systems are offered. They can provide different environments for application creation. In order to facilitate multimedia application development by authors with no knowledge about the declarative authoring language used, the authoring system can provide the use of templates. This way authors can create applications without knowing specific details about the final document specification.

NCL (Nested Context Language) is used for creating declarative applications in the Brazilian Digital TV System. In order to enable authors with no knowledge about NCL to contribute in developing interactive digital TV applications, an authoring system that provides the use of templates is desirable. This work proposes a graphical editor for authoring NCL documents using templates, named NEXT - NCL Editor Supporting XTemplate. NEXT supports composite templates based on the XTemplate 3.0 language for NCL 3.0 document authoring.

The editor is based on plugins, which allow many features. Besides the editor, this work presents four plugins that were developed for use in NEXT: connector editor, layout view, structural view and plugin for composite template use. They can be used for developing an NCL document by authors with different profiles.

At the end of the document development, the editor provides document validation and verification using the aNaa API, ensuring document consistency according to the NCL 3.0 language.

Keywords: multimedia graphical editor, multimedia authoring, NEXT, NCL, XTemplate, composite templates.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

1.1 MOTIVAÇÃO

Aplicações para TV digital podem ser representadas como documentos multimídia, contendo diferentes tipos de mídia, como texto, imagem, áudio e vídeo. Além disso, essas aplicações podem permitir interatividade, oferecendo uma programação mais rica para o telespectador uma vez que disponibilizam vários tipos de conteúdo, além do vídeo e áudio tradicionais em um sistema de televisão.

A elaboração de aplicações multimídia interativas normalmente requer sólidos conhecimentos de programação, o que implica grandes gastos e considerável esforço para sua criação. Além disso, o cenário atual de TV digital requer metodologias fáceis e rápidas para agilizar a criação dessas aplicações, de forma que autores de diferentes perfis, inclusive aqueles que não possuem conhecimento sobre a linguagem de autoria, possam contribuir para seu desenvolvimento.

A metodologia escolhida para a criação de um aplicativo deve tratar cada uma de suas características em separado e permitir, de uma forma simples e eficaz, posteriores modificações, caso sejam necessárias.

As aplicações para TV digital podem ser desenvolvidas utilizando-se dois paradigmas de programação distintos: declarativo e imperativo. Uma linguagem declarativa possui um nível de abstração mais alto do que a imperativa, pois o autor não precisa se preocupar com detalhes sobre como a aplicação será executada no dispositivo do telespectador. Isto facilita o desenvolvimento de uma aplicação mesmo oferecendo uma menor expressividade, quando comparado ao paradigma imperativo, que é de propósito geral.

Apesar do uso de uma linguagem declarativa visar facilitar a criação de aplicações, autores sem conhecimento sobre a linguagem ainda encontram grandes dificuldades em sua utilização. Então, com o objetivo de minimizar ainda mais as dificuldades relacionadas à criação de aplicativos, são oferecidos os sistemas de autoria multimídia. Tais sistemas visam facilitar o trabalho de autoria oferecendo um ambiente gráfico para criação e edição de documentos multimídia por autores com pouco ou nenhum conhecimento na linguagem de autoria declarativa utilizada.

Uma forma de facilitar ainda mais o desenvolvimento desses documentos é utilizar editores gráficos que oferecem templates. Um template nada mais é do que um modelo pré-definido de documento sem conteúdo, ou seja, ele possui apenas as partes essenciais de um

documento, como por exemplo o cabeçalho, e as definições de como as mídias devem se comportar durante sua execução. Desta forma, a única tarefa que o autor precisará realizar é indicar quais mídias serão utilizadas no documento que utiliza o template. A Figura 1 apresenta o conceito de um editor que oferece templates para a criação de documentos multimídia.

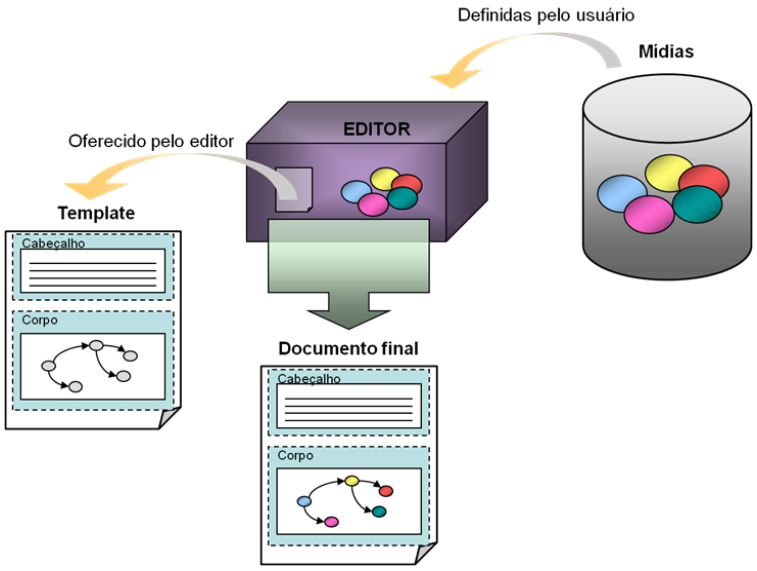


Figura 1: Editor para criação de documentos multimídia com a utilização de templates.

De acordo com a Figura 1, o editor fornece um conjunto de templates onde o autor escolhe aquele que melhor se adapta à sua necessidade, por exemplo, um template que exibe uma sequência de mídias no tempo. Após escolher o template, o próximo passo é definir quais serão as mídias e a ordem em que deverão ser exibidas. Essas mídias deverão ser informadas pelo autor. Depois de saber quais mídias serão utilizadas, o editor se encarrega de criar a aplicação final, inserindo nos locais apropriados do corpo do documento as mídias que foram escolhidas pelo autor.

Como o template é responsável por especificar a estrutura de um documento multimídia, ou seja, definir seus nós e elos, onde nós representam informações sobre as mídias e elos os relacionamentos entre as mesmas, é interessante que ele reutilize estruturas já especificadas e defina relações genéricas no documento, tornando sua autoria mais simples. Uma linguagem que permite a definição de templates é a linguagem XTemplate 3.0 [Dos Santos e Muchaluat-Saade 2011].

XTemplate permite a definição de templates de composição hipermídia, onde templates especificam componentes genéricos e seus relacionamentos, independentemente de quem são esses componentes, cuja identificação é responsabilidade do autor do documento que usará o template. XTemplate pode ser usada em conjunto com a linguagem NCL (*Nested*

Context Language) [ABNT 2007], adotada no Sistema Brasileiro de TV Digital (SBTVD) e pela recomendação ITU H.761 [International Telecommunication Union 2009] para serviços IPTV como padrão para criação de aplicações interativas utilizando o paradigma declarativo.

Quando se considera a edição de documentos NCL com templates de composição, espera-se a participação de dois perfis de autores no processo de autoria. Autores com experiência em NCL e XTemplate farão a autoria de bibliotecas de templates de composição, que poderão ser utilizadas por autores com pouca ou nenhuma experiência em NCL e XTemplate, de maneira declarativa ou através de editores gráficos que ofereçam uso de templates.

1.2 OBJETIVOS

O propósito desta dissertação é o desenvolvimento de um editor gráfico para documentos multimídia que ofereça a possibilidade de uso de templates em sua criação. Assim sendo, este trabalho apresenta o editor NEXT - NCL *Editor supporting XTemplate*, uma ferramenta de autoria gráfica que foi produzida com o intuito de facilitar a criação de aplicações para TV digital com a linguagem NCL 3.0 e oferecer suporte ao uso de templates de composição, criados com a linguagem XTemplate 3.0.

A principal motivação para o desenvolvimento do editor foi a possibilidade de permitir que autores sem conhecimento da linguagem NCL possam ser capazes de construir aplicações graficamente através do uso de templates. Além disso, a ferramenta também objetiva atender a outros autores com diferentes habilidades, possibilitando o uso de diferentes visões para edição do documento multimídia durante seu desenvolvimento.

Dando suporte a futuras extensões, NEXT também foi desenvolvido de forma a facilitar a inclusão de novas funcionalidades sem que seja necessário modificar seu código, permitindo que o desenvolvedor de novas funcionalidades preocupe-se apenas com a extensão a ser adicionada ao editor. A arquitetura do NEXT é baseada em um núcleo capaz de se comunicar com um conjunto de plugins, os quais podem ser configurados e estendidos, tornando o editor adaptável a autores de diferentes perfis.

A fim de validar a comunicação do editor com seus plugins, foram desenvolvidos quatro plugins gráficos para facilitar/agilizar a autoria de uma aplicação NCL. O uso simultâneo destes plugins também foi testado para verificar a notificação sobre alterações no documento aos diversos plugins utilizados pelo autor, permitindo, assim, que os mesmos mantenham-se sempre consistentes com o documento em edição.

Sendo assim, pode-se ressaltar como principal contribuição desta dissertação, um editor gráfico multimídia que:

- permite a edição de qualquer documento NCL 3.0, independentemente de onde este tenha sido criado;

- oferece o uso de templates, permitindo sua utilização por autores sem conhecimento da linguagem NCL;

- cria uma representação gráfica dos templates para auxiliar na compreensão do mesmo por parte do autor;

- permite a inserção de novas funcionalidades através da instalação de plugins, possibilitando seu uso por autores com diferentes habilidades;

- está disponível em dois idiomas (inglês e português) e possibilita a inclusão de outros idiomas de forma facilitada;

- foi desenvolvido em Java para permitir seu uso em diferentes plataformas;

- oferece análise estrutural e comportamental do documento, ajudando o autor a criar aplicações consistentes.

1.3 ESTRUTURA DA DISSERTAÇÃO

O trabalho está estruturado da seguinte forma. No Capítulo 2, são discutidos os trabalhos relacionados ao trabalho proposto. São apresentadas diversas ferramentas de autoria, destacando-se as principais funcionalidades de cada uma. Além disso, são abordados os requisitos básicos que um editor gráfico multimídia deve contemplar a fim de facilitar o desenvolvimento de aplicações multimídia, os quais foram definidos com base nas funcionalidades dos trabalhos estudados e nos objetivos principais do presente trabalho.

O Capítulo 3 apresenta a linguagem NCL versão 3.0 e sua estrutura, dando ênfase aos elementos que são abordados pelos plugins que foram desenvolvidos para utilização com o NEXT. O capítulo também apresenta brevemente a linguagem XTemplate 3.0 e como NCL foi estendida para uso de templates.

O Capítulo 4 o editor NEXT, sua arquitetura, a forma utilizada para comunicação com seus plugins e como deve ser realizado o desenvolvimento dos mesmos.

No Capítulo 5, são apresentados diversos plugins que foram desenvolvidos para o NEXT. Estes plugins permitem a criação de um documento multimídia por autores com conhecimento da linguagem NCL. O capítulo apresenta também os resultados dos testes de usabilidade sobre os plugins desenvolvidos.

Já o Capítulo 6, apresenta o plugin que foi desenvolvido especialmente para permitir o desenvolvimento de documentos NCL através da utilização de templates de composição descritos com a linguagem XTemplate 3.0. Este plugin objetiva atender aos autores sem conhecimento algum sobre a linguagem NCL. O capítulo apresenta também os resultados do teste de usabilidade sobre o plugin para uso de templates.

Finalmente, no Capítulo 7, apresenta-se a comparação entre os trabalhos estudados e o editor proposto, as principais contribuições da dissertação e trabalhos futuros.

CAPÍTULO 2 – TRABALHOS RELACIONADOS

Com o objetivo de obter informações sobre características principais de editores gráficos e embasar a escolha de requisitos importantes a serem contemplados por um editor, foi realizado um estudo sobre alguns trabalhos que têm como finalidade facilitar a autoria de documentos multimídia, seja propondo novas ferramentas ou propondo estruturas de templates. Uma breve descrição de cada um dos trabalhos estudados é apresentada, assim como suas principais contribuições.

São comentados os seguintes trabalhos relacionados: A-SCORM [López et al. 2008], DEMAIS [Bailey et al. 2001], Composer 3 [Lima et al. 2010], LAMP [Celentano e Gaggi 2003], EDITEC [Damasceno et al. 2010], LIMSEE 3 [Deltour e Roisin 2006], *Template-Based MHP Authoring Tool* [Chiao et al. 2006], *Mobile Multimedia Presentation Editor* [Jokela et al. 2008], Berimbau [Berimbau iTV Author 2011] e ISB Designer [Araújo 2012]. Ao final do capítulo, os requisitos que foram considerados mais importantes em uma ferramenta de autoria gráfica multimídia e uma comparação entre os trabalhos estudados são apresentados.

2.1 A-SCORM COURSE CREATOR TOOL

T-MAESTRO [López et al. 2008] é um sistema que visa selecionar conteúdos personalizados, os quais relacionam entretenimento com aprendizado, para serem exibidos ao usuário através de um sistema de televisão digital interativa. Entretanto, para que esse sistema atinja seu objetivo, é necessária a existência de conteúdo interativo.

O editor RELOAD foi criado para permitir que autores criassem objetos de aprendizagem no padrão SCORM [Advanced Distributed Learning 2006]. Com o objetivo de adaptar estes objetos ao sistema T-MAESTRO, a ferramenta gráfica de autoria A-SCORM *Course Creator Tool* foi desenvolvida. Esta ferramenta permite a criação de cursos adaptativos para o padrão ADL SCORM, que é o padrão escolhido para funcionamento do sistema. A-SCORM é composta pelo editor RELOAD adicionado de dois submódulos, o submódulo de edição das regras de adaptação e o de repositórios de SCO (*Sharable Content Object*).

Um SCO representa um objeto de conteúdo adaptável que se comunica com o sistema permitindo a adaptação do curso de acordo com as regras de adaptação, as quais levam em consideração as características do usuário, possibilitando, assim, a personalização do

conteúdo. O SCO é composto por um arquivo JAR (*Java ARchives*) que engloba todos os arquivos multimídia e arquivos XML necessários para o correto funcionamento do SCO.

O primeiro módulo, ou SCO *creator tool*, permite que o autor especifique características como aparência e configuração do conteúdo, assim como a adaptação dos SCOs. Com este módulo, basta o autor selecionar um template, dentre os que são fornecidos pela ferramenta, especificar as opções, que variam de acordo com as regras, e criar as regras de adaptação.

Depois que o autor seleciona o template desejado, a janela é dividida em três áreas, que podem ser visualizadas na Figura 2. À esquerda, o template escolhido é exibido, assim como os seus componentes. Para alterar a configuração de algum dos componentes, basta que o mesmo seja selecionado e a caraterística pode ser alterada utilizando-se o editor de propriedades, que é exibido à direita.

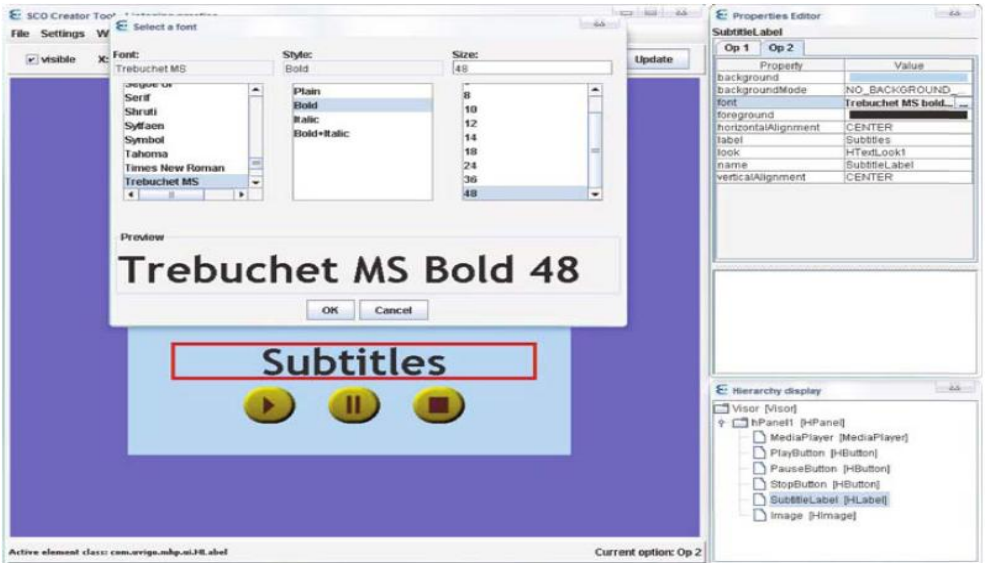


Figura 2: SCO creator tool: edição das características da letra a ser utilizada no campo selecionado, retirado de [López et al. 2008].

Características referentes à posição, tamanho, visibilidade, entre outras, podem ser editadas utilizando-se a barra de ferramentas que aparece acima do template. E, por fim, é possível observar a hierarquia entre os objetos, e também selecioná-los, no canto esquerdo inferior da tela.

O segundo módulo, ou SCO *repository*, permite a inclusão de SCOs que foram criados pelo submódulo anterior, no curso que está sendo desenvolvido.

2.2 DEMAIS

O objetivo da ferramenta de autoria DEMAIS [Bailey et al. 2001] é oferecer um ambiente para a criação de protótipos de conteúdo multimídia para ser utilizada no início de seu desenvolvimento, ou seja, quando suas características iniciais ainda estão sendo definidas. A ferramenta tem como público alvo os designers. Algumas entrevistas foram realizadas com os mesmos para que fosse possível determinar os maiores problemas encontrados por eles nas ferramentas de autoria e que características são importantes no início da criação de conteúdo multimídia.

Com o resultado das entrevistas, foi concluído que os designers usualmente preferem utilizar papel e lápis no início, pois desta forma é fácil alterar alguma característica e fazer anotações/rabiscos em qualquer lugar. No entanto, com apenas papel e lápis não é possível verificar e simular o programa, ou seja, saber se o programa irá funcionar como esperado quando o usuário interagir com o mesmo, ou até mesmo, se o comportamento pensado inicialmente corresponde às expectativas do próprio autor.

Com o intuito de oferecer uma ferramenta adequada aos designers, DEMAIS é composto por um conjunto de storyboards onde o autor pode "rabiscar" e fazer anotações utilizando um canvas, como se tivesse mesmo com lápis e papel nas mãos. É possível editar os storyboards, fazer reconhecimentos de traços que formam retângulos inseridos no mesmo pelo canvas, inserir narração e objetos multimídia, adicionar sincronizações entre os mesmos, e/ou entre outros storyboards, definir comportamentos baseados no tempo ou em eventos e editar os mesmos utilizando uma linguagem visual expressiva. A simulação do conteúdo também é realizada pela ferramenta.

Na Figura 3, é apresentado um storyboard da ferramenta com alguns objetos e anotações feitas pelo autor. A existência de elos entre os componentes também são representadas na figura.

No storyboard o autor pode cortar, copiar, colar e selecionar elementos. O reconhecimento de um retângulo é feito utilizando uma implementação modificada do *Rubine's gesture classifier* [Hong e Landay 2000].

Também é possível especificar um comportamento através de uma anotação. Para isto DEMAIS utiliza um parser baseado na gramática LL 1, a qual suporta um conjunto de sentenças como, por exemplo, "*After 5 seconds, start the video*". No entanto, é preciso que o autor conheça as regras de sintaxe da gramática.

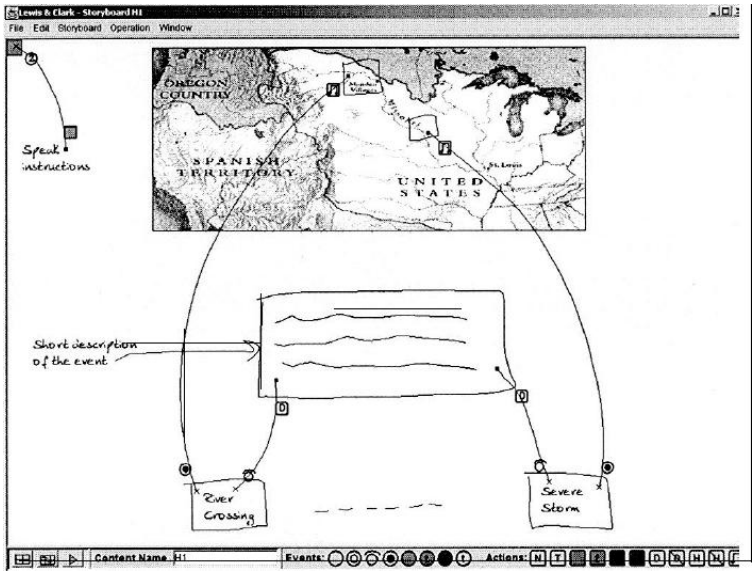


Figura 3: Storyboard do editor DEMAIS com alguns componentes e respectivos comportamentos, retirado de [Bailey et al. 2001].

Os ícones que aparecem abaixo do storyboard, na Figura 3, são utilizados para editar os eventos da fonte e do destino. Cada ícone de evento expressa uma condição que, ao ocorrer no elemento fonte, causará uma ação no elemento destino. Cada uma das ações disponíveis também são representadas por ícones. Para alterar o evento, basta clicar no evento desejado e depois clicar perto do evento a ser alterado. O mesmo procedimento deve ser realizado para alterar uma ação.

Existem também os ícones de transição que, quando adicionados a um comportamento, definem a velocidade e o tipo da transição. O autor deve escolher estes parâmetros de acordo com as opções existentes.

Com o *Narration Editor* é possível fazer gravações de voz e inserir marcadores de sincronização. Estes marcadores podem ser posteriormente utilizados para especificar ações a serem tomadas quando eles forem alcançados. Para especificar uma ação ao marcador, o autor deve arrastar a narração e soltá-la no *multiview editor*. Depois, basta marcar um traço a partir do marcador desejado até outro storyboard que também esteja no *multiview editor*.

O *multiview editor* é utilizado para definir uma ação para um marcador ou para definir comportamentos que não podem ser definidos no storyboard. Na Figura 4, é exibido o *multiview editor* com quatro storyboards e três marcadores com ações definidas.

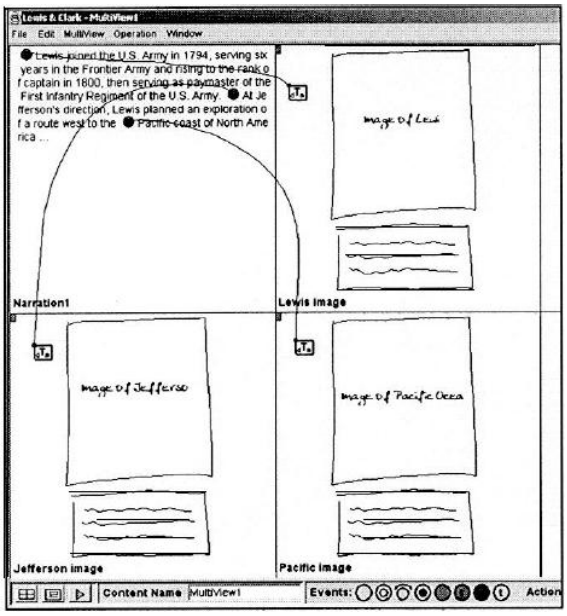


Figura 4: Multiview editor, retirado de [Bailey et al. 2001].

Finalmente, para simular o aplicativo que está sendo desenvolvido, basta utilizar a opção play, que também se encontra abaixo do storyboard. Como a ferramenta não verifica a consistência entre os comportamentos criados, pode ser que algum resultado inesperado ocorra durante a simulação do aplicativo.

2.3 COMPOSER 3

Com o intuito de oferecer um único ambiente de autoria adequado para diferentes perfis de usuários, desde usuários domésticos a produtores de conteúdo, o Composer 3 [Lima et al. 2010] propõe a base para a construção de um ambiente integrado, capaz de se adaptar a vários perfis e oferecer suporte a requisitos não funcionais.

Como exemplos de requisitos não-funcionais, podemos citar a extensibilidade, que introduz a possibilidade de acrescentar novas funcionalidades à ferramenta de forma fácil, o desempenho permitindo a escalabilidade, ou seja, que o desempenho da ferramenta não seja degradado com o aumento de funcionalidades, a portabilidade, que permite o funcionamento da mesma em qualquer plataforma, entre outros.

O padrão arquitetural do Composer 3 é baseado em um micronúcleo, um modelo central e extensões. O micronúcleo agrupa as funcionalidades mínimas do sistema e coordena a forma como as extensões devem colaborar. As extensões são feitas por meio de plugins, que são programas capazes de interagir com o micronúcleo e que adicionam novas funcionalidades e/ou recursos à ferramenta.

A Figura 5 exibe o funcionamento do micronúcleo, que realiza a comunicação com as outras partes do sistema através da troca de mensagens.

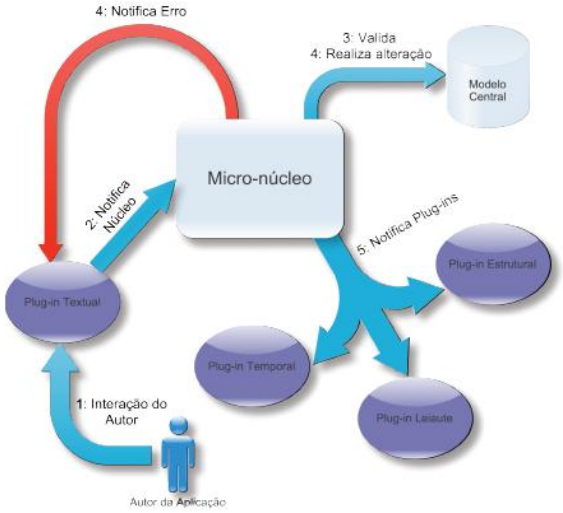


Figura 5: Comunicação entre micronúcleo e plugins, retirado de [Lima et al. 2010].

O modelo central é a representação interna da aplicação em desenvolvimento pelo usuário. Sobre ele é possível realizar operações de consulta e de modificações. As de consulta são realizadas pelos plugins e as de modificações só podem ser realizadas através do micronúcleo, desta forma, o controle de acesso concorrente ao modelo central é realizado. Cada vez que o modelo central é alterado, o micronúcleo informa as modificações aos plugins para que os mesmos possam se atualizar, visando a consistência do modelo.

A fim de evitar mensagens e processamentos desnecessários, quando ocorre alguma alteração em determinado componente do modelo central, apenas os plugins que têm interesse nesse componente são informados sobre a alteração. Para isso, toda vez que um plugin é adicionado à ferramenta, o mesmo deve informar os seus componentes de interesse ao micronúcleo.

Para desenvolver um plugin é necessário implementar duas interfaces: *IPluginFactory* e *IPluginMessage*. A primeira define como criar e destruir instâncias do plugin e a segunda trata da comunicação do mesmo com o micronúcleo e da definição do filtro de interesse, que possui os componentes de interesse do plugin criado.

Até o momento, foram desenvolvidos seis plugins para o Composer, visão estrutural, visão de leiaute, visão estrutural em forma de árvore, visão de propriedades, visão textual e o validador NCL. Mais informações sobre os mesmos podem ser encontradas em [NCL Composer 2012].

Apesar de facilitar a autoria de documentos multimídia, o Composer não possibilita o uso de templates, não realiza a simulação do documento final, assim como não analisa o comportamento de execução do mesmo.

2.4 LAMP

Com o intuito de permitir a geração automática de apresentações multimídia, o trabalho realizado por [Celentano e Gaggi 2003] modela esquemas de templates. A idéia é oferecer ao usuário a possibilidade de definir o layout e o comportamento da apresentação, as características e atributos dos objetos sem se preocupar com as instâncias que serão utilizadas para preencher o template. Esse trabalho teve como foco a coordenação e a sincronização de objetos de mídia contínuos.

A geração automática da apresentação é dividida em duas partes: na primeira o template é definido e, na segunda, os dados que serão utilizados na apresentação são retirados de um repositório de dados e inseridos no template. Após a segunda parte, a apresentação é iniciada.

Para descrever visualmente o comportamento da apresentação, foi adotada a utilização de grafos. A Figura 6 mostra um exemplo de utilização dos grafos para a representação de uma aplicação. Cada caixa representa um objeto de mídia e as relações entre os componentes são representadas pelas arestas que ligam os mesmos. Cada uma das arestas contém um símbolo, que define o tipo de evento da relação, e um semicírculo ligado a um dos componentes. O componente do semicírculo é o mestre (*master*) da relação e, o outro componente, é o escravo (*slave*).

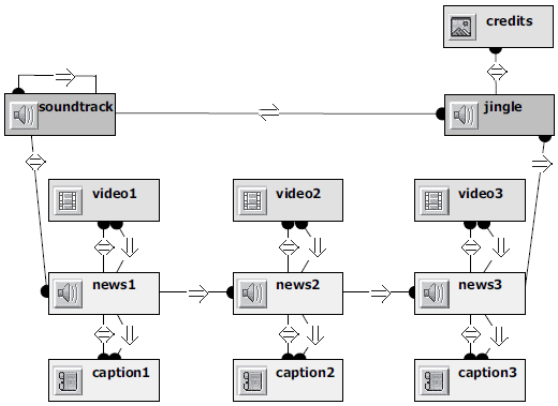


Figura 6: Grafo de sincronização de uma aplicação, retirado de [Celentano e Gaggi 2003].

O modelo de sincronização é baseado em eventos. Eventos internos, que são gerados pelos objetos de mídia, e eventos externos, gerados através da interação do usuário com a

apresentação, são considerados diferentes tipos de evento. Por exemplo, o fim natural de uma mídia de vídeo é um evento interno. Por outro lado, ocasionar o término do vídeo selecionando a opção stop, é um evento externo. Seja uma relação entre dois objetos, a e b, definida de forma que quando ‘a’ chegar ao fim, ‘b’ deve iniciar. Caso o usuário finalize ‘a’, o objeto ‘b’ não será iniciado uma vez que ‘a’ não atingiu seu fim natural.

O modelo também permite a criação de composições, que representam um grupo de eventos que ocorre diversas vezes na apresentação. Uma composição recebe o nome de estêncil (stencil). A Figura 6 exibe o mesmo grafo da aplicação da Figura 7, mas desta vez com a utilização de um stencil.

O modelo organiza os objetos de mídia em uma estrutura hierárquica, a qual facilita a apresentação de aplicações complexas. Ele também permite que um mesmo objeto de mídia possa ser referenciado diversas vezes, sem causar redundâncias, através de seu identificador. Desta forma, é possível fazer o reúso de estruturas de outras apresentações, ou até mesmo, do documento inteiro.

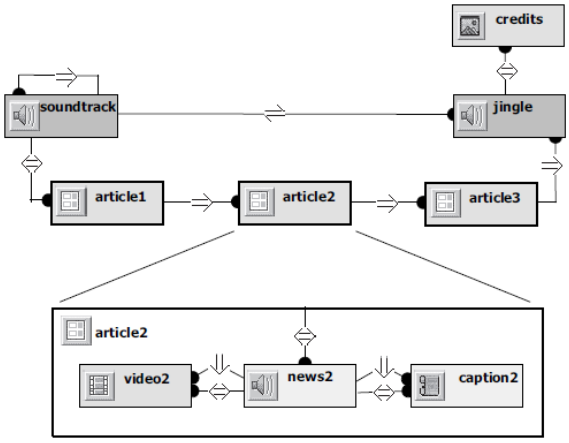


Figura 7: Grafo de sincronização de uma aplicação utilizando stencil, retirado de [Celentano e Gaggi 2003].

A estrutura e o comportamento da apresentação são descritos através de um esquema XML, baseado no mesmo modelo de sincronização representado pelo grafo. A linguagem XML é utilizada para separar as relações espaciais e temporais das referências dos objetos de mídia em três seções: uma que define o layout espacial do documento, uma que define as mídias envolvidas na apresentação e uma que define o comportamento temporal das mídias.

A apresentação do modelo descrito é possível através da utilização da ferramenta de autoria LAMP (*LAboratory for Multimedia presentation Prototyping*), a qual permite ao autor criar, testar e executar a apresentação especificando as mídias envolvidas e as relações de sincronização entre as mesmas. Esta ferramenta é composta por um editor visual, um simulador para testar os comportamentos considerando as relações entre as mídias e as

possíveis interações do usuário, um gerador que relaciona as mídias com o template, e o apresentador, que exibe a apresentação criada.

O editor permite tanto a criação de leiautes quanto a criação de apresentações, adicionando-se os objetos de mídia e as relações entre eles no grafo. O layout da apresentação é feito através da criação de retângulos na janela de apresentação. A Figura 8 mostra a interface da ferramenta. Para facilitar a compreensão das relações de evento entre as mídias, quando o usuário ativa um objeto, o simulador reforça a representação do mesmo no grafo e o seu efeito é propagado para os outros objetos relacionados ao que foi ativado.

Embora LAMP forneça vários recursos para a autoria de documentos multimídia, ele não realiza a análise do comportamento de execução do documento criado.

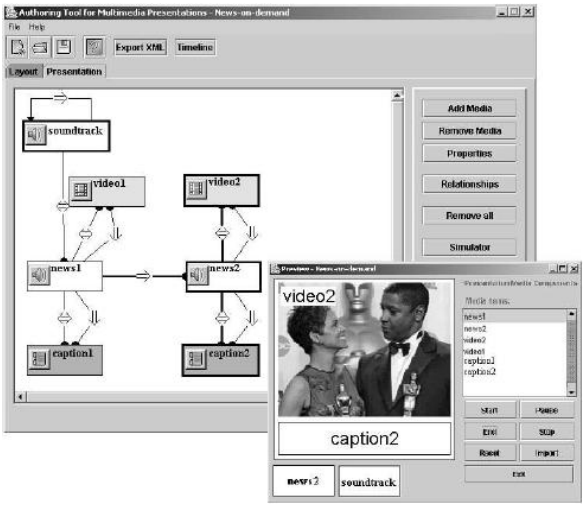


Figura 8: Interface visual da ferramenta de autoria LAMP, retirado de [Celentano e Gaggi 2003].

2.5 EDITEC

O EDITEC [Damasceno et al. 2010] é um editor gráfico de templates de composição multimídia baseado na linguagem Xtemplate 3.0. O sistema oferece três visões permitindo ao usuário ter uma melhor visualização do template durante o processo de autoria. As visões oferecidas são: estrutural, leiaute e textual. O usuário tem total liberdade para organizar o seu espaço de trabalho conforme desejar e os elementos visuais podem ser movidos e redimensionados.

A ferramenta oferece uma visão miniatura (Overview) da área de trabalho com o objetivo de facilitar a busca por algum componente que pode não estar localizado na área visível da área de trabalho. Além disso, o editor apresenta recursos de zoom, scroll e filtragem de elementos.

A visão estrutural é a principal visão da ferramenta e nela o usuário pode criar os elementos e definir seus relacionamentos, de acordo com a linguagem XTemplate, que farão parte do documento. Ela é dividida em duas abas, Vocabulary View e Body View, a primeira apresenta os elementos da linguagem e a segunda, apresenta os relacionamentos espaço-temporais entre os componentes do template. Quando um elemento é selecionado, suas propriedades são apresentadas no canto inferior esquerdo da visão estrutural. A Figura 9 exibe a visão estrutural do EDITEC.

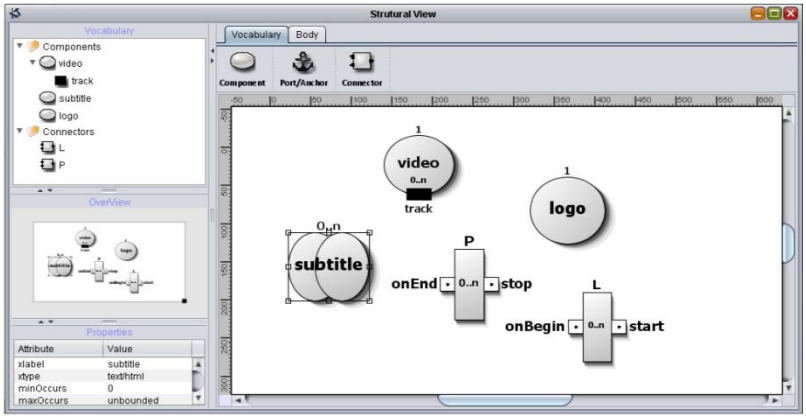


Figura 9: Visão estrutural do EDITEC, retirado de [Damasceno et al. 2010].

A visão de leiaute apresenta as regiões do documento nas quais as mídias podem ser exibidas. Esta visão permite criar, editar e apagar regiões graficamente. Todas as regiões são exibidas no canto superior esquerdo da tela em forma de árvore, desta forma é possível ter uma melhor visualização da hierarquia entre as regiões. As regiões podem ser movidas e redimensionadas, de forma que as regiões filhas também o sejam, mantendo, assim, as posições relativas. Opções de alinhamento também são oferecidas, possibilitando o alinhamento de um grupo de regiões de forma rápida. É também possível definir descritores para as regiões, os quais são listados no canto inferior esquerdo da tela. Descritores especificam, além da região, outras características de exibição de um objeto de mídia. A Figura 10 exibe a visão de leiaute.

A visão textual é dividida em duas partes, uma para exbir o código do template e, a outra, o código da base de descritores, regiões e regras do template.

Estruturas de iteração também podem ser construídas de forma gráfica utilizando-se as opções oferecidas pela ferramenta. Estas estruturas são especificadas em binds e port mappings [Damasceno et al. 2010], quando esses se ligam a componentes que representam um conjunto de mídias, cuja cardinalidade pode ser desconhecida.

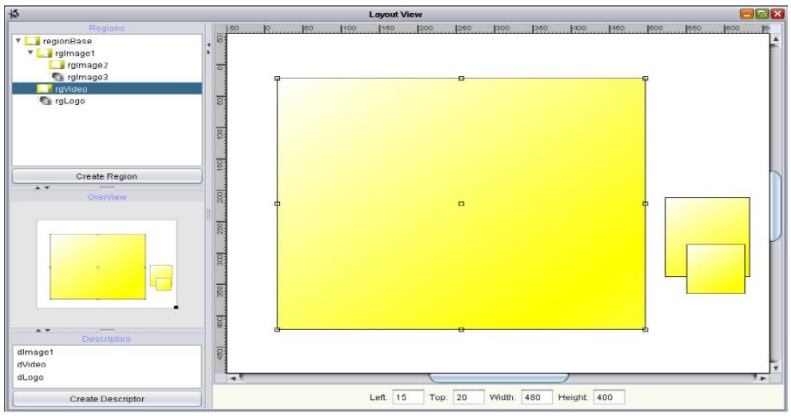


Figura 10: Visão de leiaute do EDITEC, retirado de [Damasceno et al. 2010].

A linguagem XPath [W3C 1999a] é utilizada na definição de restrições adicionais sobre os componentes do documento. Para facilitar a criação dessas expressões, foi desenvolvida uma interface com diversas opções, dando ao usuário liberdade na construção de diversos tipos de expressões básicas. No entanto, também é possível definir as expressões XPath de forma textual.

Através da interface gráfica, não é possível oferecer todas as funcionalidades lógicas permitidas pela linguagem XTemplate 3.0. Sendo assim, em alguns casos especiais, é necessário usar a edição textual, o que requer mais conhecimento da linguagem por parte do autor. EDITEC permite o desenvolvimento de templates, porém não oferece a criação de documentos multímidia utilizando templates previamente criados. As ferramentas que permitem a utilização de templates de composição para criar um documento NCL são a ferramenta Wizard para criação de documentos NCL com templates [Dos Santos e Muchaluat-Saade 2011] e o plugin de templates desenvolvido no contexto desta dissertação, que será apresentado no Capítulo 6.

2.6 LIMSEE 3

A proposta de Limsee3 [Deltour e Roisin 2006] foi fornecer um modelo de documento para a criação de ferramentas de autoria usando templates. Baseia-se na integração dos componentes de acordo com suas estruturas lógica, temporal e espacial. O modelo considera a estruturação lógica como a estrutura principal do documento, o qual é composto por uma árvore de componentes modulares.

Cada componente possui um template próprio o qual define as estruturas temporal e espacial do componente utilizando a linguagem SMIL [Synchronized Multimedia Integration Language 2010]. Desta forma, estes componentes podem ser definidos independentemente do

documento e reutilizados em outros documentos. A modularidade é garantida ao fornecer abstração aos tipos de componentes através do uso de nomes simbólicos.

São definidos dois nós de template, “media zone” e “repeatable structure”, para facilitar a autoria por parte do autor. O primeiro define a posição espacial do objeto e o segundo representa uma lista homogênea de objetos.

Com o modelo de Limsee3, esperou-se obter uma melhor modularidade, reúso e facilidades na manutenção e definição de novos templates. Na Figura 11, pode-se observar o processo de autoria de um documento com Limsee3.

Por se tratar de um modelo de documento, as características e/ou requisitos de um editor que utilize este modelo são definidas pelo autor do mesmo, já que o modelo não impõe nenhuma restrição e não define nenhum requisito relacionado ao modelo de funcionamento de um editor.

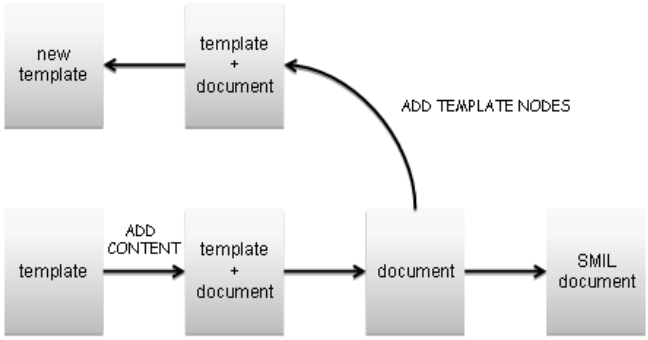


Figura 11: Processo de autoria em LimSee3, retirado de [Deltour e Roisin 2006].

2.7 TEMPLATE-BASED MHP AUTHORING TOOL

O objetivo do trabalho *Template-Based MHP Authoring Tool* [Chiao et al. 2006] foi a criação de uma ferramenta de autoria baseada na geração de código compatível com o MHP (*Multimedia Home Platform*) [Multimedia Home Plataform 2010] middleware do sistema europeu de TV digital. A ferramenta utiliza templates baseados em XML. As aplicações para o padrão MHP devem utilizar a linguagem Java.

Para possibilitar a introdução de novas funcionalidades à ferramenta e adaptações no conteúdo gerado, isto é, algumas modificações na estrutura da aplicação, a ferramenta de autoria conta com um “gerador de metaprograma” para gerar o “programa real”. Este gerador é responsável por gerar o código Java de acordo com o template XML, o qual especifica os parâmetros para a criação das instâncias de cada módulo. Sendo assim, para adicionar novas funcionalidades à ferramenta, não é necessário modificar a ferramenta inteira, basta realizar as modificações necessárias no gerador de metaprograma e no template do módulo que receberá a nova funcionalidade.

Na Figura 12, pode-se observar a estrutura da aplicação utilizando o modelo do template proposto. Um ator (*actor*) representa qualquer objeto multimídia, uma tomada (*shot*) é um conjunto de atores e um programa deve ser composto de pelo menos uma cena (*scene*), ou seja, um conjunto de tomadas. A ferramenta de autoria descreve a relação entre esses módulos utilizando o template XML.

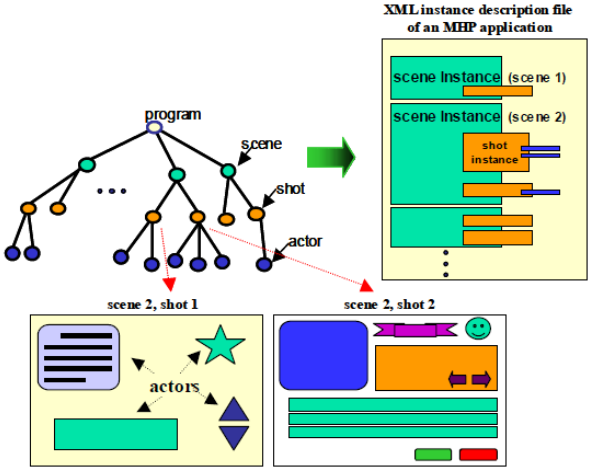


Figura 12: Estrutura da aplicação utilizando Template-Based MHP Authoring Tool, retirado de [Chiao et al. 2006].

2.8 MOBILE MULTIMEDIA PRESENTATION EDITOR

O aplicativo *Mobile Multimedia Presentation Editor* [Jokela et al. 2008] possibilita a criação de apresentações multimídia com diversos tipos de mídias em um dispositivo móvel. A apresentação pode ser armazenada no próprio dispositivo ou, ainda, ser distribuída através de qualquer mecanismo de distribuição. Alguns estudos e experimentos com usuários comuns foram feitos para se determinar as principais ferramentas que o aplicativo deveria disponibilizar aos mesmos.

Depois de realizados os estudos e de posse dos resultados dos experimentos, chegou-se a conclusão de que a ferramenta deveria ser flexível, expressiva, possibilitar a criação de apresentações personalizadas e advertir o usuário sobre possíveis falhas ou limitações. Ser flexível se refere à possibilidade do usuário realizar alterações nas diversas telas da apresentação de forma fácil e eficiente, como, por exemplo, trocá-las de ordem a qualquer momento. Ser expressiva significa que mídias de alta qualidade devem ser suportadas para garantir expressividade e elegância na apresentação que se deseja transmitir. Os usuários desejam poder personalizar suas apresentações, logo não se espera que estes utilizem apenas os templates que são oferecidos. E, por último, os usuários devem ser avisados sobre suas possibilidades e limitações de edição.

A interface do usuário é composta por diferentes visões e menus de opções que mudam de acordo com a visão que está sendo utilizada no momento. Na Figura 13, pode-se visualizar a interface do editor.

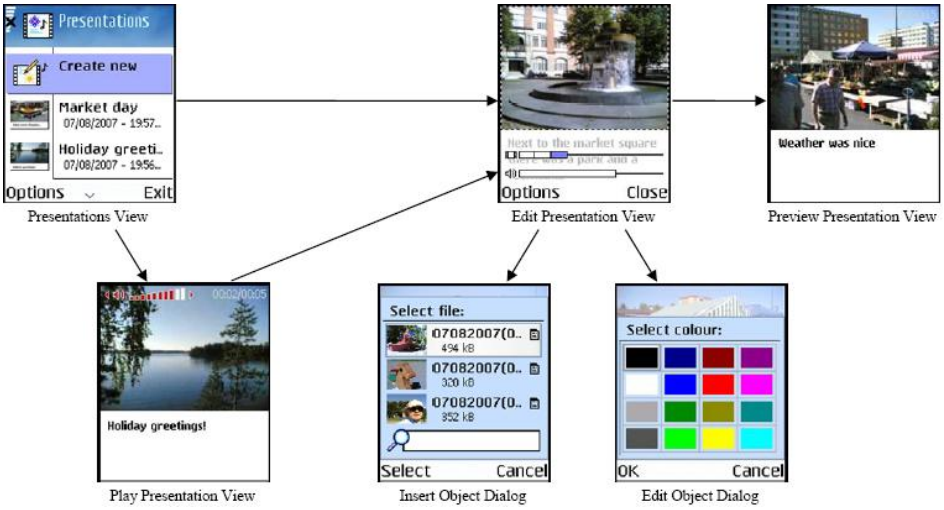


Figura 13: Interface do aplicativo Mobile Multimedia Presentation Editor, retirado de [Jokela et al. 2008].

A visão de apresentação (*Presentation View*) permite utilizar as apresentações existentes ou criar uma nova apresentação. Na visão de Play (*Play Presentation View*), como o próprio nome diz, é onde as apresentações são executadas e o usuário pode, a partir desta visão, alterar para a visão de edição (*Edit Presentation View*), que é onde as apresentações são alteradas. A visão de preview (*Preview Presentation View*) permite visualizar a apresentação enquanto a mesma ainda está sendo criada.

As apresentações criadas são representadas utilizando-se a linguagem SMIL, mas nem todas as possibilidades que a linguagem permite são contempladas pela ferramenta e mídias de vídeo não são suportadas. As relações temporais são baseadas na sequência das páginas, sendo que uma página só chega ao fim de sua exibição quando todas as suas mídias terminam de ser exibidas. Também é possível que um objeto faça parte de várias páginas. É possível, por exemplo, que um áudio seja tocado durante toda a apresentação e, consequentemente, durante as trocas de páginas. A visualização temporal é feita através de duas linhas temporais que aparecem no topo das telas, uma mostra a página corrente e a outra a duração da mídia áudio, caso exista.

As teclas de navegação para direita e esquerda são utilizadas para trocar de página e as teclas de navegação para cima e para baixo são utilizadas para que o usuário possa navegar pelas mídias da página. Ao selecionar uma das mídias, um menu é exibido com as opções de

edição para a mídia selecionada. Alguns templates são oferecidos pela ferramenta, mas o usuário tem a possibilidade de criar uma apresentação sem utilizá-los.

O aplicativo não oferece todas as funcionalidades da linguagem SMIL e não permite a utilização de todos os tipos de mídia, como vídeo por exemplo.

2.9 BERIMBAU

Berimbau iTV Author [Berimbau iTV Author 2011] é uma ferramenta de autoria gráfica para aplicações de TV digital voltada para profissionais de mídias que não possuem conhecimentos de programação. A ferramenta oferece uma interface bastante simples e intuitiva, além de um repositório de mídias para uso durante a criação da aplicação. Utiliza uma linguagem própria para representação de seu modelo de documento, o qual é utilizado para gerar a aplicação final na linguagem NCL.

Apesar de facilitar a autoria de documentos NCL, o editor apresenta algumas limitações, como por exemplo, não oferecer diferentes visões, não conseguir contemplar todas as funcionalidades da linguagem NCL, só permitir a edição de documentos inicialmente criados pela própria ferramenta e somente documentos que contenham mídias de imagens, nenhum outro tipo de mídia é suportado pelo editor. Além disso, não oferece o uso de templates para a criação do documento final.

2.10 ISB DESIGNER

O ISB Designer [Araújo 2012] é uma ferramenta voltada para profissionais de cinema/televisão a fim de auxiliar a criação de conteúdo para TV digital interativa. Ele oferece um storyboard interativo capaz de considerar eventos de adaptação, intervenção e distribuição. Adaptação significa considerar diferentes caminhos da aplicação com base em determinada característica, por exemplo, localização do telespectador. Intervenção considera diferentes caminhos com base em um evento imprevisível, por exemplo, interação do telespectador. E, por fim, distribuição considera a apresentação do conteúdo em diferentes dispositvos.

Seu processo de planejamento e autoria é feito a partir de estruturas lineares, denominadas sequências, nas quais não podem ocorrer eventos de adaptação, intervenção ou distribuição. É possível definir várias sequências lineares que podem ser relacionadas por elos e, assim, definem-se os diferentes caminhos baseados nos eventos já mencionados. As sequências são compostas de painéis, os quais possuem os objetos que serão utilizados pela

aplicação, e é onde o autor pode organizar esses objetos. A ferramenta oferece três visões: rascunho, autoria e narrativa.

A visão de rascunho, exibida na Figura 14, permite a criação de painéis e a especificação de como o leiaute da aplicação vai se modificando com o passar do tempo. Também é possível adicionar anotações abaixo dos painéis, adicionar uma nova sequência ou reutilizar uma sequência já criada. Ao introduzir sequências, elos também são adicionados permitindo navegação entre os diferentes caminhos. A inserção de painéis e/ou sequências é feita através do botão “mais”.

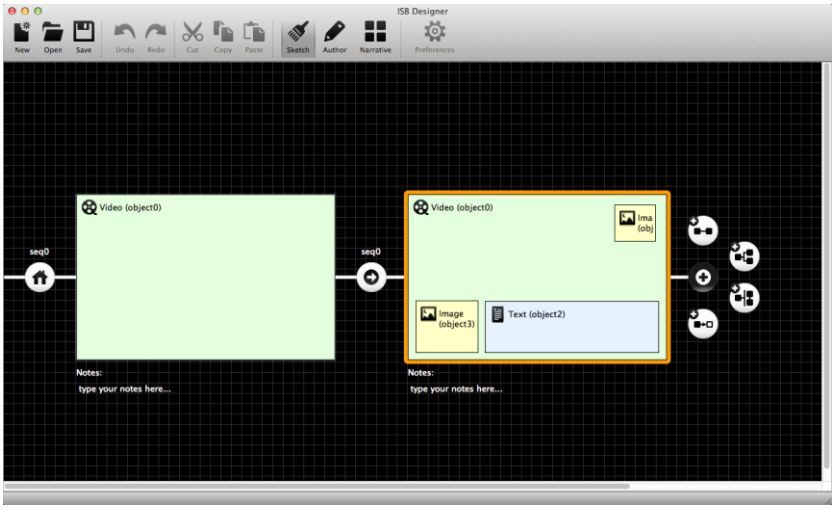


Figura 14: Visão de rascunho da ISB Designer, retirado de [Araújo 2012].

A visão de autoria se refere a detalhes específicos da aplicação final, como a duração de cada painel e a especificação da condição dos elos, por exemplo. Essas condições podem ser sequencial (caminho natural), adaptativa (baseada em uma característica) ou interativa (intervenção do usuário). Nesta visão, os objetos de mídia que serão utilizados também devem ser especificados.

A visão de narrativa, exibida na Figura 15, apresenta a estrutura da aplicação a fim de facilitar sua visualização. A estrutura permite que o autor navegue mais facilmente pelas sequências e tenha uma visualização geral da organização da aplicação.

Embora a ferramenta possua um conversor de código NCL, o código gerado apresenta-se completamente desestruturado por se aproximar do modelo interno da ferramenta, ou seja, não utiliza contextos NCL, que são elementos cuja finalidade é organizar logicamente um documento NCL. Os autores de ISB Designer apontam uma solução para esse problema sugerindo que o autor do documento NCL utilize a visão estrutural do documento, que foi implementada como plugin da ferramenta Composer 3, para editar sua estrutura e inserir manualmente os contextos NCL. A Figura 16 mostra a ferramenta

funcionando como plugin e exibe a nova visão estrutural do Composer 3, no lado inferior direito.

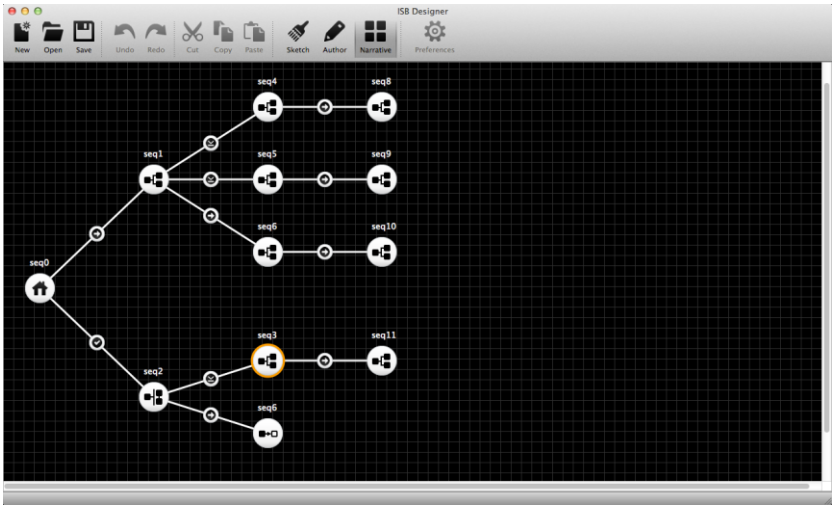


Figura 15: Visão de narrativa da ISB Designer, retirado de [Araújo 2012].

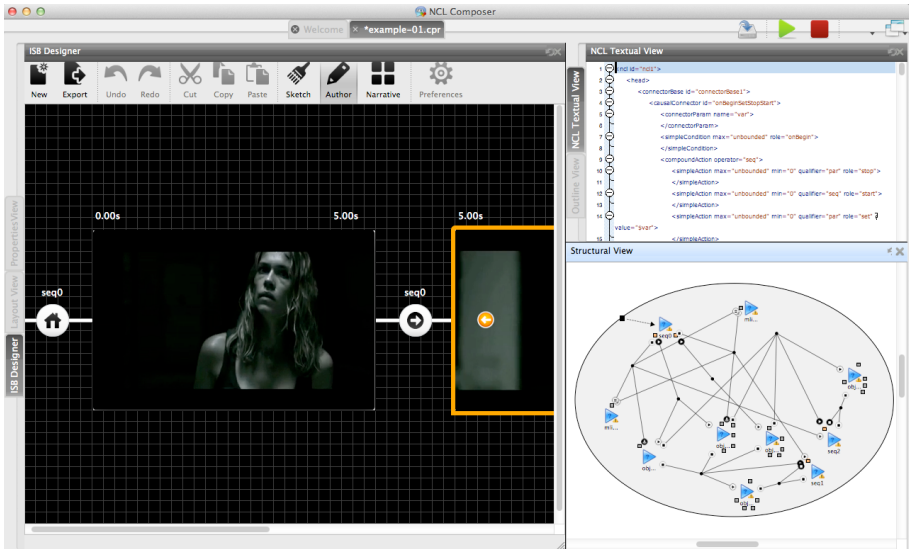


Figura 16: Estrutura de um documento gerado a partir da visão de storyboard, retirado de [Araújo 2012].

Algumas limitações da ferramenta ISB Designer são: não possibilitar a tradução do modelo NCL utilizado pelo Composer 3 para o modelo utilizado pela ISB Designer e a falta de análise comportamental da aplicação.

2.11 REQUISITOS DE UM EDITOR GRÁFICO MULTIMÍDIA

Após a análise das características presentes nos editores estudados, em especial os trabalhos Composer 3 [Lima et al. 2010], DEMAIS [Bailey et al. 2001] e *Template-Based MHP Authoring Tool* [Chiao et al. 2006], foi realizada a seleção de alguns requisitos

considerados como importantes para uma ferramenta de autoria gráfica para aplicações multimídia. Estes requisitos estão listados a seguir:

- Diferentes perfis de usuários: a ferramenta deve alcançar o maior número de usuários possíveis. Desta forma, o conhecimento da linguagem utilizada para a criação da aplicação deixa de ser impactante em sua elaboração;
- Extensibilidade: a ferramenta deve ser capaz de suportar novas funcionalidades de forma fácil e sem comprometer seu desempenho;
- Portabilidade: a ferramenta deve ser capaz de funcionar em diferentes plataformas, independentemente do sistema operacional utilizado;
- Simulações: o usuário deve ser capaz de visualizar/simular o funcionamento do aplicativo que está sendo desenvolvido;
- Templates: a ferramenta deve dar suporte ao uso de templates, agilizando ainda mais o desenvolvimento da aplicação;
- Templates genéricos: é desejável que o conjunto de templates disponíveis na ferramenta possa ser facilmente estendido, permitindo a utilização de outros templates e não apenas daqueles oferecidos pela ferramenta;
- Diferentes visões: a ferramenta deve oferecer diversas visões do aplicativo em desenvolvimento, permitindo, assim, que o usuário escolha a mais agradável e possa, consequentemente, facilitar seu processo de desenvolvimento;
- Análise do documento: ao final da criação da aplicação, o documento final gerado deve ser analisado. Esta análise consiste tanto nas validações sintática e semântica (análise estrutural) como na verificação da consistência temporal da aplicação (análise comportamental). A importância da verificação temporal reside na possibilidade de notificar o autor sobre inconsistências criadas na aplicação, as quais, provavelmente, podem não permitir o comportamento esperado da mesma. Como exemplo de inconsistência, imagine uma aplicação em que o usuário deverá interagir após a apresentação de 20 segundos de um vídeo, mas o mesmo possui apenas 15 segundos. Neste caso, o usuário jamais poderá interagir com a aplicação. Outras situações mais complexas e difíceis de se detectar, como a criação de um loop infinito na apresentação de um objeto de mídia ou a não finalização da aplicação, também devem ser encontradas através desta verificação.

2.12 COMPARAÇÃO DOS TRABALHOS RELACIONADOS

Considerando os requisitos essenciais que uma ferramenta de autoria gráfica para aplicações multimídia deve satisfazer, conforme apresentado na seção anterior, a Tabela 1 indica quais deles são encontrados em cada uma das ferramentas comentadas anteriormente neste capítulo.

O propósito principal desta dissertação é a proposta de um editor gráfico que contemple todos os requisitos identificados como importantes. Em especial, o editor a ser proposto deve permitir a criação de documentos NCL por autores sem conhecimento da linguagem. Portanto, o uso de templates para criação de documentos multimídia e o suporte a templates genéricos são requisitos essenciais que devem ser satisfeitos pelo editor proposto neste trabalho. É importante notar que considerando os trabalhos relacionados a autoria gráfica para documentos NCL, como Composer 3, Berimbau e ISB Designer, nenhum deles oferece a facilidade de uso de templates.

Tabela 1: Requisitos x Editores

Requisitos / Editores	ASCCT	DEMAIS	Composer 3	LAMP	EDITEC	T-B MHP AT	MMPE	Berimbau	ISB Designer
Diferentes perfis de autores	✓		✓	?			✓	✓	
Extensibilidade	?		✓			✓			✓
Portabilidade	?	✓	✓	✓	✓	?			✓
Simulação		✓		✓			✓		
Uso de templates	✓	✓		✓		✓	✓		
Suporte a templates genéricos	✓			✓	✓	✓			
Diferentes visões		✓	✓	✓	✓	✓	✓		✓
Análise de consistência	+/-		+/-			?			

✓ contemplado pelo editor
? não foi possível definir
+/- atende parcialmente

Este capítulo apresentou uma comparação de diversos trabalhos relacionados a autoria gráfica multimídia e identificou um conjunto de requisitos importantes a serem satisfeitos por um editor gráfico multimídia. Como esta dissertação tem como foco a autoria de documentos NCL com uso de templates, as linguagens NCL e XTemplate são apresentadas no próximo

capítulo, explicando-se como é realizada a construção de um documento NCL final com a utilização de um template de composição.

CAPÍTULO 3 – LINGUAGEM NCL

A linguagem NCL (Nested Context Language) [ABNT 2007], que foi adotada pelo Sistema Brasileiro de TV digital através do middleware Ginga-NCL, é uma linguagem declarativa para autoria de documentos multimídia interativos baseada no padrão XML [Extensible Markup Language 2011].

NCL é responsável por definir como os objetos de mídia de uma aplicação interagem entre si, suas relações espacias e temporais e, também, como ocorre a interação do usuário com a aplicação.

Apesar de ser uma linguagem declarativa, NCL permite o uso das linguagens imperativas Lua e Java [Soares et al. 2007] para implementar objetos de mídia que compõem um documento. Como linguagens imperativas são bastante expressivas, através destas é possível inserir códigos responsáveis pela lógica ou pela realização de cálculos aritméticos, por exemplo, em um documento NCL. Como a linguagem NCL não define elementos com esta finalidade, seriam tarefas um tanto trabalhosas, ou até impossíveis de serem realizadas.

Este capítulo apresenta a estrutura geral de um documento NCL, assim como seus principais elementos. A estrutura da linguagem XTemplate, utilizada para especificar templates de composição, também será apresentada. Além disso, será explicado como a linguagem XTemplate pode ser usada em conjunto com a linguagem NCL para criar documentos multimídia.

3.1 ESTRUTURA DE UM DOCUMENTO NCL

Assim como todo documento XML, a primeira linha de um documento NCL deve indicar a versão XML utilizada e o tipo de codificação. Logo em seguida, apresenta-se a estrutura básica de um documento NCL, a qual possui três elementos: ncl <ncl>, cabeçalho <head> e corpo <body>.

O elemento <ncl> possui dois atributos obrigatórios: *id* e *xmlns*. O primeiro representa o identificador do documento, o qual deve ser único em todo o documento, e, o segundo, o perfil de linguagem utilizado [Soares e Rodrigues 2006].

O elemento <head> é composto por elementos que possuem informações sobre o leiaute (layout) e outros elementos da aplicação. Estes elementos são as bases de transições, regras, regiões, descritores e conectores.

O elemento <body> define o conteúdo da aplicação, ou seja, elementos representando os objetos de mídia e elementos que representam seus relacionamentos, tais como elos, contextos e switches.

A Figura 17 apresenta a estrutura básica de um documento NCL.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/BDTVProfile">

    <head>
        <regionBase>
            <!-- Define as regiões da tela onde as mídias são apresentadas -->
        </regionBase>

        <descriptorBase>
            <!-- Define como as mídias são apresentadas -->
        </descriptorBase>

        <connectorBase>
            <!-- Define como ativar os elos e a ação a ser disparada -->
        </connectorBase>
    </head>

    <body>
        <!-- Define portas, nós de mídia, contextos, elos e switches -->
    </body>

</ncl>
```

Figura 17: Estrutura básica de um documento NCL.

3.1.1 CABEÇALHO NCL

A base de regiões, <regionBase>, é composta por elementos <region>, os quais especificam áreas espaciais onde um nó de mídia pode ser exibido. Um documento NCL pode definir várias bases de regiões. Cada base pode conter inúmeras regiões, as quais podem estar aninhadas, uma dentro da outra.

O elemento <regionBase> possui dois atributos opcionais: *id* e *device*. O primeiro identifica a base de regiões, e o segundo, a classe de dispositivos à qual os elementos <region> da base se referem. Quando um documento define mais de uma base, é obrigatório que cada uma delas possua o atributo *id*, para que as mesmas possam ser referenciadas.

Os atributos do elemento <region> podem ser visualizados na Tabela 2, sendo o *id* seu único atributo obrigatório, utilizado nas referências às regiões, e que deve ser único em todo o documento.

Tabela 2: Atributos do elemento <region>.

Atributo	Significado
id	Identificador único da região
left	Coordenada x do lado esquerdo da região
top	Coordenada y do lado superior da região
right	Coordenada x do lado direito da região

bottom	Coordenada y do lado inferior da região
width	Dimensão horizontal da região
height	Dimensão vertical da região
zIndex	Coordenada z, indicando a profundidade da região
title	Título da região

Caso os atributos opcionais não sejam fornecidos, a área total do dispositivo ou a área total da região pai, caso a mesma esteja aninhada, será associada à região. A Figura 18 apresenta o código NCL de quatro regiões (‘blue’, ‘red’, ‘green’ e ‘orange’) e sua visão de leiaute.

```
<regionBase>
    <region id='blue' left='0' top='0' height='360' width='480' zIndex='2'/>
    <region id='red' left='50' bottom='0' height='310' width='250' zIndex='3'>
        <region id='green' left='0' bottom='0' height='50%' width='100%' zIndex='4'/>
    </region>
    <region id='orange' right='0' top='0' height='45' width='150' zIndex='4'/>
</regionBase>
```

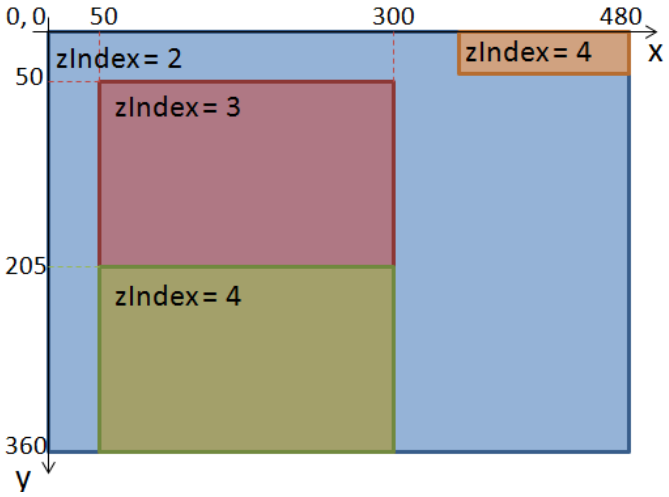


Figura 18: Código NCL de uma base de regiões e sua visão de leiaute.

Na Figura 18, os valores utilizados nos atributos *left*, *top*, *right*, *bottom*, *height* e *width* foram especificados em pixels para as regiões ‘blue’, ‘red’ e ‘orange’. No entanto, estes atributos também podem ser especificados em porcentagem, como feito para a região ‘green’, e, neste caso, os valores dos atributos são relativos aos respectivos valores dos mesmos atributos da região pai.

A base de descritores, <descriptorBase>, é composta por elementos <descriptor>, os quais especificam como um nó de mídia será exibido. Um documento NCL deve conter apenas uma base de descritores e os descritores não podem estar aninhados. Seu único atributo é o *id* e é opcional.

Os atributos básicos do elemento <descriptor> podem ser visualizados na Tabela 3, sendo o *id* seu único atributo obrigatório e que deve ser único em todo o documento. Além

dos atributos listados na Tabela 3, vários outros são definidos pela linguagem NCL para permitir uma navegação por itens que podem estar sendo exibidos na tela. Estes outros atributos podem ser encontrados em [Soares e Barbosa 2009].

Tabela 3: Atributos básicos do elemento <descriptor>.

Atributo	Significado
id	Identificador único do descritor
region	Atributo *id* da região associada ao descritor
explicitDur	Duração do nó de mídia associado ao descritor
freeze	Indica ação a ser tomada ao fim da apresentação do nó de mídia associado ao descritor
player	Indica a ferramenta utilizada para exibir o nó de mídia associado ao descritor

Os descritores também podem definir diversos parâmetros, através do elemento <descriptorParam>, os quais estão relacionados ao tipo de mídia que fará referência ao descritor. Estes são definidos pelo par 'propriedade: valor' e permitem maior flexibilidade, já que os valores desses parâmetros podem variar ao longo da aplicação. A Figura 19 exibe o código NCL de uma base de descritores.

```
<descriptorBase>
    <descriptor id='blueDesc' region='blue' explicitDur='20.0s' freeze='true'/>
    <descriptor id='orangeDesc' region='orange'/>
    <descriptor id='redDesc' region='red' moveDown='2' focusIndex='1'/>
    <descriptor id='greenDesc' region='green' moveUp='1' focusIndex='2'/>
</descriptorBase>
```

Figura 19: Código NCL de uma base de descritores.

Cada um dos descritores da Figura 19 faz referência a uma das regiões da Figura 18. O descritor 'blueDesc' definiu seu atributo *freeze* com o valor 'true', indicando que a mídia que fizer referência a este descritor deverá 'congelar' seu último quadro de apresentação ao final de 20 segundos, já que o atributo *explicitDur* possui o valor 20. O descritor 'orangeDesc' apenas fez referência à região 'orange'. Já os outros descritores definiram o atributo *focusIndex*, o qual permite navegação sobre as mídias apresentadas nas regiões 'red' e 'green', alterando o foco de cada uma delas ao utilizar as teclas direcionais para cima e para baixo do controle remoto.

A base de conectores, <connectorBase>, é composta por elementos <connector>, os quais definem ações que devem ser executadas sobre eventos, quando determinada condição sobre eventos for satisfeita. Conectores são usados por elos, no corpo de um documento NCL, para especificar os relacionamentos entre componentes do documento, a serem apresentados na Seção 3.1.2.

O elemento <connector> possui apenas um atributo obrigatório, *id*, seu identificador único, e ele é composto por papéis de condição e de ação, além de permitir a definição de parâmetros. A definição de um conector é feita através da máquina de estados de eventos, ilustrada na Figura 20. Os tipos de eventos que podem ser usados na definição de um conector são apresentação (*presentation*), seleção (*selection*) e atribuição (*attribution*).

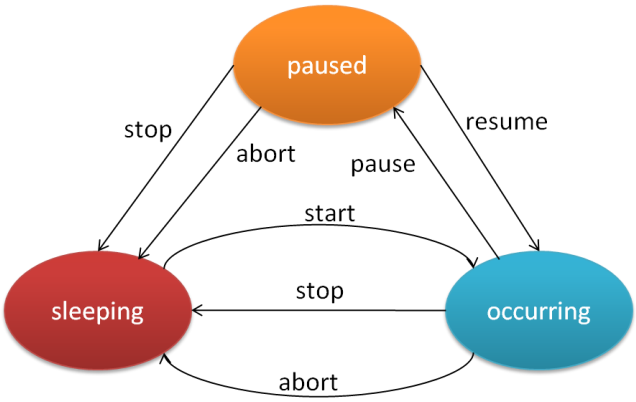


Figura 20: Máquina de estado de eventos.

A Figura 21 representa um elo que relaciona as mídias 1, 2 e 3, ilustrando o conector utilizado. Este conector especifica que quando um participante associado ao papel '*onBegin*' tiver sua apresentação iniciada, os participantes associados ao papel '*startN*' terão suas apresentações iniciadas. O elo ilustrado sincroniza o início das apresentações das mídias 1, 2 e 3.

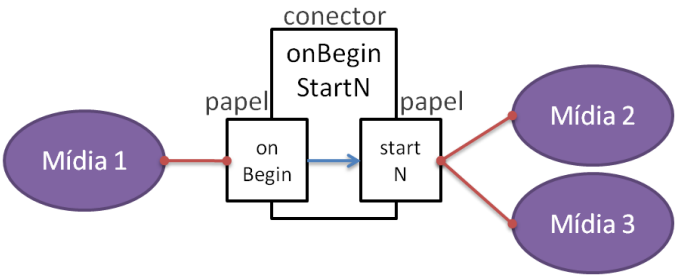


Figura 21: Exemplo de elo NCL.

Como pode ser visualizado na Figura 21, um elo pode interligar diversas mídias e cada uma destas é associada a um papel do conector utilizado pelo elo. Papéis podem ser de condição ou de ação e podem ser simples ou compostos.

Um papel de condição simples, <simpleCondition>, especifica uma condição sobre um determinado evento, baseado em seu estado ou em uma transição de sua máquina de estados. Por sua vez, um papel de ação simples, <simpleAction>, especifica qual ação um evento sofrerá. Ações provocam transições nos estados dos eventos.

Uma condição composta, <compoundCondition>, é formada por duas ou mais condições simples ou compostas, e é representada por uma expressão lógica com os operadores ‘e’ (*AND*) e ‘ou’ (*OR*). Uma ação composta, <compoundAction>, é formada por duas ou mais ações simples ou compostas, que devem ser executadas seguindo uma ordem explícita (*SEQ*) ou em qualquer ordem (*PAR*).

Como o papel de um conector pode ser associado a mais de um participante em um elo, como visto na Figura 21, o conector deve definir um operador, através do atributo *qualifier*, para este papel. Se for uma condição, o operador pode ser *AND* ou *OR*; se for uma ação, o operador pode ser *PAR* ou *SEQ*. Por exemplo, no elo ilustrado na Figura 21, o papel “*startN*’ define que as mídias devem ser executadas na ordem em que foram definidas pelo elo (operador *SEQ*), ou seja, primeiro inicia a mídia 2 e depois a mídia 3.

Ao criarmos um papel, seja de condição ou de ação, alguns atributos precisam ser definidos. Em ambos papéis, o atributo *role* identifica o papel através de um nome. Ele é obrigatório e deve ser único no conector. Além do nome, também é preciso especificar os atributos *eventType* e *transition* para condição ou *actionType* para ação, que representam o tipo de evento associado e a transição/ação na máquina de estados, respectivamente.

Por conveniência, alguns papéis predefinidos são oferecidos pela linguagem. Estes já possuem os atributos obrigatórios definidos e são reconhecidos pelo formatador NCL através de seu nome. Os outros atributos permitem uma maior expressividade ao conector, como esperar determinado tempo até ativar uma condição ou ação, por exemplo. Todos os atributos podem ser verificados na Tabela 4.

Tabela 4: Atributos dos papéis de um conector NCL.

Atributo	Condição	Ação	Significado
role	✓	✓	Nome do papel
eventType	✓	✓	Tipo de evento associado ao papel
transition	✓		Transição na máquina de estados
actionType		✓	Ação que causa transição na máquina de estados
delay	✓	✓	Tempo decorrido para ativar a condição/ação
min	✓	✓	Cardinalidade mínima
max	✓	✓	Cardinalidade máxima
repeat		✓	Número de vezes que a ação deve ser executada
repeatDelay		✓	Tempo decorrido entre cada repetição da ação
qualifier	✓	✓	*AND* ou *OR* para condição e *SEQ* ou *PAR* para ação (apenas se ‘*max’* for maior que 1)

value		✓	Valor a ser atribuído à propriedade do papel (apenas se o tipo de evento for atribuição)
key	✓		Código da tecla do controle remoto (apenas se o tipo de evento for seleção)
duration		✓	Tempo de duração de uma atribuição
by		✓	Incremento ao longo do tempo definido por '*duration*', caso seja maior que zero

Além dos papéis, um conector também pode definir parâmetros, através do elemento <connectorParam>, permitindo uma melhor reutilização do conector em elos distintos. Por exemplo, suponha que queremos que uma mídia tenha sua propriedade *top* alterada para o valor 100 px em um elo, e que, em um outro elo, esta mudança seja realizada para o valor de 200 px. Ao invés de declararmos dois conectores praticamente iguais (a única diferença seria no novo valor a ser atribuído), criamos apenas um conector, que define um parâmetro.

Este parâmetro será utilizado pelo elo, o qual irá definir o novo valor a ser atribuído, neste caso, à propriedade *top*. A Figura 22 mostra o código NCL de um conector causal que define um parâmetro.

```
<causalConnector id='onBeginSet'>
    <connectorParam name='pTop'/>
    <simpleCondition role='onBegin'/>
    <simpleAction role='set' value='$pTop'/>
</causalConnector>
```

Figura 22: Código NCL de um conector.

As bases apresentadas até agora precisam ser declaradas para que uma aplicação NCL seja criada. No entanto, como NCL permite o reúso, é possível importar bases de um outro documento NCL e utilizá-las como se estivessem declaradas no próprio documento. Estas importações podem ser feitas utilizando-se o elemento <importBase>.

O elemento <importBase> possui dois parâmetros obrigatórios: *alias* e *documentURI*. O primeiro indica um apelido para a base e deverá ser utilizado quando algum elemento da base importada for referenciada. O segundo informa a localização do documento NCL que possui a base importada. A Figura 23 mostra o código NCL de um elo (<link>) que utiliza um conector de uma base que foi importada.

```
<connectorBase>
    <importBase alias='connectorBase' documentURI='conBase.ncl'/>
</connectorBase>

<link xconnector='connectorBase#onBeginStart'>
    ...
</link>
```

Figura 23: Elo utilizando um conector de uma base importada.

O parâmetro *xconnector* do elo define a base importada, através do *alias* da mesma, e define o conector a ser utilizado, através do *id* do mesmo, que neste caso é ‘onBeginStart’, separado do *alias* pelo caracter *#*, de uso obrigatório.

Duas outras bases que também podem fazer parte do documento são as bases de regras e de transições. A primeira está relacionada à adaptação do conteúdo de acordo com algumas regras, e a segunda está relacionada à forma de exibição das mídias.

Como exemplo de uso de uma base de regras, podemos imaginar uma aplicação que oferece diferentes idiomas e o telespectador deve escolher o de sua preferência. Existirá, então, uma regra para cada um dos idiomas disponíveis e a escolha do usuário deverá satisfazer apenas a uma das regras, a qual irá adaptar o conteúdo da aplicação.

A base de regras, <ruleBase>, é composta por elementos <rule>, os quais são utilizados pelo elemento <switch>, que será apresentado na Seção 3.1.2. O único parâmetro da base de regras é o *id*, e é opcional.

O elemento <rule> possui quatro parâmetros obrigatórios: *id*, *var*, *comparator* e *value*, onde *id* é o seu identificador; *var* representa a propriedade de um elemento <media> do tipo ’application/x-ncl-settings’ (explicado na Seção 3.1.2); *comparator* indica o tipo de comparação a ser realizada, de acordo com a Tabela 5; e *value*, o valor da variável.

Tabela 5: Operadores de comparação das regras.

Operador	Significado
eq	Igual a
ne	Diferente de
gt	Maior que
lt	Menor que
gte	Maior ou igual a
lte	Menor ou igual a

Já a base de transições, <transitionBase>, é composta por elementos <transition>, os quais especificam efeitos de transição que podem ser utilizados por descritores. Seu único parâmetro é o *id*, e é opcional.

O elemento <transition> possui dois atributos obrigatórios, *id* e *type*, e outros dez atributos opcionais que permitem especificar maiores detalhes sobre a transição, como seu tempo de duração e direção, por exemplo. O *id* identifica o elemento e deve ser único, enquanto *type* define o tipo de transição. Estes tipos podem ser visualizados na Tabela 6.

Tabela 6: Tipos de transição.

Tipo	Significado
barWipe	Varredura simples
irisWipe	Varredura a partir do centro do objeto de mídia
clockWipe	Varredura como o movimento do ponteiro de relógio
snakeWipe	Varredura em espiral
fade	Mudança gradual de cores

A Figura 24 apresenta o código NCL de uma base de transição e de uma base de regras. A base de transição é composta por duas transições, 'transInicio' e 'transFim', cada uma delas apresenta tipo (*type*) e duração (*dur*) diferentes. A base de regras é composta por duas regras, a regra 'pt' com a variável 'leg' com o valor '1' e 'en' com o valor '2'. A regra será satisfeita se a comparação do valor definido pela variável 'leg' (*var*) for satisfeita.

```
<transitionBase>
    <transition id='transInicio' type='fade' dur='2s'/>
    <transition id='transFim' type='barWipe' dur='1s'/>
</transitionBase>

<ruleBase>
    <rule id='pt' var='leg' comparator='eq' value='1'/>
    <rule id='en' var='leg' comparator='eq' value='2'/>
</ruleBase>
```

Figura 24: Código NCL de uma base de transições e uma base de regras.

3.1.2 CORPO NCL

O corpo de um documento NCL, definido pelo elemento <body>, é responsável por conter os nós de mídia que fazem parte do documento e seus relacionamentos, bem como definir o comportamento da aplicação. Ele pode conter os seguintes elementos: <media>, <port>, <link>, <switch> e <context>.

O elemento <media> representa as mídias do documento e seus atributos podem ser visualizados na Tabela 7. Dentre eles, o único obrigatório é o *id*.

Tabela 7: Atributos do elemento <media>.

Atributo	Significado
id	Identificador único da mídia
src	Localização do objeto de mídia
type	Define o tipo de mídia
descriptor	Identifica o descritor que controla a apresentação da mídia
refer	Faz referência a outra mídia previamente definida
instance	Define o grau de relação com a mídia referenciada (apenas quando o atributo *refer* é definido)

O atributo type deve seguir obrigatoriamente o formato *mimetype*, ou seja, uma cadeia de caracteres que define a classe da mídia e um tipo de codificação (como por exemplo, MPEG). Estes *mimetypes* são padronizados pela IANA [IANA 2012].

Além dos tipos mais comuns, como imagem, texto e vídeo, por exemplo, cinco tipos especiais de mídia foram definidos para utilização em documentos NCL. Estes tipos são:

- application/x-ginga-NCL ou application/x-ncl-NCL: indica que a mídia é um documento NCL;
- application/x-ginga-NCLua ou application/x-ncl-NCLua: indica que a mídia é um programa Lua;
- application/x-ginga-NCLet ou application/x-ncl-NCLet: indica que a mídia é um programa Java;
- application/x-ginga-settings ou application/x-ncl-settings: deve ser único no documento e é responsável por definir atributos globais do mesmo. Estes atributos podem ser referenciados através do atributo var do elemento <rule>;
- application/x-ginga-time ou application/x-ncl-time: deve ser único no documento e seu conteúdo é o *Universal Time Coordinated* (UTC), seguindo a sintaxe Ano:Mês:Dia:Hora:Minuto:Segundo.Fração.

O elemento <media> também pode definir dois tipos de interface: as âncoras de conteúdo, <area>, e de propriedade, <property>. A primeira define um trecho da mídia, e a segunda define propriedades da mesma.

Estas interfaces podem ser manipuladas pelos elos, seja para permitir várias sincronizações entre as mídias ou por alterar o valor de suas propriedades durante a execução da aplicação. Os atributos das âncoras de conteúdo podem ser visualizados na Tabela 8 e os de propriedade na Tabela 9. Apenas os atributos *id* e *name* são obrigatórios, nas âncoras de conteúdo e de propriedade, respectivamente.

Tabela 8: Atributos do elemento <area>.

Atributo	Significado
id	Identificador único da âncora
begin	Início da âncora (em segundos)
end	Término da âncora (em segundos)
first	Início da âncora (em quadros ou amostras)
last	Término da âncora (em quadros ou amostras)
text	Indica um texto (apenas em mídias texto)
position	Posição de ocorrência do texto definido no atributo *text*

coords	Coordenadas de um polígono (apenas mídias visuais)
label	Identificador da âncora no arquivo de origem
clip	Identificador de um trecho de uma cadeia temporal

Tabela 9: Atributos do elemento <property>.

Atributo	Significado
name	Nome da propriedade ou grupo de propriedades
value	Valor inicial da propriedade ou grupo de propriedades

A Figura 25 apresenta o código NCL de uma mídia com duas âncoras. O elemento é do tipo imagem, com extensão *png*, e utiliza o descritor com *id* igual a 'blueDesc'. Sua âncora de conteúdo, 'a1', começa em 2 segundos e termina em 12 segundos. Sua âncora de propriedade atribui o valor 10 para a propriedade *top* da mídia, ou seja, não importa o valor *top* definido pela região que é referenciada pelo descritor 'blueDesc' (vide Figura 19), seu valor inicial será 10.

```
<media id='fundo' src='C:\julia\fundo.png' type='image/png' descriptor='blueDesc'>
    <area id='a1' begin='2.0s' end='12.0s'/>
    <property name='top' value='10'/>
</media>
```

Figura 25: Código NCL de um elemento <media>.

O elemento <port> é um ponto de interface do corpo do documento e é utilizado para indicar qual mídia irá inicializar a aplicação, por isso, pelo menos um elemento <port> precisa ser definido.

Ele possui três atributos: *id* e *component*, que são obrigatórios, e *interface*, opcional. O primeiro indica seu identificador único, o segundo faz referência a uma das mídias que foram declaradas no corpo e, o terceiro, a uma interface da mídia referenciada, caso exista.

O elemento <link>, elo, é responsável por definir os relacionamentos de sincronismo entre as mídias do documento e seu comportamento está relacionado ao conector que é referenciado pelo mesmo. O <link> possui apenas dois atributos, *id*, opcional, e *xconnector*, obrigatório e responsável por referenciar um dos conectores da base de conetores do documento.

Um elo é composto por elementos <bind>, os quais fazem referências aos papéis do conector referenciado e aos nós do documento (mídias, contextos e switches). Além disso, ele também pode ter como filhos o elemento <linkParam>, que define um parâmetro através do par [propriedade, valor]. E, assim como o elo, o elemento <bind> também pode definir parâmetros, através do elemento <bindParam>, com o mesmo objetivo. Entretanto, estes

parâmetros só podem ser definidos caso o conector referenciado também possua parâmetros, com o elemento <connectorParam>.

O elemento <bind> possui três atributos, sendo dois obrigatórios, *role* e *component*, e um opcional, *interface*. O significado destes atributos podem ser visualizados na Tabela 10.

Tabela 10: Atributos do elemento <bind>.

Atributo	Significado
role	Faz referência a um dos papéis do conector referenciado pelo elo
component	Faz referência a um nó do documento
interface	Faz referência a alguma interface do nó referenciado em *component*

A Figura 26 apresenta o código NCL de um elo que faz referência ao conector 'onBeginStartN' e define três elementos <bind>. O elo define que quando a interface 'a1' da mídia 'fundo' começar, as mídias 'image' e 'text' devem ser iniciadas.

```
<link id='startProg' xconnector='onBeginStartN'>
    <bind role='onBegin' component='fundo' interface='a1'/>
    <bind role='start' component='image'/>
    <bind role='start' component='text'/>
</link>
```

Figura 26: Código NCL de um elemento <link>.

O elemento <switch> serve para adaptar o conteúdo de uma aplicação NCL. Ele apresenta um conjunto de componentes sendo que apenas um deles será escolhido para ser apresentado e esta escolha varia de acordo com regras que são estipuladas na base de regras do documento.

Ele é composto por um conjunto de elementos <bindRule>, <media>, <context> e <switch>. O primeiro é responsável por mapear cada um dos componentes do switch a uma das regras encontradas na base de regras do documento, enquanto os outros definem os componentes alternativos do switch.

O elemento <switch> possui dois parâmetros, *id*, representando seu identificador único e obrigatório, e *refer*, o qual permite fazer referência a outro switch do documento e é opcional. O elemento <bindRule> possui dois parâmetros obrigatórios: *constituent* e *rule*. O primeiro faz referência a um dos componentes do switch e, o segundo, a uma das regras da base de regras do documento.

O switch pode, ainda, utilizar os elementos <defaultComponent> e <switchPort>. O primeiro define qual dos componentes do switch será inicializado caso nenhuma das regras seja satisfeita, e o segundo deve ser utilizado quando deseja-se fazer referência a uma das interfaces dos componentes do switch.

O elemento <defaultComponent> possui um atributo obrigatório, *component*, que indica qual das mídias é usada por default. Já o <switchPort> possui apenas o *id* como atributo obrigatório e é composto por elementos <mapping>. Cada <mapping> faz referência a um dos componentes do switch, através do atributo *component* (obrigatório), e a uma das interfaces do componente, caso exista, através do atributo *interface* (opcional).

A Figura 27 exibe o código NCL de um switch. Este switch define duas mídias texto e cada uma delas está associada a uma regra da base, vide Figura 24, através do elemento <bindRule>. Além disso, caso nenhuma das regras seja atendida, a mídia exibida será 'port', como definido pelo elemento <defaulComponent>.

```
<switch id='legendas'>
    <bindRule rule='pt' constituent='port'/>
    <bindRule rule='en' constituent='ingl'/>
    <defaultComponent component='port'/>

    <media id='port' src='C:\Julia\Leg\port.html' type='text/html'/>
    <media id='ingl' src='C:\Julia\Leg\ingl.html' type='text/html'/>
</switch>
```

Figura 27: Código NCL de um elemento <switch>.

Finalmente, o elemento <context> permite a criação de contextos no documento. Estes são utilizados para atribuir uma melhor organização lógica ao documento. Um contexto pode definir os mesmos elementos definidos no <body>, que é um contexto 'especial', pois não precisa definir o atributo *id* (o mesmo é herdado do documento), obrigatório para qualquer outro contexto do documento.

Além do atributo *id*, o contexto também possui o atributo *refer*, opcional, permitindo referenciar um outro contexto do documento. Contextos podem estar aninhados e seus elementos filhos só podem ser referenciados através de portas definidas no próprio contexto.

A Figura 28 representa a visão estrutural do elemento <body> com dois contextos, 'context1' e 'context2'. Além dos contextos, <body> possui quatro mídias, das quais duas são referenciadas por portas do mesmo. O elemento 'context1', composto por três mídias, não possui nenhuma porta, então nenhum de seus elementos internos podem ser referenciados por elos existente no <body>. Já 'context2' possui uma porta, a qual é referenciada por um dos elos de <body>. A Figura 29 apresenta o código NCL do elemento <body> representado na Figura 28.

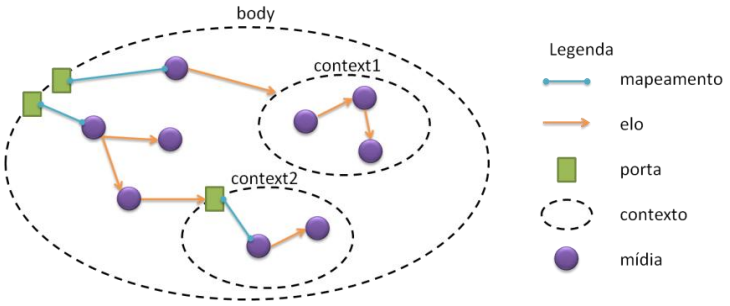


Figura 28: Representação de contextos e seus componentes.

```
<body>

    <port id='pFundo' component='fundo'/>
    <port id='pMusic' component='sound'/>

    <media id='fundo' src='C:\julia\fundo.png' type='image/png' descriptor='blueDesc'>
        <area id='a1' begin='2.0s' end='12.0s'/>
        <property name='top' value='10'/>
    </media>

    <media id='sound' src='C:\Julia\Músicas\IMissYou.mp3' type='audio/mp3'/>
    <media id='image' src='C:\Julia\Documents\image2.png' type='image/png'/>
    <media id='text' src='C:\Julia\Documents\txt.html' type='text/html'/>

    <context id='context1'>
        <media id='med1' src='C:\Julia\Documents\vdo.mpeg' type='video/mpeg'/>
        <media id='med2' src='C:\Julia\Documents\okTxt.html' type='text/html'/>
        <media id='med3' src='C:\Julia\Documents\optTxt.html' type='text/html'/>
        <link xconnector='onBeginStartN'>
            <bind role='onBegin' component='med3'/>
            <bind role='start' component='med1'/>
        </link>
        <link xconnector='onBeginStartN'>
            <bind role='onBegin' component='med1'/>
            <bind role='start' component='med2'/>
        </link>
    </context>

    <context id='context2'>
        <port id='portFig1' component='fig1' interface='anc'/>
        <media id='fig1' src='C:\Julia\Documents\fig1.gif' type='image/gif'>
            <area id='anc' begin='5.0s'/>
        </media>
        <media id='fig2' src='C:\Julia\Documents\fig2.png' type='image/png'/>
        <link xconnector='onBeginStartN'>
            <bind role='onBegin' component='fig1' interface='anc'/>
            <bind role='start' component='fig2'/>
        </link>
    </context>

    <link id='startProg' xconnector='onBeginStartN'>
        <bind role='onBegin' component='fundo' interface='a1'/>
        <bind role='start' component='image'/>
        <bind role='start' component='text'/>
    </link>

    <link id='startCtx1' xconnector='onEndStart'>
        <bind role='onEnd' component='sound'/>
        <bind role='start' component='context1'/>
    </link>

    <link id='startCtx2' xconnector='onEndStart'>
        <bind role='onEnd' component='context1'/>
        <bind role='start' component='context2' interface='portFig1'/>
    </link>

</body>
```

Figura 29: Código NCL do elemento <body> e seus elementos filhos.

3.2 TEMPLATES DE COMPOSIÇÃO COM AS LINGUAGENS XTEMPLATE E NCL

Um template de composição hipermídia é responsável por especificar a estrutura de um documento multimídia, definindo de forma genérica seus componentes (mídias ou composições) e como eles se relacionam. Um documento hipermídia que utiliza um template reúsa estruturas já especificadas, tornando sua autoria mais simples.

A linguagem XTemplate 3.0 [Dos Santos e Muchaluat-Saade 2011] permite a definição de templates de composição hipermídia utilizando o padrão XML [Extensible Markup Language 2011] e pode ser usada em conjunto com a linguagem NCL 3.0. Em sua terceira versão, a linguagem XTemplate define um template através de quatro partes principais: cabeçalho, vocabulário, corpo e restrições, sendo todas opcionais, com exceção do vocabulário.

O cabeçalho é responsável por importar os elementos <descriptorBase> e <connectorBase> de um documento NCL, os quais serão utilizados na especificação de características de apresentação e de relacionamentos dos componentes, respectivamente. Além disso, ele também pode estender definições contidas em outros templates, através do elemento <extend>.

O vocabulário utiliza o elemento <component> para especificar seus componentes, os quais podem possuir pontos de interface ou outro <component> aninhado. Ele também é responsável por definir as relações que serão utilizadas pelo template através do elemento <connector>. Além disso, é possível definir um número mínimo e máximo de ocorrências para cada elemento do vocabulário.

O corpo define os pontos de interface da composição, cada uma das instâncias dos componentes do vocabulário e suas interfaces e os relacionamentos entre esses componentes. Ele pode, ainda, declarar variáveis, com o elemento <variable>, e instruções para realizações de repetições, com o elemento <for-each>, conforme especificados pelo padrão XSLT [W3C 1999B].

As restrições permitem adicionar restrições sobre os elementos do vocabulário através do elemento <constraint>. Cada restrição possui uma expressão na linguagem XPath [W3C 1999a] e retorna um valor booleano.

Como é possível que um template estenda outro template, todas as definições do template estendido serão incluídas no template que o estende, com exceção das restrições, as quais serão ou não sobrescritas de acordo com o atributo *overwriteConstraints* do elemento <extend>.

Como mencionado anteriormente, XTemplate 3.0 permite a declaração de variáveis com o elemento <variable>. Este elemento possui um atributo, *select*, que permite definir o valor da variável através de uma expressão XPath. Este permite, ainda, determinar o valor de parâmetros de elos usados em suas condições ou ações.

Como exemplo, a Figura 30 e a Figura 31 apresentam as partes principais de um template que especifica uma propaganda de supermercado interativa chamado 'propagandaSupermercado'. O template permite a apresentação de três produtos e seus respectivos preços (mídias de imagem), os quais devem ser sincronizados com as interfaces de uma mídia de áudio. Durante a apresentação dos produtos, o telespectador pode selecioná-los, através do botão 'ENTER' do controle remoto. Ao final da exibição dos produtos, aqueles que foram selecionados pelo telespectador são exibidos. A definição completa do template pode ser encontrada no Anexo A.

Conforme mencionado, um template de composição especifica componentes genéricos, ou seja, a identificação desses componentes é responsabilidade do autor que fará uso do template. Os relacionamentos de sincronização entre esses componentes já estão especificados no template.

Para que um nó de composição de um documento hipermídia utilize um template, este documento deve fazer referência ao template a ser utilizado. Então, a linguagem NCL precisou ser estendida a fim de permitir o uso de templates de composição. Após fazer referência ao template e ter seus componentes definidos, o documento NCL estendido [Dos Santos e Muchaluat-Saade 2011], com referência ao template a ser utilizado, deve ser processado a fim de gerar o documento NCL final padrão (sem referência ao template), o qual poderá ser executado no formatador NCL.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xtemplate id="propagandaSupermercado" ...>
3
4 <head>
5     <connectorBase>
6         <importBase alias="connBase" documentURI="connectorBase.ncl"/>
7     </connectorBase>
8
9     <descriptorBase>
10        <importBase alias="descBase" documentURI="descriptorBase.ncl"/>
11    </descriptorBase>
12 </head>
13
14 <vocabulary>
15    <component xlabel="fundo" xtype="image/png" descriptor="descBase#dpTV" maxOccurs="1"/>
16    <component xlabel="audio" xtype="audio/mp3" descriptor="descBase#dpAudio" maxOccurs="1">
17        <port xlabel="prodX" minOccurs="3" maxOccurs="3"/>
18    </component>
19    ...
20
21    <connector src="connBase#onBeginStartN" xlabel="onBeginStartN" maxOccurs="unbounded"/>
22    <connector src="connBase#onEndStopN" xlabel="onEndStopN" maxOccurs="unbounded"/>
23    <connector src="connBase#onKeySelectionPresentingSet" xlabel="onKeySelectionPresentingSet"
          maxOccurs="unbounded"/>
24    ...
25 </vocabulary>
```

Figura 30: Cabeçalho e vocabulário do template 'Propaganda de supermercado'.

```
<body>
    <port id="pFundo" component="fundo"/>
    <media id="compras" xlabel="compras" type="application/x-ginga-settings">
        <property name="compProd1" value ="false" xlabel="compProd1"/>
        <property name="compProd2" value ="false" xlabel="compProd2"/>
        <property name="compProd3" value ="false" xlabel="compProd3"/>
    </media>

    <!-- INICIO -->
    <link xtype="onBeginStartN">
        <bind role="onBegin" select="child::*[@xlabel='fundo']"/>
        <bind role="start"   select="child::*[@xlabel='audio']"/>
        <bind role="start"   select="child::*[@xlabel='botao']"/>
    </link>

    <!-- EXIBE PRODUTOS -->
    <variable name="i" select="1"/>
    <for-each select="child::*[@xlabel='audio']/child::area[@xlabel='prodX']">
        <link xtype="onBeginStartN">
            <bind role="onBegin" select="current()"/>
            <bind role="start"   select="child::*[@xlabel='produto'][position() = $i]"/>
            <bind role="start"   select="child::*[@xlabel='preco'][position() = $i]"/>
        </link>

        <link xtype="onEndStopN">
            <bind role="onEnd" select="current()"/>
            <bind role="stop"   select="child::*[@xlabel='produto'][position() = $i]"/>
            <bind role="stop"   select="child::*[@xlabel='preco'][position() = $i]"/>
        </link>
        <variable name="i" select="$i + 1"/>
    </for-each>
    ...
<body>

<constraints>
    <constraint select="count(child::media[@xlabel='produto']) = 3" description="O número de produtos deve ser igual a três"/>
    <constraint select="count(child::media[@xlabel='preco']) = 3" description="O número de preços deve ser igual a três"/>
    <constraint select="count(descendant::area[@xlabel='prodX']) = 3" description="O número de âncoras (prodX) do áudio deve
        ser igual a três"/>
</constraints>
</xtemplate>
```

Figura 31: Corpo e restrições do template ‘Propaganda de supermercado’.

A Figura 32 apresenta o documento NCL estendido, com referência ao template ‘Propaganda de supermercado’. O documento possui o elemento <templateBase>, que é composto pelo elemento <importBase>, responsável por indicar a base de templates utilizada. O corpo do documento indica qual template é utilizado, através do atributo *xtemplate*, e cada um de seus componentes indica que componente do template eles representam, através do atributo *xlabel*. Note que nenhum elo é definido nesse documento NCL.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!-- Generated with NCL API -->

<ncl id='propaganda'>
    <head>
        <templateBase>
            <importBase alias='template_propagandaSupermercado' documentURI='C:\Users\julia\propSuper.xml'/>
        </templateBase>
    </head>

    <body xtemplate='template_propagandaSupermercado'>
        <media id='fundo_1' xlabel='fundo' src='C:\Users\julia\Documents\fundo.png' type='image/png'/>
        <media id='audio_1' xlabel='audio' src='C:\Users\julia\Documents\jingle.mp3' type='audio/mp3'>
            <area id='prodX_1' xlabel='prodX' begin='9.0s' end='12.0s'/>
            <area id='prodX_2' xlabel='prodX' begin='13.0s' end='17.0s'/>
            <area id='prodX_3' xlabel='prodX' begin='18.0s' end='21.0s'/>
        </media>
        <media id='produto_1' xlabel='produto' src='C:\Users\julia\Documents\prod1.png' type='image/png'/>
        <media id='preco_1' xlabel='preco' src='C:\Users\julia\Documents\preco1.png' type='image/png'/>
        <media id='botao_1' xlabel='botao' src='C:\Users\julia\Documents\botao.png' type='image/png'/>
        <media id='produto_2' xlabel='produto' src='C:\Users\julia\Documents\prod2.png' type='image/png'/>
        <media id='preco_2' xlabel='preco' src='C:\Users\julia\Documents\preco2.png' type='image/png'/>
        <media id='produto_3' xlabel='produto' src='C:\Users\julia\Documents\prod3.png' type='image/png'/>
        <media id='preco_3' xlabel='preco' src='C:\Users\julia\Documents\preco3.png' type='image/png'/>
    </body>
</ncl>
```

Figura 32: Documento NCL com referência ao template ‘Propaganda de supermercado’.

A Figura 33 apresenta o processamento de um documento que utiliza um template de composição. O processador utiliza como entradas o template e o documento NCL estendido, que por sua vez utiliza o referido template, e gera como saída o documento NCL padrão,

pronto para ser executado em seu formatador. O processador analisa o cabeçalho do template para obter as URIs das bases a serem utilizadas no documento NCL final, adiciona os componentes e relacionamentos com base nas definições genéricas do template e valida as restrições do mesmo, gerando, assim, o documento NCL final.

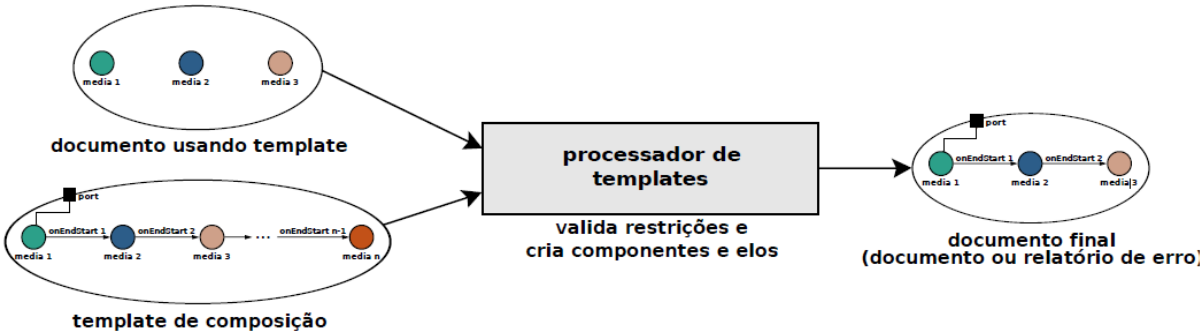


Figura 33**: Processamento de documentos usando templates de composição, retirado de [Dos Santos 2012].**

O plugin para uso de templates, proposto por esta dissertação, que será apresentado no Capítulo 6, utiliza o processador desenvolvido em [Dos Santos 2012] para gerar o documento NCL padrão final.

Este capítulo apresentou brevemente a linguagem NCL e a linguagem XTemplate, utilizadas como base para a proposta de editor gráfico multimídia realizada por esta dissertação. O próximo capítulo apresenta o editor proposto, chamado NEXT - NCL *Editor supporting XTemplate*.

CAPÍTULO 4 – NEXT: EDITOR GRÁFICO DE DOCUMENTOS NCL COM SUPORTE A TEMPLATES DE COMPOSIÇÃO

Este capítulo apresenta o editor NEXT – NCL *Editor supporting XTemplate* [Silva e Muchaluat-Saade 2012], proposto para facilitar o desenvolvimento de aplicações para TV digital que utilizam a linguagem NCL versão 3.0, permitindo o uso de templates de composição. Os templates de composição, apresentados no Capítulo 3, Seção 3.2, são utilizados pelo plugin de templates, o qual será apresentado no Capítulo 6.

A fim de contemplar os requisitos básicos destacados no Capítulo 2, Seção 2.11, o editor foi desenvolvido em Java, possibilitando seu uso em diversas plataformas, ou seja, sua portabilidade é garantida uma vez que apenas a instalação da JVM (*Java Virtual Machine*) é necessária para seu funcionamento. Além disso, esta escolha também permite a utilização de outras ferramentas já implementadas em Java em um mesmo ambiente, como a API aNaa - *API for NCL Authoring and Analysis* [Dos Santos 2012], responsável por realizar a análise de um documento NCL, e o EDITEC, editor gráfico para criação de templates, a ser adaptado como plugin para o NEXT em um trabalho futuro.

Sua extensibilidade é possível através do uso de uma arquitetura baseada em um núcleo e diversos plugins. O conjunto de funcionalidades do NEXT pode ser estendido através da adição de novos plugins. Com a utilização de um núcleo, que é capaz de se comunicar com os diferentes plugins instalados, é possível isolar as funcionalidades específicas do editor das dos plugins.

A utilização da ferramenta por diferentes perfis de autores pode ser contemplada através de plugins específicos para cada um destes perfis. Até o momento, foram desenvolvidos três plugins que requerem conhecimento da linguagem NCL e, um outro, que não requer conhecimento da linguagem, o qual permite a criação de uma aplicação com a utilização de templates. Os plugins já disponíveis serão apresentados no Capítulo 5 e no Capítulo 6.

Com relação às diferentes visões, NEXT oferece as visões estrutural e de leiaute, as quais foram implementadas como plugins, e também possui uma visão textual, entretanto, a mesma não é editável.

NEXT oferece ainda a possibilidade de criar uma pasta final com o documento NCL e todas as mídias necessárias para que a aplicação possa ser executada em uma implementação do formatador NCL. As mídias selecionadas são copiadas de seu local de origem para a pasta

final e o documento NCL é atualizado em relação à localização das mídias que foram copiadas.

Ao final da construção do documento NCL, é possível realizar uma análise do mesmo com a utilização da API aNaa [Dos Santos 2012].

A arquitetura do editor NEXT, sua implementação, as características de sua interface gráfica e as limitações da implementação atual serão apresentadas nas próximas seções.

4.1 ARQUITETURA

O trabalho realizado em [Lima et al. 2010] mostrou a importância de alguns requisitos não-funcionais relevantes para uma ferramenta de autoria, que foram considerados na arquitetura do NEXT.

A fim de manter a independência das funcionalidades do editor em relação às dos plugins, foi desenvolvido um núcleo capaz de representar o documento NCL e, também, de se comunicar com os plugins, mantendo o sincronismo entre eles. A Figura 34 ilustra a arquitetura do editor NEXT, composta pelo núcleo e um conjunto de plugins. O diagrama de pacotes do núcleo do editor pode ser visualizado na Figura 35.

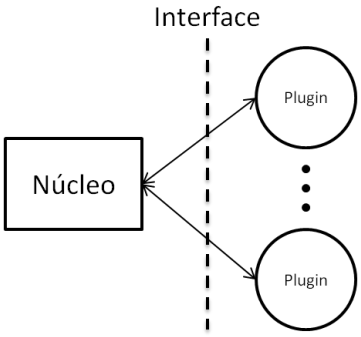


Figura 34. Arquitetura do editor NEXT.

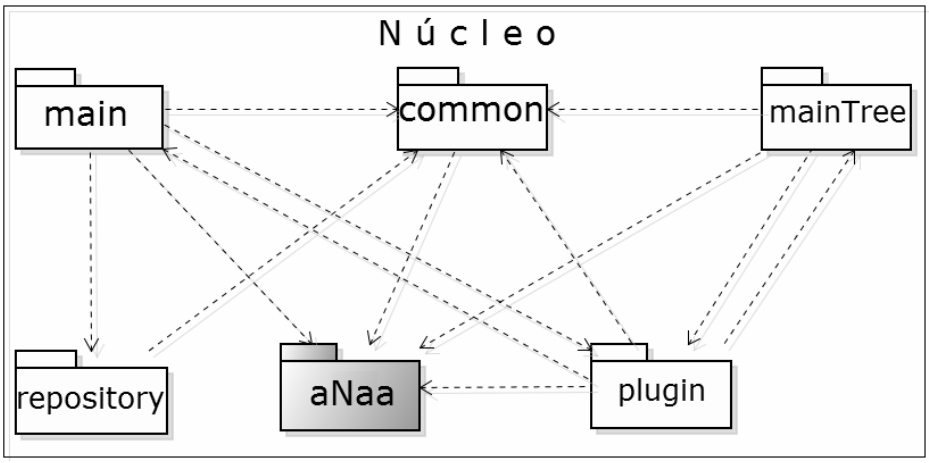


Figura 35: Núcleo do NEXT.

Além de ser responsável pela interface gráfica do editor, o núcleo do NEXT também trata da consistência do documento através do uso da API aNaa. Além disso, quando algum plugin realiza alterações no documento, a API avisa ao NEXT sobre a alteração, e este, por sua vez, se encarrega de avisar aos outros plugins que estão sendo utilizados.

O uso da aNaa, além de permitir a criação de novos documentos NCL, também possibilita a edição de qualquer documento NCL, ou seja, é possível abrir qualquer arquivo com a extensão ncl, independentemente de onde este tenha sido criado, e modificá-lo.

O núcleo é composto pelos pacotes *main*, *mainTree*, *common*, *plugin* e *repository*, e pela API aNaa, que é adicionada à biblioteca. E, como pode ser visto na Figura 34, o nucleo é completamente independente dos plugins. A seguir, é dada uma explicação de cada pacote componente do núcleo.

A API aNaa é responsável por criar objetos Java para representar cada um dos elementos de um documento NCL. A API também fornece métodos para criação e edição desses elementos, os quais devem ser utilizados pelos plugins para que alterações no documento sejam realizadas. Além disso, ela também é responsável pela análise do documento ao final da criação. O objeto do tipo *NCLDoc* criado pela API é analisado e, caso algum erro seja encontrado, o mesmo é informado ao autor. aNaa é capaz de encontrar erros de sintaxe, de referência e também especificações temporais inadequadas [Dos Santos 2012].

aNaa permite a leitura de documentos NCL, através de seu método *load*, assim como a escrita, através do método *parse*. Desta forma, NEXT não precisa de uma estrutura de dados adicional para manipular documentos NCL, mais ainda, o uso da API permite que NEXT seja capaz de manipular qualquer documento NCL independentemente de onde este tenha sido criado.

Com os objetos criados pela aNaa, o pacote *mainTree* cria uma representação do documento NCL em forma de árvore, facilitando a localização dos elementos no documento e a visualização da estrutura do documento no editor.

O pacote *main* é responsável pela interface gráfica do editor. Ele implementa botões para criação e edição de documentos NCL, além de permitir a inserção e remoção de plugins na ferramenta.

O pacote *common* permite a utilização do editor em diferentes idiomas e implementa alguns métodos auxiliares que são utilizados por outros pacotes da ferramenta.

O pacote *plugin* possui métodos para o núcleo se comunicar com os plugins. Desta forma, é possível colher informações destes, como por exemplo, a lista de elementos NCL de interesse do plugin, assim como mantê-los informados sobre alterações no documento. Ele

também é responsável pela criação de um arquivo XML de configuração do editor, que possui informações sobre os plugins que foram instalados e, portanto, que devem ser oferecidos ao usuário quando o programa for iniciado.

O pacote *repository* é responsável pela implementação do repositório de mídias. Ele organiza as mídias inseridas pelo usuário em forma de árvore para facilitar a exibição das mesmas.

4.2 IMPLEMENTAÇÃO

Esta seção descreve, com maiores detalhes, a implementação de cada um dos pacotes do núcleo do NEXT, representados na Figura 35, com exceção da API aNaa. Maiores informações sobre a API podem ser encontradas em [Dos Santos 2012] e em [Dos Santos et al. 2012].

4.2.1 PACOTE MAIN

O pacote *main* é responsável pela criação da interface gráfica do NEXT. Sua classe principal, *Main*, inicia o editor exibindo uma caixa de diálogo onde o usuário deverá escolher um dos idiomas oferecidos pelo editor. Atualmente, são oferecidos os idiomas português e inglês. Após selecionado o idioma, a classe *MainWindow* é chamada para exibir a janela principal do programa, no idioma escolhido pelo usuário.

A Figura 36 apresenta o diagrama de classes do pacote *main*.

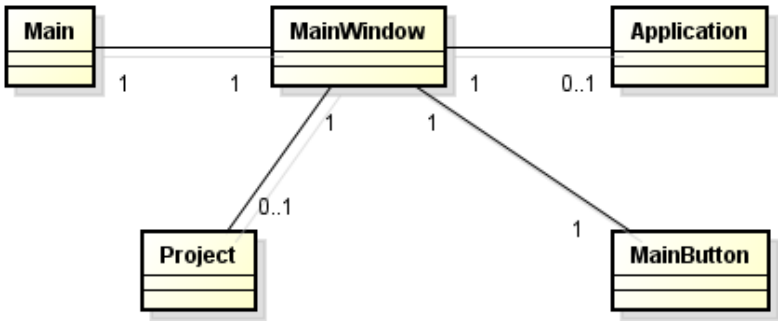


Figura 36: Diagrama de classes do pacote *main*.

MainWindow estende a classe *JFrame* do pacote swing do Java [Java 2012], o qual implementa classes e métodos para a criação de interface gráfica. Assim, ela é responsável pela interface inicial do NEXT e pela criação do menu de opções. Além disso, ela também cria uma pasta, com o nome ‘NEXT’, para o editor e, dentro desta, cria outra pasta, chamada ‘Plugins’, a qual é responsável por armazenar os plugins que foram adicionados ao editor.

MainButton estende a classe *JPanel*, também do pacote swing, e é responsável apenas pela exibição gráfica dos botões do editor, uma vez que a funcionalidade dos mesmos é implementada pela classe *MainWindow*.

A classe *Project* é responsável por permitir a exibição da árvore criada com base no documento NCL que está sendo criado/editado e pela exibição dos diversos plugins selecionados pelo usuário. Ela possui as informações referentes ao documento sendo editado e é responsável por oferecer estas informações à classe *MainWindow* quando o usuário desejar salvar o documento.

Application foi criada para oferecer a possibilidade de criar uma pasta apenas com o documento NCL e as mídias do documento, alterando, portanto, o atributo *src* das mesmas no documento.

Quando o usuário cria um nó de mídia, é preciso que o caminho do arquivo com o conteúdo deste nó seja informado no documento NCL e, muitas vezes, estas mídias podem estar distribuídas pelo disco rígido do computador do usuário. De forma a organizar e garantir que todas as mídias estarão disponíveis quando o usuário for executar a aplicação em um formatador NCL, a classe *Application* cria uma pasta que conterá o documento NCL e uma outra pasta, com o nome ‘medias’, contendo uma cópia de todas as mídias que foram utilizadas no documento pelo usuário.

4.2.2 PACOTE MAINTREE

Este pacote é responsável por lidar com o documento NCL em si. A classe *MainTree* cria a representação do documento em forma de árvore e, através dos métodos disponibilizados pela API aNaa, cria as partes essenciais do documento, o cabeçalho e o corpo. Já o preenchimento destas partes é de responsabilidade dos plugins. A Figura 37 apresenta o diagrama de classes do pacote *mainTree*.

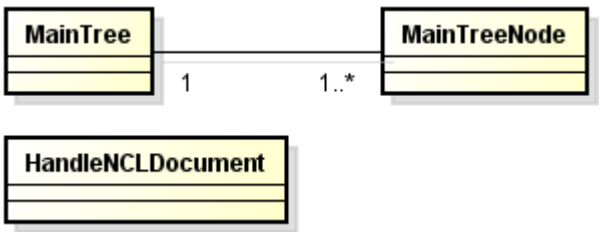


Figura 37: Diagrama de classes do pacote *mainTree*.

MainTree também implementa a classe *NCLModificationListener* da API, assim ele é capaz de saber quando alguma alteração no documento é realizada e informar à classe *HandleNCLDocument* sobre a modificação.

A classe *HandleNCLDocument*, por sua vez, deve informar a modificação realizada aos diversos plugins que o usuário pode estar utilizando. Ele mantém uma lista dos plugins que estão sendo utilizados e verifica se o elemento alterado é de interesse do plugin, se for, o mesmo será notificado.

A classe *mainTreeNode* estende a classe *DefaultMutableTreeNode* a fim de permitir a criação de um objeto com informações adicionais àquelas que são utilizadas por objetos *DefaultMutableTreeNode*. Estas informações adicionais englobam, por exemplo, o tipo de elemento NCL que o nó representa e o seu *id* (identificador), caso tenha.

4.2.3 PACOTE PLUGIN

O pacote *Plugin* é composto por apenas duas classes e, como o próprio nome indica, este pacote é responsável pelo tratamento dos plugins inseridos no editor NEXT.

A classe *Plugin* é responsável por criar objetos do tipo *Plugin*, um para cada plugin adicionado à ferramenta. Estes objetos possuem informações do plugin, como, por exemplo, o seu *alias*, utilizado na lista de plugins que é exibido no menu Ferramentas, o arquivo *jar* do mesmo, se o plugin está sendo utilizado ou não, entre outras.

Além disso, a classe *Plugin* é responsável por inicializar o plugin quando o mesmo é escolhido pelo usuário e por criar um documento XML que armazena informações sobre os plugins que já foram inseridos. Este documento é criado pela classe *PluginXML* e apresenta a estrutura ilustrada na Figura 38.

```
<NCLCreatorPlugins>

    <plugin id="1">
        <pluginAlias>Create connectors</pluginAlias>
        <pluginFile>CONNECTOR4.jar</pluginFile>
        <path>NEXT\Plugins\CONNECTOR4.jar</path>
        <interest>Connector,ConnectorBase</interest>
    </plugin>

</NCLCreatorPlugins>
```

Figura 38: Documento XML com informações sobre os plugins inseridos no NEXT.

Cada um dos plugins inseridos recebe um identificador único (*id*), um apelido (<pluginAlias>), o arquivo jar (<pluginFile>) do mesmo e a lista com os elementos NCL de interesse do plugin (<interest>).

Toda vez que o editor for executado este arquivo é lido para que os plugins previamente adicionados sejam disponibilizados para o usuário.

4.2.4 PACOTE COMMON

O *common* é composto por quatro classes, as quais fornecem métodos auxiliares que são utilizados por outras classes do NEXT. As classes são: *Constants, MultiLanguage, NEXTException* e *MyButton*.

A classe *Constants* define o *enum Element*, o qual classifica os diferentes elementos de um documento NCL. Os plugins do NEXT devem criar sua lista de interesses de acordo com o tipo *Element*, montando uma lista de elementos *Element* para informar ao NEXT seus interesses. Além disso, ele define também o *enum ModType*, que define os tipos de modificações realizadas no documento (‘adição’, ‘remoção’ ou ‘modificação’).

A classe *MultiLanguage* é responsável pelos idiomas oferecidos. Ao iniciar o programa, ela recebe o idioma escolhido pelo usuário e toda frase exibida pelo editor é informada pela classe, já no idioma escolhido. Para habilitar outro idioma no editor, basta defini-lo no *enum Language* da classe e traduzir as frases do programa para o novo idioma. Todas as frases do editor encontram-se nos *enums* da *MultiLanguage*.

A classe *NEXTException* estende *Exception* com o objetivo de passar informações sobre exceções e/ou erros que possam ocorrer para as classes superiores, para que essas possam tratá-las ou informar ao usuário sobre o problema ocorrido.

A classe *MyButton* estende a classe *JButton* e serve apenas para criar botões com a mesma aparência, exibidos pela interface do NEXT.

4.2.5 PACOTE REPOSITORY

Este pacote é responsável por implementar as classes que cuidam do repositório de mídias. O diagrama de classes do pacote pode ser visualizado na Figura 39.

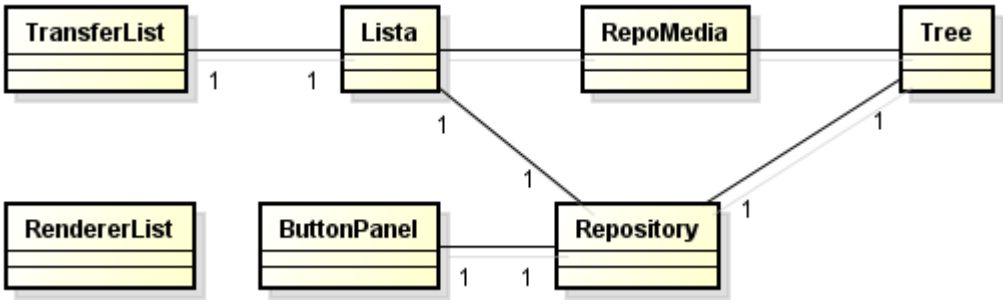


Figura 39: Diagrama de classes do pacote *repository*.

A classe *Repository* estende uma *JInternalFrame* para que o repositório possa ser apresentado na mesma região dos plugins. Ela é responsável pela interface gráfica do repositório, por apresentar a árvore de mídias, onde estas são separadas conforme o tipo,

apresentar os botões para manipular o repositório e por exibir as mídias em si, as quais ficam armazenadas em uma lista.

A classe *ButtonPanel* cria o *JPanel* com os botões do repositório e implementa as ações dos mesmos. São três botões: um para adicionar mídias, um para excluir as mídias selecionadas e outro para remover todas as mídias do repositório.

A classe *Tree* é responsável por criar a árvore de mídias e por atualizá-la. Ela cria um nó para cada mídia e, além da árvore, também atualiza a lista de mídias conforme as modificações realizadas pelo usuário. Os nós da árvore podem ser utilizados para selecionar um tipo de mídia específico, e, assim, apenas as mídias do tipo selecionado serão exibidas para o usuário.

A classe *RepoMedia* cria a representação gráfica das mídias. No caso de ser uma figura, a mesma é exibida, em qualquer outro caso, uma imagem predefinida é apresentada, uma para cada tipo de mídia exibida pela árvore. Enquanto que a classe *RendererList* implementa *ListCellRenderer* para informar como as listas devem apresentar seus elementos, do tipo *RepoMedia*.

A classe *TransferList* estende um *TransferHandler* a fim de permitir que as mídias do repositório sejam arrastadas para os plugins. Os quais, para recebê-las, também devem implementá-la, conforme será explicado no Capítulo 5, Seção 5.1.

4.3 INTERFACE GRÁFICA

NEXT oferece diferentes opções de idiomas, atualmente português e inglês, e representa o documento NCL em forma de árvore, possibilitando uma melhor visualização do documento, de forma organizada. Os elementos do documento ocupam diferentes ramos na árvore de acordo com o seu tipo. O código NCL do documento também pode ser visualizado no editor.

A Figura 40 apresenta a interface gráfica que é oferecida pelo NEXT, juntamente com a visão textual de um documento NCL.

Na parte superior da Figura 40, existe um menu com funções comuns como abrir, salvar e sair. Além disso, algumas dessas opções também são oferecidas através de ícones na barra superior do editor. A Tabela 11 apresenta as funcionalidades dos botões oferecidos pelo NEXT.

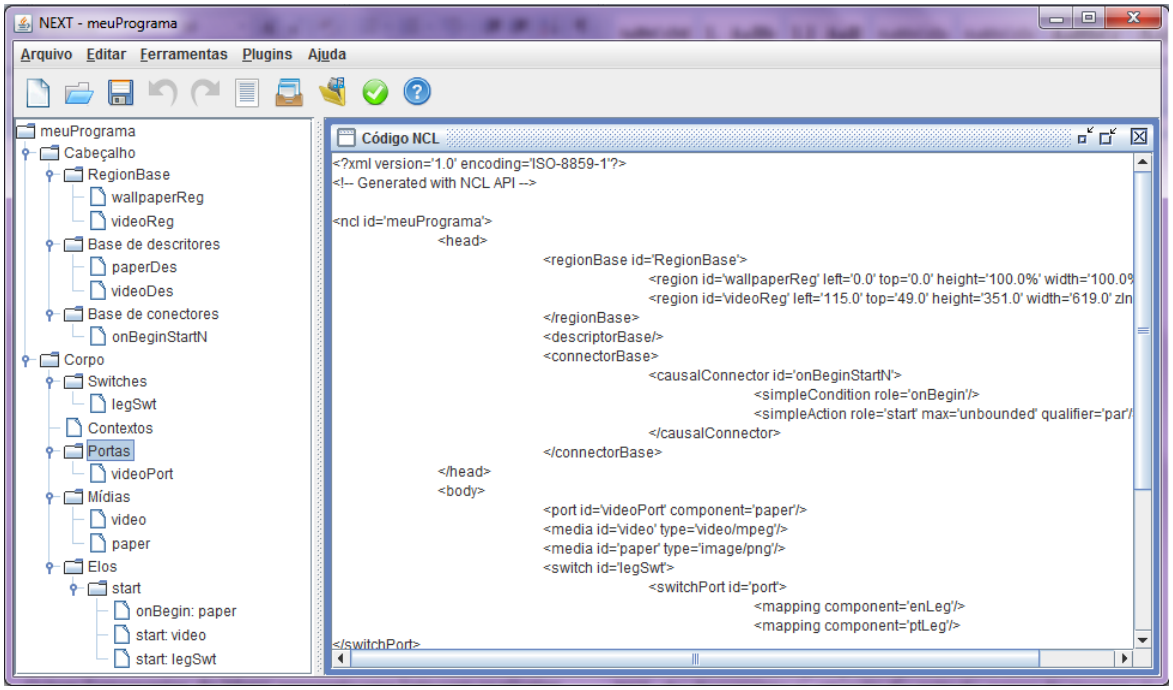


Figura 40: Interface gráfica do NEXT.

Tabela 11: Funcionalidades dos botões do NEXT.

Botão	Funcionalidade
	Criar novo documento
	Abrir documento
	Salvar documento
	Desfazer última ação (ainda não disponível)
	Refazer última ação (ainda não disponível)
	Exibir código NCL do documento
	Construir aplicação
	Abrir repositório de mídias
	Analisar documento
	Exibir menu de ajuda

O item do menu chamado Plugins permite a inserção e remoção de plugins, além de exibir uma lista com os plugins instalados no editor. Para escolher um plugin, basta que o autor o selecione na lista. O mesmo será exibido na área central do programa, que na Figura 40 está exibindo o código NCL do documento.

O item Ferramentas do menu apresenta a opção para analisar o documento e para abrir o repositório de mídias. O repositório pode ser utilizado a qualquer momento durante a criação do documento, já a análise do documento deve ser realizada quando a autoria do documento for concluída.

O repositório de mídias tem o intuito de permitir que o autor organize as mídias a serem utilizadas no documento. O repositório as agrupa de acordo com seu tipo, mas também permite que todas sejam visualizadas simultaneamente. As mídias do repositório podem ser arrastadas e liberadas (*drag-and-drop*) nos plugins. Entretanto, é importante salientar que o plugin deve ser capaz de receber a mídia, o que não depende do NEXT.

O repositório pode ser visualizado na Figura 41, onde todas as mídias são apresentadas simultaneamente e, para exibir apenas um tipo de mídia, basta que o autor selecione na árvore à esquerda o tipo desejado.

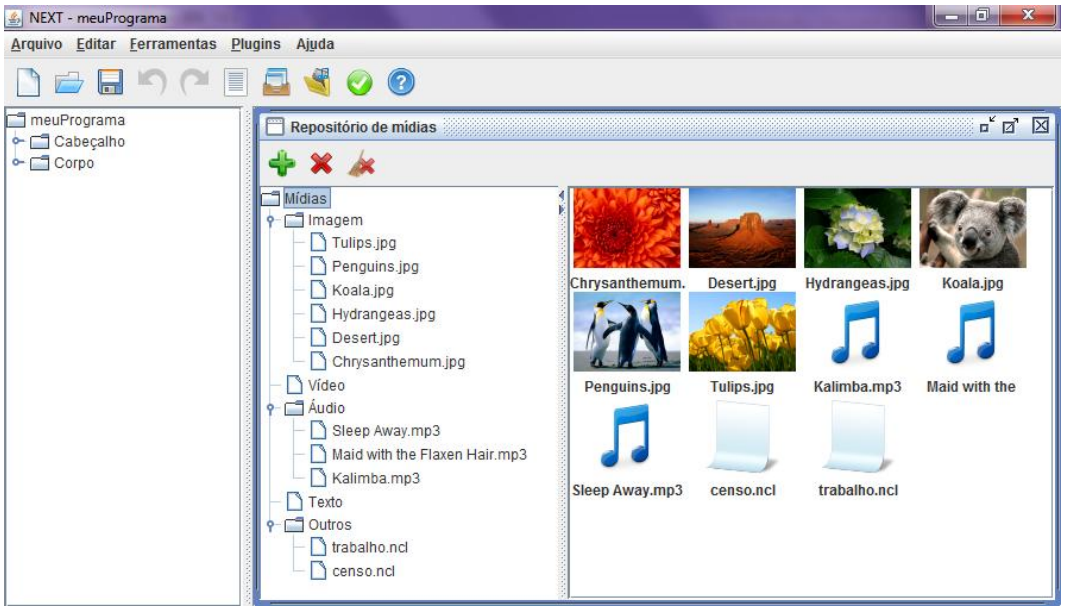


Figura 41: Repositório de mídias do NEXT.

4.4 TESTE DE USABILIDADE

A fim de verificar a usabilidade do NEXT e de seus plugins, três testes de usabilidade foram realizados. O primeiro teste consistiu em verificar a usabilidade do plugin para criação de conectores e será apresentado no Capítulo 5, Seção 5.3.1. Já o segundo, permitiu avaliar a usabilidade do plugin para uso de templates, sendo apresentado no Capítulo 6, Seção 6.3. Por fim, o último teste avaliou os plugins de visão de leiaute e de visão estrutural. Também neste último teste, foram avaliadas algumas características gerais do NEXT, as quais serão apresentadas nesta seção.

Este terceiro teste foi realizado com um grupo de 15 alunos que deveriam construir uma aplicação NCL completa com o uso do editor NEXT. Para a realização do teste, o perfil de usuário procurado era aquele que possuía conhecimentos da linguagem NCL, independentemente do grau exato de conhecimento. O grupo foi composto por alunos de graduação e de doutorado, ambos da computação.

A aplicação consistiu de uma propaganda de supermercado interativa, onde eles deveriam sincronizar a imagem de três produtos e seus respectivos preços com âncoras de uma mídia de áudio e o telespectador poderia selecionar o produto quando o mesmo fosse exibido através do botão 'ENTER' do controle remoto. Além disso, uma imagem informando se algum produto havia sido selecionado também deveria ser exibida ao longo da propaganda. Ao final da apresentação dos produtos, as imagens daqueles que foram selecionados, caso houvesse, deveriam ser exibidas. As mídias a serem utilizadas foram disponibilizadas assim como a base de conectores, já contendo os conectores que seriam necessários para criar a aplicação.

Após criar a aplicação, os alunos responderam um formulário composto por perguntas sobre os plugins utilizados e o editor NEXT. As questões eram objetivas e as opções de resposta variavam numa escala de seis opções (onde zero era a pior e cinco a melhor opção).

A Tabela 12 apresenta as perguntas gerais relacionadas ao editor NEXT em si e sobre o conhecimento do aluno sobre a linguagem NCL. Os resultados foram separados de acordo com o conhecimento do aluno sobre a linguagem NCL. Assim sendo, os alunos que marcaram a opção 2 ou menor na questão sobre conhecimento da linguagem (9 alunos), estão representados na coluna 'Baixo' (baixo conhecimento) e, os outros (6 alunos), na coluna 'Bom' (bom conhecimento). A tabela apresenta a média das respostas por grupo de usuários e o desvio padrão.

Tabela 12: Teste de usabilidade do NEXT.

Pergunta	**Média / Desvio padrão**		**Conclusão do resultado**	
	Baixo	Bom	Baixo	Bom
Como você classifica o seu conhecimento em NCL?	1 / 1	4 / 1	Muito pouco	Muito alto
Você já criou alguma aplicação NCL?	3 Sim 6 Não	Sim, todos	-	-
Com relação à importância do repositório de mídias do NEXT, que nota você daria à mesma?	3 / 1,6	3 / 2,3	Importante	Importante
Você conseguiu criar a aplicação NCL	8 Sim	5 Sim	-	-

completa?	1 Não	1 Não		
Sua aplicação funcionou na máquina virtual conforme o esperado?	5 Sim 4 Não	2 Sim 4 Não	-	-

Durante a execução do terceiro teste de usabilidade foi possível verificar que o uso dos plugins fica muito complicado quando não se tem um bom conhecimento sobre a linguagem NCL. Todos os alunos com baixo conhecimento da linguagem tiveram bastante dificuldade para criar a aplicação completa e precisaram de ajuda durante seu desenvolvimento. Entretanto, estes relataram que a dificuldade era devido ao baixo conhecimento sobre a linguagem NCL e não ao utilizar os plugins e o editor em si.

Das aplicações criadas que não funcionaram como esperado, a maioria apresentou problemas relacionados às definições das regiões. Muitos definiram regiões maiores do que a dimensão da tela utilizada na máquina virtual Ginga-NCL, causando a não visualização de algumas mídias. Erros como a falta de definição de descritores nos elementos de mídia também ocorreram.

Dos alunos com bom conhecimento da linguagem, apenas um não conseguiu finalizar a aplicação, pois o mesmo tentou criar regras compostas e o plugin de visão estrutural não possui esta funcionalidade. Ele também tentou alterar o descritor de uma mídia através do elemento <bind> do elo e não conseguiu.

Seguem algumas observações e sugestões indicadas pelos alunos sobre o NEXT: implementar a opção desfazer; realizar testes de usabilidade mais simples e, progressivamente, aumentar a dificuldade dos testes.

As sugestões referentes aos plugins de visão de leiaute e de visão estrutural serão apresentadas nas Seções 5.2.1 e 5.4.1, respectivamente.

Os questionários utilizados nos testes de usabilidade podem ser encontrados no Anexo B.

4.5 LIMITAÇÕES DO NEXT

NEXT foi desenvolvido de forma a atender os requisitos básicos de um editor gráfico multimídia conforme apresentado no Capítulo 2, Seção 2.11. Dentre os requisitos listados, NEXT não atende apenas à simulação da aplicação em desenvolvimento. O desenvolvimento de um plugin com a devida funcionalidade é um trabalho futuro.

Uma característica importante do NEXT é a possibilidade de editar qualquer documento NCL, mesmo que este não tenha sido criado pelo NEXT. Isto é possível graças à utilização da API aNaa, que cria objetos Java para representar o documento NCL e seus

elementos. Entretanto, algumas limitações da própria API aNaa acarretam algumas limitações ao editor.

A API cria um objeto para cada elemento do documento, entretanto, não há nenhuma representação para os comentários do mesmo. Assim sendo, não é possível adicionar, e nem salvar, caso existam, comentários no documento.

Como visto no Capítulo 3, um documento NCL é composto por três partes básicas: elemento <ncl>, cabeçalho e corpo. Entretanto, é possível criar um documento sem o corpo. Neste caso, o documento pode ser referenciado por outros documentos, os quais poderão importar as bases declaradas no documento sem corpo. Isto é permitido para que seja possível reutilizar o código NCL.

Uma limitação da aNaa é a leitura de um documento que faz referência a outro documento NCL, como por exemplo, um documento importando uma base de regiões. Como a API não é capaz ler documentos com referências externas, esta não é capaz de criar o objeto que representa o documento e, consequentemente, o NEXT não é capaz de exibir o conteúdo do mesmo. Uma forma de tentar contornar esta situação no NEXT seria implementando um trecho de código para descobrir se o arquivo que se deseja abrir no editor faz alguma referência a outro documento e, em caso afirmativo, copiar a parte de código que é referenciada para o arquivo que se deseja abrir.

Como é muito comum encontrar documentos fazendo referência a uma base de conectores, foi implementado um código para tratar desta referência específica. Ou seja, atualmente, NEXT é capaz de abrir um documento que faça referência a uma base de conectores, substituindo parte do documento a ser aberto pela base referenciada. Entretanto, qualquer outro tipo de referência não é tratada e o NEXT apenas informa que não é possível carregar o arquivo, exibindo uma mensagem de erro que é gerada pela aNaa.

Uma outra limitação relacionada à referência é o fato da API não conseguir ler um documento apenas com o cabeçalho NCL caso o mesmo contenha uma base de regras, ainda que o documento não faça nenhuma referência externa a outro documento. Neste caso, o problema encontra-se na forma como o objeto Java é criado pela API. As regras da base precisam referenciar, através de seu atributo *var*, interfaces de uma mídia do tipo *application/x-ginga-settings* ou *application/x-ncl-settings*, entretanto, a mesma não será encontrada já que o documento não possui corpo e uma exceção será lançada pela API.

aNaa foi desenvolvida em Java para que seu uso fosse possível em diferentes plataformas. Entretanto, a funcionalidade de análise do comportamento temporal ainda necessita de alguns ajustes para rodar no sistema operacional Windows e, por este motivo, a

verificação da consistência temporal não pode ser executada nesta plataforma. Entretanto, as validações sintáticas e semânticas funcionam normalmente, garantindo, portanto, a criação de um documento consistente. Já nos sitemas operacionais MAC e Linux, estas funcionalidades operam normalmente.

Apesar das limitações apresentadas, o uso da API no NEXT possibilita a criação de documentos NCL completos e de acordo com a norma da linguagem. Além disso, a API aNaa está sendo refinada para resolver estas limitações e possibilitar o uso de todas as facilidades que são oferecidas pela linguagem.

Este capítulo apresentou a arquitetura geral do editor NEXT, proposto por esta dissertação, comentando seus principais módulos e funcionalidades. No Capítulo 5 será explicado como é feita a criação de plugins para o NEXT.

CAPÍTULO 5 – PLUGINS DESENVOLVIDOS PARA O NEXT

A fim de validar a utilização de plugins no NEXT, foram desenvolvidos plugins para criação de descritores e regiões (visão de leiaute) e conectores, além de um plugin que oferece a visão estrutural de contextos NCL. Apesar de facilitar/agilizar a autoria de documentos NCL, estes plugins requerem que o autor possua conhecimento da linguagem NCL.

Entretanto, como um dos principais objetivos do editor é possibilitar sua utilização por autores sem nenhum conhecimento da linguagem, um plugin que permite a utilização de templates de composição na criação do documento também foi desenvolvido e será apresentado no Capítulo 6.

Este capítulo apresenta como desenvolver plugins para o NEXT, abordando as classes e os métodos obrigatórios que devem ser implementados. Além disso, também serão apresentados os plugins para criação/edição de documentos NCL desenvolvidos para autores que já possuem conhecimento da linguagem, onde cada um deles é responsável por modificar partes específicas do documento. Estes plugins são: visão de leiaute, editor de conectores e visão estrutural. Ao final da apresentação de cada plugin, serão apresentados os resultados dos testes de usabilidade realizados para cada plugin e as limitações de cada um.

5.1 IMPLEMENTAÇÃO DE PLUGINS PARA O NEXT

Plugins são programas, geralmente leves, que adicionam funcionalidades e recursos específicos a um programa principal hospedeiro. Estes normalmente são de fácil instalação e não devem comprometer o desempenho, nem o funcionamento normal de seu hospedeiro.

No NEXT, eles são responsáveis por realizar alterações no documento NCL que está sendo criado e/ou editado. Um plugin pode realizar mudanças no documento inteiro ou apenas em parte dele, esta questão ficará a cargo de seu desenvolvedor.

Como é preciso prover comunicação entre o plugin e o núcleo, para que um plugin possa se comunicar com o núcleo do NEXT, este deverá, obrigatoriamente, utilizar um pacote chamado *myPlugin*, o qual deverá implementar as classes *StartPlugin* e *PluginInfo*.

A classe *StartPlugin*, que obrigatoriamente estende a classe *JInternalFrame*, é responsável por iniciar o plugin e será chamada pelo núcleo do NEXT quando o usuário desejar utilizá-lo. Desta forma, é possível que o usuário utilize diferentes plugins ao mesmo tempo, onde cada um será exibido em sua *JInternalFrame*.

Já a classe *PluginInfo* é responsável por fornecer ao núcleo informações sobre o plugin. Ela é composta pelos métodos *getAlias* e *getNCLInterest*. O primeiro método, informa

o alias do plugin, ou seja, o nome que será exibido pelo editor para representar o plugin. Já o segundo método, deve retornar uma lista com os elementos NCL de interesse do plugin. Sendo assim, sempre que um dos elementos NCL desta lista sofrer alguma alteração que não tenha sido ocasionada pelo próprio plugin, o mesmo receberá uma notificação de adição, remoção ou edição do elemento.

Para que os plugins possam ser notificados sobre as alterações no documento, é necessário que a classe *StartPlugin* também implemente o método *getUpdateAtNCLDocument*. Este método recebe três parâmetros, o primeiro do tipo *ModType*, o segundo do tipo *NCLElement* e o último é um *array* formado por elementos do tipo *Object*. O primeiro parâmetro informa o tipo de alteração (adição, remoção ou modificação), o segundo informa o elemento NCL alterado, e o terceiro apresenta um *array* cujo tamanho depende do tipo de alteração. Caso a alteração tenha sido uma adição ou uma remoção, o *array* terá apenas um elemento que indicará o elemento pai que teve um filho inserido ou removido. Já no caso de uma alteração, o *array* será composto por três elementos, o primeiro indicará o atributo modificado, o segundo, seu valor antigo e, o terceiro, seu novo valor.

De posse das informações de alterações providas pelo NEXT e realizando as devidas providências, o plugin consegue manter consistência com o documento, que pode estar sendo editado por outro plugin do editor.

Para realizar alterações no documento NCL, o plugin deverá utilizar o objeto do tipo *HandleNCLDocument*, passado como parâmetro no construtor da classe *StartPlugin*. Aplicando-se o método *getNCLDocument* neste objeto, o programador receberá um objeto, do tipo *NCLDoc*, que representa o documento NCL de acordo com a API aNaa. Possuindo este objeto, as alterações no mesmo deverão ser aplicadas utilizando-se os métodos que a API disponibiliza.

A Figura 42 apresenta um exemplo de código Java de um plugin apenas com as importações, pacotes, classes e métodos necessários.

Para incorporar um plugin ao editor NEXT, o programador deverá criar um arquivo utilizando a extensão jar. A inserção de um plugin ao editor NEXT é feita durante sua execução. Basta utilizar a opção “Adicionar plugin”, oferecida no item Plugins do menu principal do NEXT. Uma janela será aberta e basta que o usuário informe o arquivo jar correspondente ao plugin. O NEXT se encarrega de criar uma cópia do arquivo e o insere na lista de plugins instalados, exibidos no item Plugins do menu.

```
package myPlugin;

import common.Constants.ModType;         // Constantes de tipo de alteração
import common.Constants.Element;         // Elementos NCL do tipo enum
import common.MultiLanguage.Language;    // Idiomas oferecidos pelo NEXT
import mainTree.HandleNCLDocument;       // Representa o documento NCL

public class StartPlugin extends JInternalFrame{
    public StartPlugin (Language language, HandleNCLDocument ne){
        ...
    }

    public void getUpdateAtNCLDocument (ModType type,
                                NCLElement elem, Object[]values){
        ...
    }
}

public class PluginInfo{
    public static String getAlias(){
        return "PluginAlias";   // Nome que será exibido no menu de plugins
    }

    public static List<Element> getNCLInterest(){
        List<Element> elements = new ArrayList();
        elements.add(Element.REGION);
        return elements;
    }
}
```

Figura 42: Exemplo de código Java de um plugin interessado apenas no elemento region.

É importante salientar que, caso o plugin utilize bibliotecas que não sejam utilizadas pelo NEXT, o seu arquivo jar deve incluir essas bibliotecas, uma vez que o NEXT só realiza a cópia do arquivo jar. Qualquer dependência deste que não esteja embutida no pacote não será copiada e, portanto, o plugin poderá não ser executado corretamente.

Como dito anteriormente, o NEXT oferece um repositório de mídias do tipo *drag-and-drop*, ou seja, permite que suas mídias sejam arrastadas e liberadas em outro lugar, possivelmente, em algum plugin. Entretanto, para que as mesmas possam ser recebidas pelo plugin é necessário que o plugin crie uma classe para tratar o evento *drop*, a qual deve estender *TransferHandler* e implementar o método *importData*.

A Figura 43 apresenta a classe *Transfer*, que foi implementada pelo plugin Visão Estrutural (o qual será apresentado na Seção 5.4) para receber e tratar uma mídia do repositório.

```
1 public class Transfer extends TransferHandler{
2
3     @Override
4     public boolean canImport(TransferSupport supp){
5         if(!supp.isDataFlavorSupported(DataFlavor.stringFlavor))
6             return false;
7         return true;
8     }
9
10    @Override
11    public boolean importData(TransferSupport supp){
12        Transferable t = supp.getTransferable();
13        try {
14            String path = (String) t.getTransferData(DataFlavor.stringFlavor);
15
16            if(path != null && !path.isEmpty()){
17                Component cmp = supp.getComponent();
18                if(cmp instanceof Media)
19                    ((Media)cmp).editMediaSrc(path);
20                else if(cmp instanceof DrawPanel)
21                    ((DrawPanel)cmp).addMedia(path);
22                return true;
23            }
24            return false;
25
26        } catch (UnsupportedFlavorException ex) {
27            Logger.getLogger(Transfer.class.getName()).log(Level.SEVERE, null, ex);
28        } catch (IOException ex) {
29            Logger.getLogger(Transfer.class.getName()).log(Level.SEVERE, null, ex);
30        }
31        return false;
32    }
33 }
```

Figura 43: Implementação de classe para receber uma mídia do repositório.

Como pode ser visto na linha 14 da Figura 43, o objeto que o plugin recebe, na verdade, é uma *string*, a qual possui o caminho completo da mídia que foi arrastada pelo autor.

Apesar do repositório ter sido criado com a finalidade de ajudar no desenvolvimento da aplicação, nenhum plugin é obrigado a tratar o evento *drop*, ficando, assim, impossibilitado de receber qualquer mídia diretamente do repositório.

As próximas seções descrevem os plugins de visão de leiaute, editor de conectores e visão estrutural.

5.2 VISÃO DE LEIAUTE

As regiões em um documento NCL determinam o espaço e a localização onde as mídias da aplicação devem ser exibidas. Já os descritores podem fazer referências às regiões e definem como as mídias devem ser apresentadas, como por exemplo, definindo a duração em que as mesmas serão apresentadas.

O plugin de visão de leiaute, apresentado na Figura 44, foi desenvolvido a fim de facilitar a criação de bases de regiões, regiões e descritores em um documento NCL. Assim sendo, sua lista de interesse engloba apenas estes três elementos NCL.

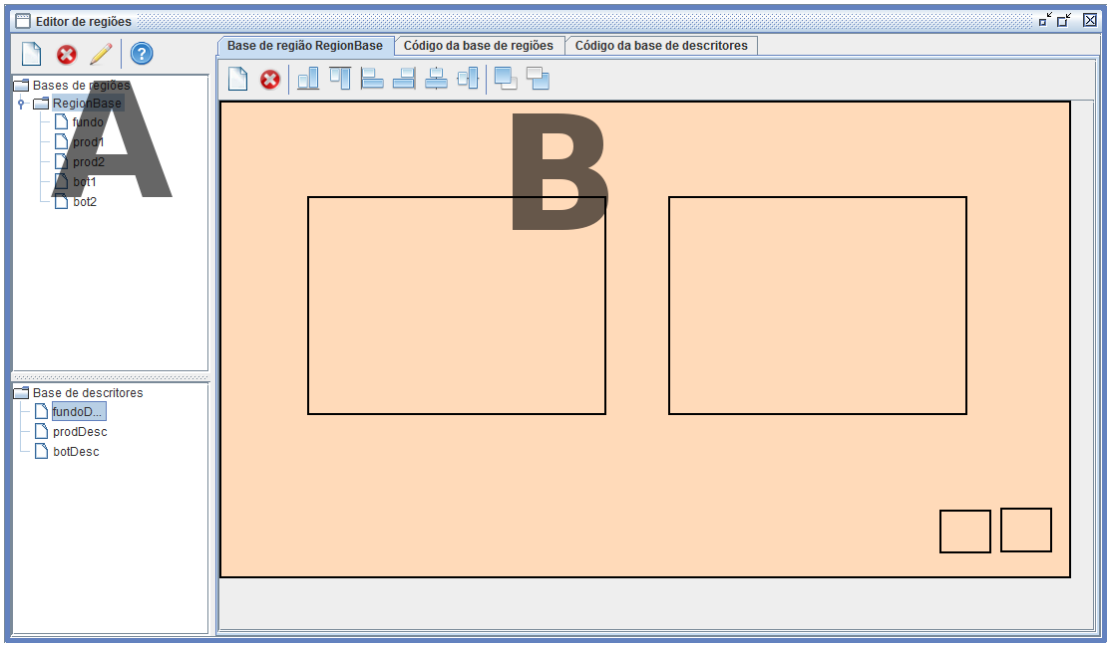


Figura 44: Plugin para criação/edição de regiões e descritores.

Ele oferece uma interface simples, onde as regiões podem ser facilmente manipuladas graficamente utilizando-se apenas o mouse. Além disso, botões auxiliares para alinhamento das mesmas também são oferecidos. Estas características podem ser visualizadas na região B da Figura 44. Esta região é dividida em três abas. A primeira permite a manipulação das regiões criadas em cada uma das bases de região, a segunda e a terceira apresentam o código NCL das bases de regiões e de descritores, respectivamente.

O botão "Novo", na região A, permite a criação de bases de regiões e descritores, os quais são exibidos nas suas respectivas árvores, também na região A. O segundo botão é utilizado para apagar uma base de região, e o terceiro permite editar os atributos da mesma.

Já os botões, localizados em B, permitem criar e remover as regiões que irão compor a base selecionada na árvore. Ao criar uma nova região, o autor deverá escolher um identificador único, *id*, para a mesma, a qual será exibida na região B. Após criada, a região poderá ser movida e redimensionada com uso do mouse. Ao dar um duplo clique sobre a mesma, uma janela com todos os seus atributos, apresentados na Tabela 2 da Seção 3.1.1, é exibida e o autor poderá realizar alterações nos valores dos mesmos. Além disso, também é possível definir uma região pai para a região em edição.

Já para a criação de descritores, é preciso que o autor tenha um conhecimento básico do que esse elemento NCL representa no documento. Para criar os descritores, o autor deverá

preencher alguns campos com os atributos oferecidos pelo elemento. Como nem todos são obrigatórios, o autor deve preencher apenas aqueles atributos que são desejados. Para editar um descritor já criado, basta selecioná-lo na árvore de descritores, que sua janela de edição será exibida. A Figura 45 exibe a janela para a criação/edição de descritores.

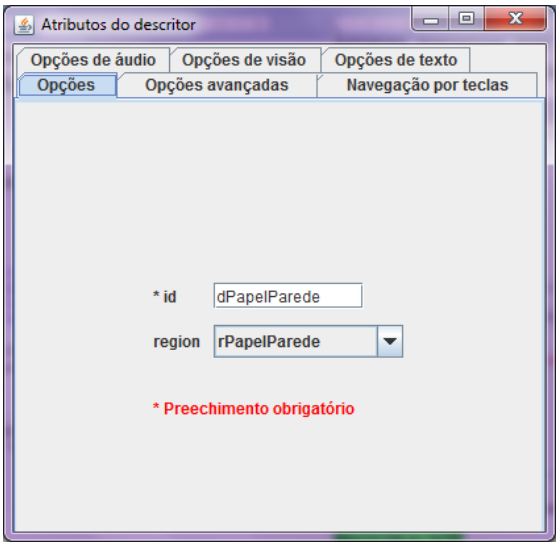


Figura 45: Janela para criação/edição de descritores.

Como pode ser visto na Figura 45, existem 6 abas na janela de edição de um descritor. Estas abas oferecem uma separação entre os diversos atributos e parâmetros que o mesmo oferece, os quais são separados de acordo com o tipo de nó que irá utilizar o descritor. Por exemplo, caso um descritor tenha sido criado para ser associado a um nó de texto, alguns parâmetros relativos a exibição de um texto, como tipo e tamanho de fonte, por exemplo, poderão ser preenchidos pelo autor na janela de criação/edição do descritor. A Figura 46 exibe estes parâmetros.

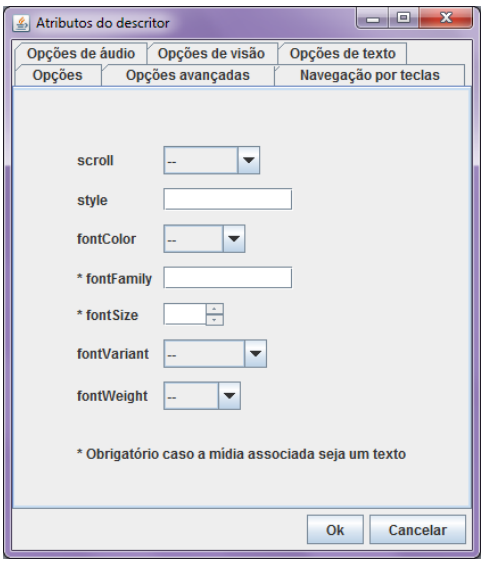


Figura 46: Parâmetros de um descritor para uma mídia do tipo texto.

A linguagem NCL define valores específicos para alguns atributos/parâmetros dos descritores, como *scroll* e *fontColor*, por exemplo. Estes, como pode ser visto na Figura 46, possuem uma caixa de seleção, onde seus possíveis valores podem ser escolhidos pelo autor. Desta forma, garante-se a especificação correta dos valores destes atributos/parâmetros. Ao oferecer caixas numéricas, parâmetro *fontSize* por exemplo, também garante-se a escolha de valores numéricos dentro dos limites estabelecidos pela linguagem.

A Tabela 13 e a Tabela 14 listam os atributos e os parâmetros disponibilizados na janela de criação/edição dos descritores, respectivamente.

Tabela 13: Atributos dos descritores.

Aba	Atributos
Opções	*id,region*
Opções avançadas	*explicitDur, freeze, player*
Navegação por teclas	*focusIndex, moveLeft, moveRight, moveUp, moveDown, focusBorderColor, focusBorderTransparency, focusBorderWidth, focusSrc, focusSelSrc, selBorderColor*

Tabela 14: Parâmetros dos descritores.

Aba	Parâmetros
Opções avançadas	*reusePlayer, playerLife*
Opções de áudio	*soundLevel, balanceLevel, trebleLevel, bassLevel*
Opções de visão	*top, left, bottom, right, width, height, location, size, bounds, zIndex, background, visible, transparency, fit*
Opções de texto	*scroll, style, fontColor, fontFamily, fontSize, fontVariant, fontWeight*

O plugin da visão de leiaute foi desenvolvido de forma que também seja possível sua utilização independentemente do NEXT. Ou seja, ele também pode ser utilizado como uma aplicação *stand-alone*, onde o usuário pode criar um documento NCL que contenha apenas uma ou mais bases de regiões e uma base de descritores. Este documento poderá ser utilizado posteriormente por outros documentos NCL, os quais poderão importar suas bases. Também é possível apenas abrir um documento NCL já criado e alterar suas regiões e descritores.

Maiores detalhes sobre a implementação deste plugin podem ser encontrados no Anexo E.

5.2.1 TESTE DE USABILIDADE

O teste de usabilidade que avaliou o uso do plugin de visão de leiaute foi o mesmo teste realizado para avaliar o NEXT, apresentado no Capítulo 4, Seção 4.4, ou seja com um grupo de 15 usuários composto por alunos de graduação e de doutorado em computação.

Como os alunos tiveram que criar uma aplicação completa, eles precisaram criar regiões e descritores através do uso do plugin de visão de leiaute.

A Tabela 15 apresenta as respostas das perguntas relacionadas ao plugin de visão de leiaute. Os resultados foram separados de acordo com o conhecimento do aluno sobre a linguagem NCL. Assim sendo, os alunos que marcaram a opção 2 ou menor na questão sobre conhecimento da linguagem (9 alunos), estão representados na coluna 'Baixo' (baixo conhecimento de NCL) e os outros (6 alunos), na coluna 'Bom' (bom conhecimento de NCL). A tabela apresenta a média das respostas por grupo de usuários e o desvio padrão.

Tabela 15: Teste de usabilidade do plugin de visão de leiaute no NEXT.

Pergunta	**Média / Desvio padrão**		**Conclusão do resultado**	
	Baixo	Bom	Baixo	Bom
Com relação à facilidade para compreender como usar o plugin de visão de leiaute, que nota você daria à mesma?	4 / 0,7	4 / 1,3	Fácil	Fácil
Com relação à criação de regiões, como você classifica a usabilidade do plugin?	4 / 0,7	4 / 0,4	Fácil	Fácil
Com relação à criação de descritores, como você classifica a usabilidade do plugin?	4 / 1	4 / 0,8	Fácil	Fácil
Como você classifica a janela de criação/edição de descritores?	4 / 0,5	4 / 1,4	Fácil	Fácil

Seguem algumas observações e sugestões indicadas pelos alunos sobre o plugin de visão de leiaute: a exibição de todas as regiões criadas fica confusa quando há muitas regiões, seria interessante que o autor pudesse escolher quais regiões ficam visíveis; na primeira vez em que se aumenta alguma região arrastando um dos lados do retângulo, o lado oposto também aumenta; colocar uma opção de zoom nas regiões.

5.2.2 LIMITAÇÕES DO PLUGIN

Com relação à linguagem NCL, todas as funcionalidades relativas aos elementos <regionBase>, <region> e <descriptor> são atendidas pelo plugin. Já em relação ao elemento <descriptorBase>, apenas a criação de elementos <descriptor> é possível, esta versão do plugin não permite a criação de elementos <descriptorSwitch>, os quais associam a escolha de um descritor de acordo com uma das regra da base de regras.

Uma nova versão do plugin pode ser desenvolvida a fim de permitir a criação do elemento <descriptorSwitch> e atender às sugestões dos alunos que realizaram o teste de usabilidade.

5.3 EDITOR DE CONECTORES

Grande parte do esforço de desenvolvimento de uma aplicação multimídia é dedicada à especificação de relacionamentos espaço-temporais, incluindo a possibilidade de interatividade do usuário. Esse esforço é ainda maior quando a linguagem de autoria declarativa requer a especificação desses relacionamentos através de elos e conectores, como é o caso da linguagem NCL.

Em um programa NCL, o autor deve especificar cada tipo de relação a ser usada como um conector hipermídia causal [Muchaluat-Saade e Soares 2002], e cada relacionamento entre os componentes de mídia como um elo que utiliza um dos conectores especificados, como explicado no Capítulo 3.

Sendo assim, o sincronismo em um documento NCL é expresso através de elos, que representam relacionamentos causais entre os componentes da aplicação, utilizando conectores hipermídia, os quais definem os papéis dos participantes da relação.

No entanto, a especificação destes conectores causais pode ser bastante complexa, principalmente quando usamos condições e ações compostas e precisamos definir vários atributos para seus papéis. O conector da Figura 47, retirado de [Soares e Barbosa 2009], é um exemplo. Por isso, com o objetivo de facilitar a criação e/ou edição de conectores, evitando que o autor despenda muito tempo em sua elaboração, um plugin para criação/edição gráfica dos mesmos foi desenvolvido em [Silva e Muchaluat-Saade 2011].

```
<causalConnector id='onEndAttrSetTrueResume'>
    <compoundCondition operator='and'>
        <simpleCondition role='onEndAttribution'/>
        <assessmentStatement comparator='eq'>
            <attributeAssessment role='testIfFalse'
                eventType='attribution'
                attributeType='nodeProperty'/>
            <valueAssessment value='false'/>
        </assessmentStatement>
    </compoundCondition>
    <compoundAction operator='seq'>
        <simpleAction role='resume'/>
        <simpleAction role='setAsTrue' value='true'
            eventType='attribution' actionType='start'/>
    </compoundAction>
</causalConnector>
```

Figura 47: Código NCL de um conector mais elaborado.

O plugin de conectores desenvolvido para o NEXT apresenta uma interface gráfica simples e exibe mensagens para orientar o usuário durante a criação do conector. A Figura 48 mostra a interface principal do plugin exibindo o conector apresentado na Figura 47.

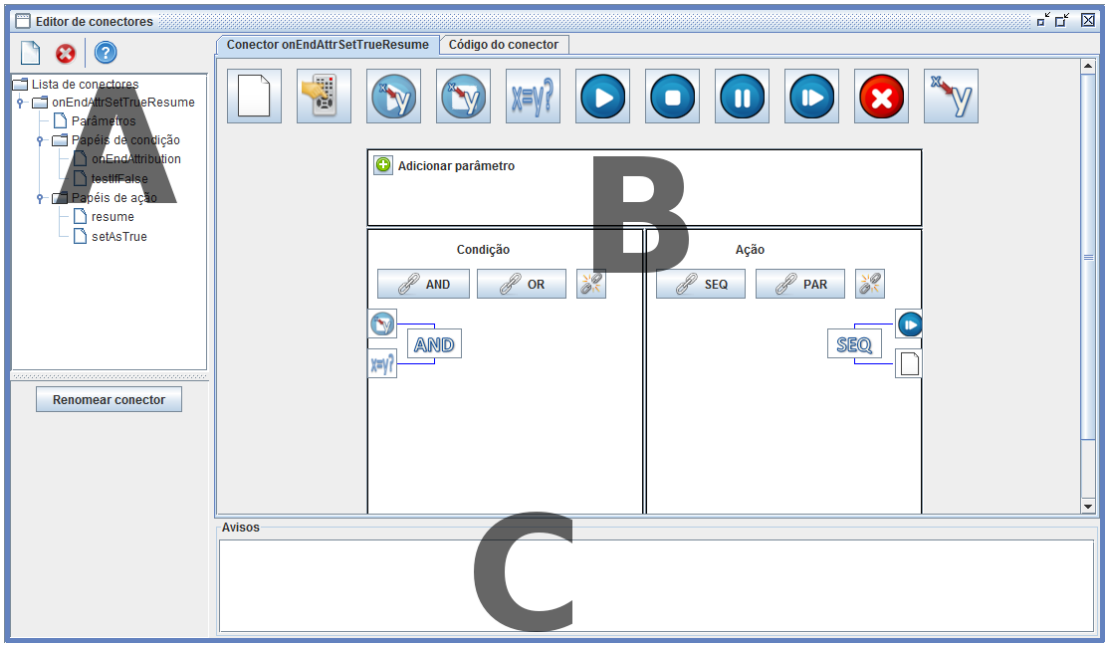


Figura 48: Plugin gráfico para criação de conectores.

Na Figura 48, no topo da região A, são exibidos três botões. O primeiro é utilizado para criar um novo conector, o segundo para remover o conector que estiver selecionado na árvore, e o terceiro exibe um menu de ajuda que explica o funcionamento do plugin. Abaixo destes botões, tem-se uma árvore que é responsável por exibir uma lista com todos os conectores encontrados na base de conectores do documento, bem como seus papéis de condição e ação e seus parâmetros. Ao selecionar um conector qualquer da árvore, o mesmo é exibido em B. A região B, primeira aba, é responsável por exibir, graficamente, o conector que está sendo criado/editado ou que foi selecionado na árvore. Já a segunda aba, é responsável por exibir o código NCL do conector selecionado. Isto facilita a compreensão do código NCL pelo autor.

Na parte superior são apresentados onze ícones, os quais representam diferentes tipos de papéis simples. O primeiro permite que o usuário crie um papel e defina todos os seus parâmetros. Com exceção do quinto ícone, que representa um papel de *assessmentStatement*, os outros ícones representam os papéis predefinidos pela linguagem NCL. Ao utilizar um dos papéis predefinidos, o usuário não precisa especificar os atributos do mesmo, pois os valores default do papel serão utilizados. Esta possibilidade visa acelerar a criação dos papéis. A Tabela 16 mostra os ícones que são utilizados para criar estes papéis.

Os ícones de papéis servem tanto para criar papéis de condição como de ação, bastando que o papel desejado seja arrastado até o desenho do conector, logo abaixo, no espaço destinado a papéis de condição (lado esquerdo) ou de ação (lado direito). Ao soltar o ícone arrastado no conector, será aberta uma janela exibindo os atributos correspondentes ao papel, os quais podem ser editados pelo usuário.

Tabela 16: Ícones dos papéis de ação e de condição.

Ícone	Papel de condição	Papel de ação	Significado
	definido pelo usuário	definido pelo usuário	Permite criar um novo papel
	onSelection	-	Seleção do usuário (interatividade)
	onBeginAttribution	-	No início da atribuição
	onEndAttribution	-	No fim da atribuição
	definido pelo usuário	-	Avalia o estado de um evento ou o valor de um atributo
	onBegin	start	Início/Iniciar apresentação
	onEnd	stop	Fim/Finalizar apresentação
	onPause	pause	Pausa/Pausar apresentação
	onResume	resume	Retomar apresentação
	onAbort	abort	Abortar apresentação
	-	set	Atribuir novo valor a um atributo

Como os papéis possuem diversos atributos, os mesmos são apresentados em diferentes abas nas janelas de criação/edição dos mesmos. Desta forma, o ambiente apresentado ao autor fica mais claro e organizado. Na Figura 49, a janela de criação de um papel de ação é apresentada, enquanto que a Figura 50 apresenta a janela de criação de um papel de condição.

Como pode ser visualizado na Figura 49, as opções de utilizar um parâmetro no papel *start* estão desabilitadas. Isto ocorre porque o usuário não criou nenhum parâmetro no conector. A partir do momento em que o usuário criar algum, as opções estarão habilitadas e uma lista com todos os parâmetros que foram criados será exibida para que o usuário selecione o parâmetro desejado. O mesmo ocorre para o papel de condição.

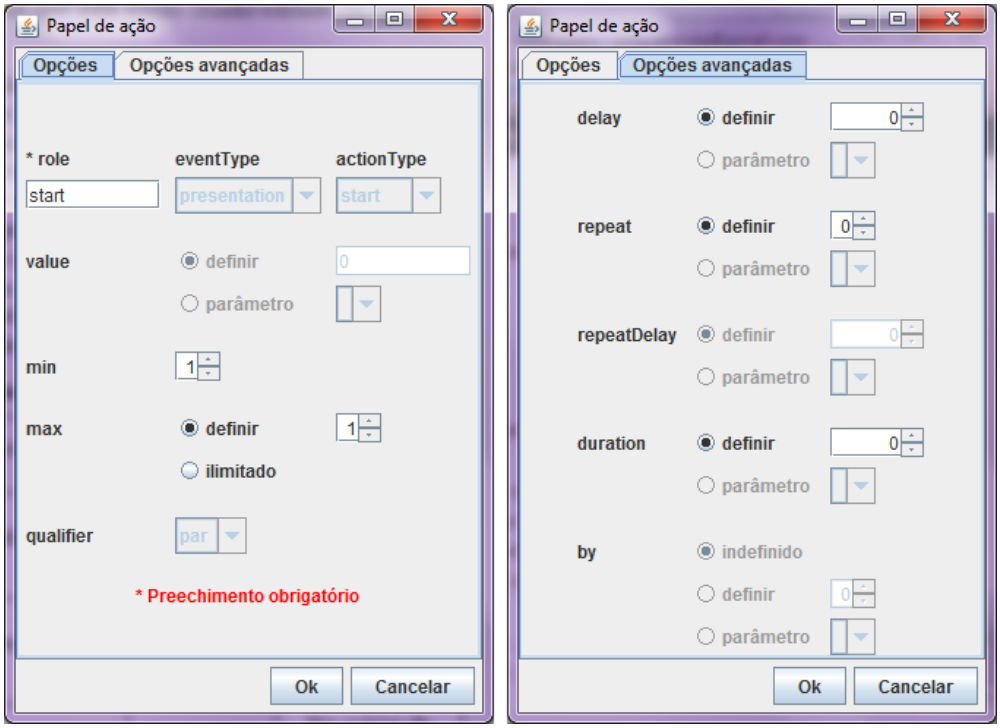


Figura 49: Janela de criação de um papel de ação.

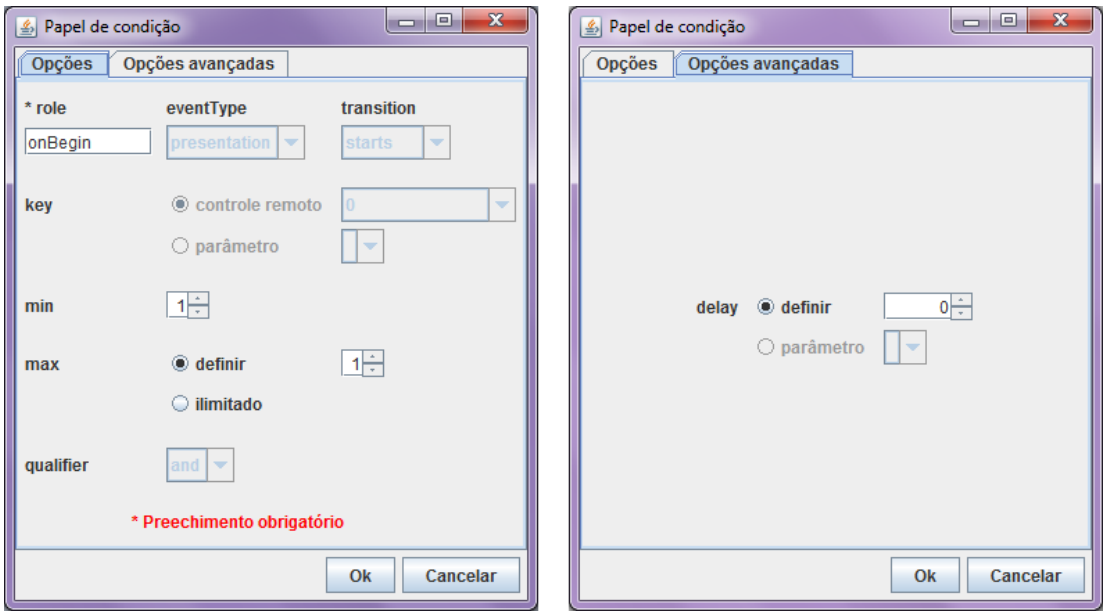


Figura 50: Janela de criação de um papel de condição.

Os papéis de condição do tipo *assessmentStatement* são criados da mesma forma que os outros. Sua diferença está nos atributos apresentados na janela de criação/edição, já que estes não são iguais aos dos outros papéis. A janela de criação/edição pode ser visualizada na Figura 51.

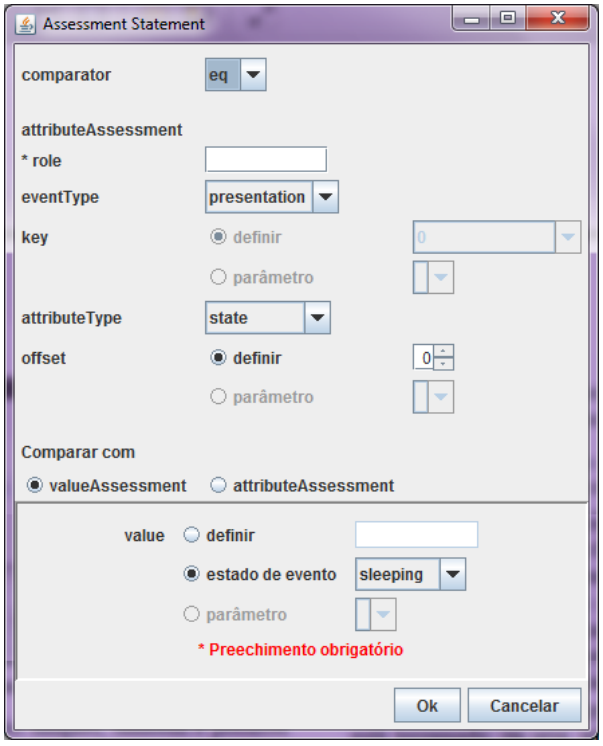


Figura 51: Janela de criação do papel de condição *assessmentStatement*.

Além de permitir a criação de papéis simples, também é possível criar papéis compostos. Cada um dos lados do conector possui ícones para criação de condições e ações compostas. Depois de adicionar os papéis simples, basta que o usuário selecione os papéis desejados e clique no operador adequado. Para criação de condições compostas, utilizam-se os ícones *AND* ou *OR* e, para criação de ações compostas, *SEQ* ou *PAR*. Para remover algum papel de uma condição ou ação composta, basta selecionar o papel e utilizar o botão "Desagrupar".

Os ícones utilizados para a criação de condições e ações compostas, assim como o botão "Desagrupar", podem ser visualizados na Tabela 17.

Tabela 17: Ícones utilizados para criação/edição de condições e ações compostas.

Ícone	Operador	Significado
AND	AND	Todas as condições devem ser satisfeitas
OR	OR	Pelo menos uma das condições deve ser satisfeita
SEQ	SEQ	Ações devem ser executadas em ordem explícita
PAR	PAR	Ações devem ser executadas em ordem aleatória
	-	Desagrupar papel de condição ou ação composta

Um conector NCL pode possuir parâmetros, como explicado no Capítulo 3. Por este motivo, na parte superior do desenho do conector (Região B na Figura 48), existe um botão que permite a criação dos mesmos. Ao adicionar um parâmetro, uma janela será exibida pedindo o nome do parâmetro. Depois de completar esta tela, o parâmetro criado será exibido juntamente com um botão para removê-lo, caso o usuário não queira mais utilizá-lo. Para evitar possíveis erros, só é possível remover um parâmetro que não esteja sendo utilizado em nenhum papel. Caso o usuário tente remover algum que esteja sendo utilizado, uma mensagem informará quais papéis o utilizam.

Ao clicar duas vezes em um papel já criado, uma janela contendo seus atributos será exibida e o usuário pode realizar modificações no mesmo ou, até mesmo, removê-lo. O mesmo ocorre para as condições/ações compostas.

A parte inferior da Figura 48, região C, é destinada a exibir mensagens de ajuda para auxiliar na criação do conector. Enquanto esta região apresentar alguma mensagem, significa que o conector ainda não está terminado, ou seja, não está pronto para ser utilizado em um documento NCL. Além disto, esta região também informa ao autor sobre possíveis problemas no conector criado como, por exemplo, quando um conector com uma condição composta utiliza o operador *AND* com duas condições que representam transições em máquinas de estados (condição *trigger*). Esta condição composta jamais seria verdadeira. Outro exemplo é quando um conector possui apenas uma única condição simples formada por um *assessmentStatement,* que nunca causará o disparo da ação, pois não é uma condição *trigger* [Soares e Rodrigues 2005].

Maiores detalhes sobre a implementação deste plugin podem ser encontrados no Anexo D.

5.3.1 TESTE DE USABILIDADE

O teste de usabilidade do editor de conectores [Silva e Muchaluat-Saade 2011] foi realizado com um grupo de 17 alunos, todos já com algum conhecimento da linguagem NCL, alguns mais experientes, outros menos. O perfil destes alunos para o teste era que estes possuíssem conhecimento da linguagem NCL, uma vez que esse conhecimento é necessário para que o aluno consiga entender a utilidade do plugin.

O teste consistiu em uma lista de conectores que os alunos deveriam criar utilizando o plugin como um editor stand alone. Ao final, eles responderam um formulário, onde opiniões sobre os ícones utilizados na interface para a representação dos papéis e sugestões foram solicitadas.

A lista de conectores teve como objetivo estimular os alunos a explorar a ferramenta, utilizando diversos tipos de papéis e seus atributos. Totalizando oito conectores, estes contemplavam a criação de condições e ações simples e compostas, fazendo uso de papéis de condição "*onSelection*" e "*assessmentStatement*" e do papel de ação "*set*", entre outros. Alguns conectores exigiam, inclusive, a definição de parâmetros e a criação de papéis não predefinidos pela linguagem.

O formulário era composto por perguntas que variavam desde o grau de conhecimento da linguagem NCL à importância da ferramenta proposta. As questões eram objetivas e as opções de resposta variavam numa escala de seis opções (de zero a cinco, onde zero era a pior opção e cinco a melhor). A Tabela 18 apresenta o resultado obtido.

Tabela 18: Teste de usabilidade do plugin para criação de conectores como um editor stand alone.

Pergunta	Média / Desvio padrão	Conclusão do resultado
Como você classifica o seu conhecimento em NCL?	3 / 1,3	Bom
Como você classifica o seu conhecimento em conectores NCL?	3 / 0,9	Bom
Você já criou algum aplicativo utilizando a linguagem NCL?	-	Sim
Com que frequência você cria os conectores que utiliza nas suas aplicações?	3 / 1,9	De vez em quando
Com relação ao tempo para compreender a ferramenta, que nota você daria à mesma?	4 / 0,8	Rápido
Com relação ao grau de usabilidade (facilidade de uso), que nota você daria à ferramenta?	4 / 0,9	Muito bom
Que nota você daria para a importância (necessidade) da ferramenta?	4 / 0,9	Muito útil
Com relação ao grau de desempenho (tempo de processamento, falhas do programa), que nota você daria à ferramenta?	4 / 1,1	Muito bom
Com relação aos ícones utilizados para representar os papéis, que nota você daria em relação à clareza dos mesmos? Alguma sugestão?	3 / 0,9	Claros
Você conseguiu criar os conectores da lista?	-	Sim, todos

Analisando-se as respostas, conclui-se que a maioria dos alunos levou pouco tempo para compreender sobre como utilizar a ferramenta e que os graus de usabilidade e de desempenho (falhas da ferramenta, tempo de processamento) foram muito bons. A maioria

dos alunos considerou os ícones utilizados para representar os papéis como “claros”. E, apesar de a maioria dos alunos nem sempre criar os conectores que utilizam em seus documentos NCL, estes consideraram a ferramenta muito útil.

A seguir, estão algumas sugestões indicadas pelos usuários, que poderão ser analisadas e contempladas numa nova versão da ferramenta: utilizar, além dos ícones, um menu para a criação dos papéis, clicando-se com o botão direito do mouse sobre o conector; fornecer uma forma alternativa para remover um papel. Atualmente, essa operação só é possível clicando-se no mesmo e escolhendo a opção “Delete”; melhorar a criação do papel de condição do tipo “*assessmentStatement*”, pois os atributos do mesmo ficaram confusos.

5.3.2 LIMITAÇÕES DO PLUGIN

Com relação à linguagem NCL, todas as funcionalidades relativas aos elementos <connectorBase> e <connector> são atendidas pelo plugin. Inclusive, ele permite a criação de documentos NCL com apenas uma base de conectores, que pode ser referenciado por outros documentos NCL. Além disso, uma nova versão do plugin está sendo desenvolvida, levando-se em consideração as sugestões dos alunos que realizaram o teste de usabilidade.

5.4 VISÃO ESTRUTURAL

Em NCL, contextos são responsáveis por agrupar diversos elementos, inclusive outros contextos, permitindo uma melhor organização lógica do documento, como mencionado no Capítulo 3. Eles agrupam objetos de mídia e outros contextos da aplicação e seus elos determinam relacionamentos entre os mesmos.

No fim do Capítulo 3, a Figura 28 mostrou a visão estrutural de um documento NCL enquanto que a Figura 29 exibiu o código NCL da mesma. Através dessas figuras, é possível perceber que a compreensão do documento através de sua visão estrutural é muito mais intuitiva do que através de seu código, assim como alterações no documento também são mais intuitivas de serem realizadas graficamente.

Então, com o objetivo de facilitar a definição de seus elementos e, também, do elemento <body> de um documento NCL, um plugin gráfico para a criação e edição de contextos foi criado. Ele oferece uma visualização estrutural simples dos elementos que compõem o corpo do documento ou um de seus contextos, e sua interface gráfica é baseada em *drag-and-drop* com o objetivo de acelerar o desenvolvimento do documento.

Apesar de facilitar o desenvolvimento, este plugin requer um conhecimento básico dos elementos NCL que compõem o corpo do documento, pois alguns elementos, como as âncoras ou switches, podem não ser tão intuitivos para um autor que não conhece NCL.

A Figura 52 apresenta a interface gráfica que é oferecida pelo plugin de visão estrutural. Ela é divida em duas partes: região A, que exibe uma árvore com todos os componentes do corpo do documento; e região B, que é dividida em duas abas. A primeira aba é responsável pela visão estrutural do corpo do documento, e a segunda exibe o código NCL do mesmo.

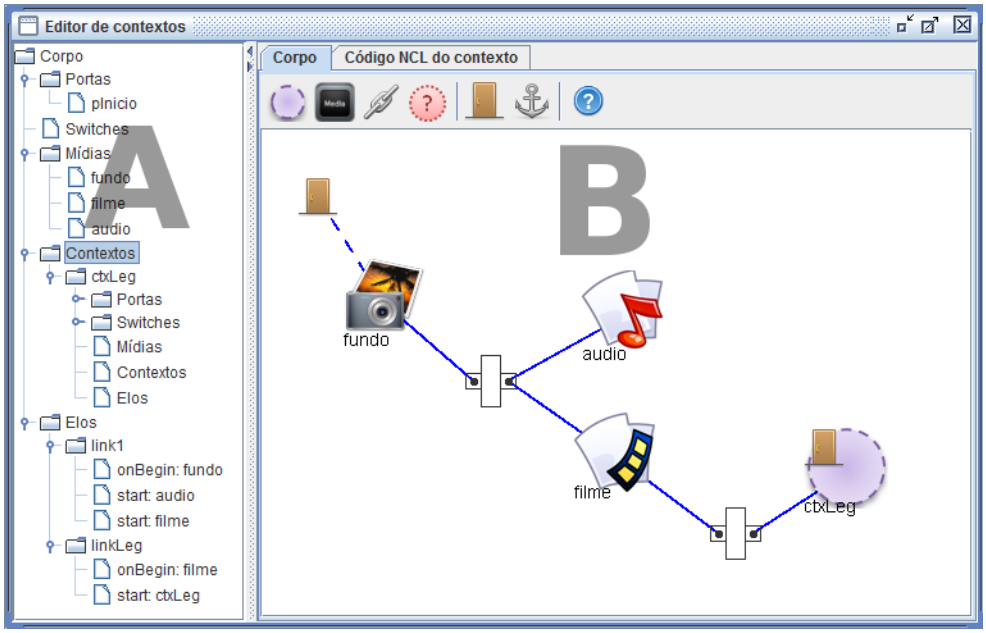


Figura 52: Plugin de visão estrutural de contextos.

A Tabela 19 apresenta a funcionalidade dos botões utilizados para criar elementos nos contextos.

Tabela 19:Ícones do plugin de visão estrutural.

Ícone	Funcionalidade
	Criar contexto (<context>)
	Criar mídia (<media>)
	Criar elo (<link>)
	Criar switch (<switch>)
	Criar porta no contexto (<port>)
	Criar âncora em uma mídia (<area> ou <property>)

O plugin permite que o autor crie elementos <media> através do botão “Nova mídia” ou, então, através do repositório de mídias do NEXT. A mídia pode ser arrastada do repositório e liberada sobre o espaço destinado aos componentes do corpo. Ao ser liberada, uma nova mídia será criada e o seu *id* será igual ao nome da mídia arrastada, além disso, seu atributo *src* também será definido de acordo com a localização da mídia.

Caso o plugin não consiga criar a mídia por algum problema com o *id*, por exemplo, já existe alguma mídia com *id* igual ao nome da mídia, a janela para criação de mídia será exibida e o autor deverá escolher um novo *id*. Com as mídias do repositório, também é possível alterar uma mídia já criada no documento, bastando soltá-la sobre a mídia que se deseja alterar. Neste caso, o atributo *src* da mídia será definido como o da mídia arrastada.

A representação gráfica do elemento <media> varia de acordo com o tipo da mídia. A Figura 52 apresenta três tipos de mídia: tipo imagem, representada por uma máquina fotográfica, tipo áudio, com uma nota musical, e tipo vídeo, com um pedaço de rolo de vídeo. Esses ícones são utilizados quando o autor define o tipo de mídia, ou ao indicar qual será a mídia utilizada no elemento, através do atributo *src*, ou ainda, através do seu atributo *type*, caso esteja definido. Caso estes não tenham sido definidos pelo autor ou a mídia selecionada não esteja definida pelo padrão NCL, sua representação será igual ao ícone de criação de mídias.

Após ter criado uma mídia, o autor poderá criar âncoras na mesma. Isto poderá ser feito de duas formas: com um duplo clique sobre a mídia desejada, sua janela de edição é exibida e, na aba ‘Âncoras’ da janela, é possível criar âncoras na mesma; ou através do botão para criação de âncoras, basta arrastá-lo e soltá-lo sobre a mídia. Os dois tipos de âncora, <property> e <area>, podem ser criados e suas janelas de edição podem ser visualizadas na Figura 53.

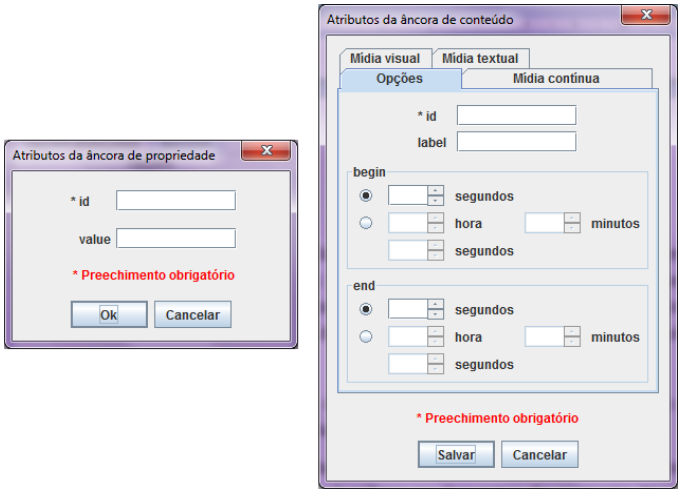


Figura 53: Janelas para criação de âncoras nos elementos <media>.

Uma âncora de propriedade apresenta apenas dois atributos: *id*, obrigatório, e *value*, opcional. Já uma âncora de conteúdo pode ter diversos atributos, os quais variam de acordo com o tipo de mídia e são disponibilizados em diferentes abas da janela de edição. Estes atributos podem ser visualizados na Tabela 20.

Tabela 20: Atributos do elemento <area>.

Aba	Atributos
Opções	*id, label, begin, end*
Mídia contínua	*first, last*
Mídia visual	*coords*
Mídia textual	*text, position*

Os elementos <port> de um contexto representam seus pontos de interface e servem para referenciar seus componentes. Cada porta pode fazer referência a um componente e elas são responsáveis por indicar quais componentes irão iniciar a apresentação do documento, no caso do contexto ser o corpo. No caso de portas de um outro contexto qualquer, elas servem para permitir que algum elemento externo ao contexto possa referenciar algum de seus elementos. Estas podem ser criadas através do botão para criação de portas.

A Figura 54 apresenta a janela de edição de uma porta. Ela exibe todos os elementos que podem ser referenciados, através do atributo *component* e, caso o elemento escolhido pelo autor possua pontos de interface, as mesmas são disponibilizadas na caixa de seleção do campo ‘interface’.

Figura 54: Janela de edição do elemento <port>.

Os elos, representados pelo elemento <link>, são criados para definir os relacionamentos entre as mídias e fazem referência a um dos conectores do documento, definido em uma base de conectores. Apesar do plugin de visão estrutural não permitir a criação de conectores, ele é informado pelo NEXT sobre a existência destes, uma vez que a

lista de interesses do plugin contempla este elemento. Assim sendo, o plugin só permite a criação de elos, através do botão para criação de elos, caso exista pelo menos um conector no documento.

Ao criar um elo, o autor poderá escolher um *id* para o mesmo, opcional, e deverá escolher um dos conectores existentes. Após criado, é possível adicionar elementos <bind> no mesmo, os quais devem ser relacionados aos componentes já inseridos no contexto.

Para editar, criar parâmetros, ou até mesmo remover um elo, basta um clique duplo sobre o mesmo que sua janela de edição será exibida. A criação de parâmetros está relacionada à existência de elementos <connectorParam> no conector, ou seja, o conector referenciado precisa possuir parâmetros para que seja possível criar parâmetros no elo. Sendo assim, o plugin verifica se é possível ou não criar estes parâmetros no elo.

A Figura 55 apresenta a visualização gráfica de um elo, envolvido de vermelho, com um *bind*, traço azul, o qual relaciona um papel de condição a um elemento <media>. E o elo está pronto para a criação de um outro *bind*, que irá relacionar um papel de ação a alguma mídia do documento.

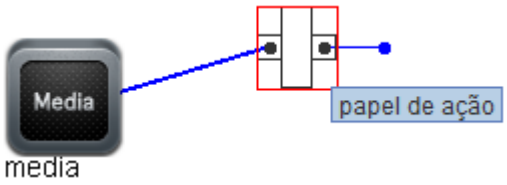


Figura 55: Visualização gráfica dos elementos <link> e <bind>.

Para criar elementos <bind>, basta clicar sobre os pontos de interface do conector utilizado pelo elo (representados por bolinhas). A bolinha à esquerda permite a criação de *binds* relacionados a papéis de condição, enquanto à da direita cria *binds* relacionados a papéis de ação. Após clicar na bolinha, um traço azul é exibido e o mesmo deverá ser arrastado a qualquer um dos componentes já criados no contexto. Ao soltar o traço sobre um dos componentes, este será referenciado pelo *bind* que acabou de ser criado. Caso o mesmo possua interfaces, uma janela é exibida e o autor tem a opção de fazer referência a uma das interfaces do componente, caso desejado.

Um clique duplo sobre o *bind* exibe sua janela de edição, onde é possível alterar o papel relacionado ao mesmo, assim como a interface do componente, caso este possua interfaces. E, assim como no elo, também é possível criar parâmetros.

A quantidade de *binds* é definida pelo conector referenciado pelo elo e este valor é sempre verificado quando o autor tenta criar um novo elemento. Desta forma, garante-se que o elo não possua mais *binds* do que o permitido, garantindo-se a consistência do documento.

Switches são elementos que permitem adaptar o conteúdo de uma aplicação e, para isso, eles definem componentes, os quais são relacionados às regras definidas na base de regras, como explicado no Capítulo 3. Como um switch é composto por outros elementos, o plugin fornece duas formas de representação para o mesmo: uma que representa o switch e outra que representa seus elementos.

O switch é representado por um círculo tracejado com um ponto de interrogação no interior. Esta representação indica que existe um elemento <switch> no documento. Para visualizar e editar os componentes definidos no switch, basta um duplo clique no mesmo ou selecioná-lo na árvore. Com um duplo clique, sua janela de edição também é exibida. A Figura 56 apresenta a visualização do interior de um switch e de sua janela de edição.

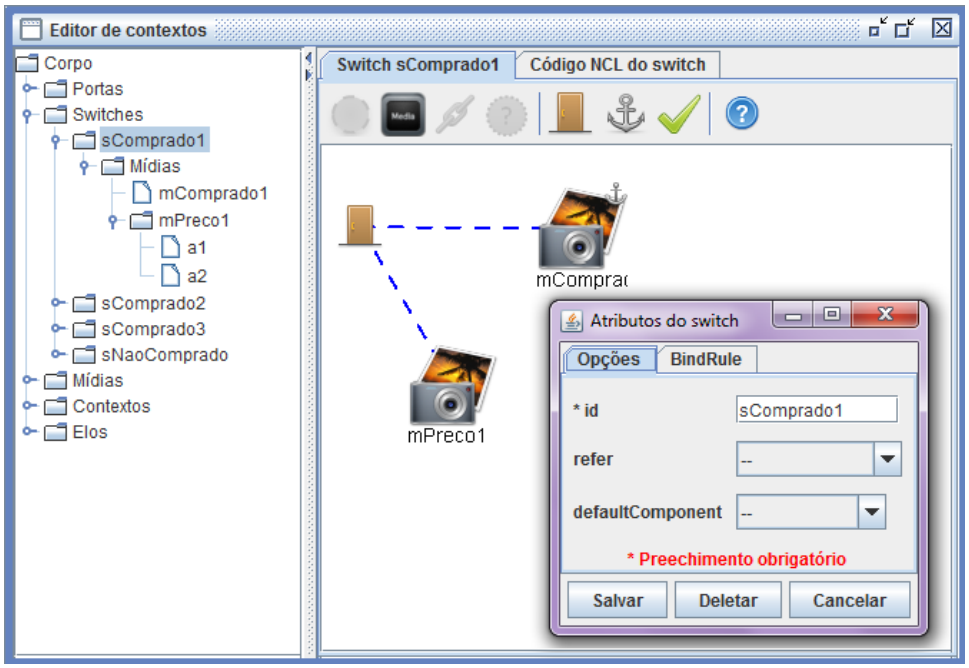


Figura 56: Visualização do elemento <switch> e sua janela de edição.

A janela de edição permite a alteração dos atributos do switch, na aba ‘Opções’, e permite a criação dos elementos <bindRule>, na aba ‘BindRule’. Para criar o bindRule, as regras definidas no documento são apresentadas, assim como os componentes definidos no switch, bastando que o autor selecione uma das regras e um dos componentes para criar o elemento. Além disso, a aba ‘BindRule’ também permite a visualização da base de regras, onde é possível criar ou remover regras.

Pode-se ver na Figura 56 que um novo ícone para criação de elementos, ao lado do ícone para criação de âncoras, aparece quando o conteúdo de um switch é exibido. Este ícone é apenas uma forma alternativa para a criação de um <bindRule>. Arrastando-o e liberando-o sobre algum componente do switch, uma janela é exibida para que o autor escolha uma das

regras existentes para que a mesma seja, então, associada ao componente através do elemento <bindRule>.

As janelas para criação de <bindRule> e de regras podem ser visualizadas na Figura 57. Para criar um <bindRule> o autor deve escolher uma mídia e uma regra. No caso de utilização do ícone, a mídia escolhida é aquela onde o ícone foi liberado. Já para criar uma regra, o autor deve escolher o *id* da mesma, a interface da mídia do tipo 'application/x-ncl-settings', a qual pode ser uma já existente ou uma nova que será criada, o comparador e o valor a serem utilizados no elemento. Apesar de ser possível a definição de regras compostas em NCL, esta versão do plugin possibilita apenas a criação de regras simples.

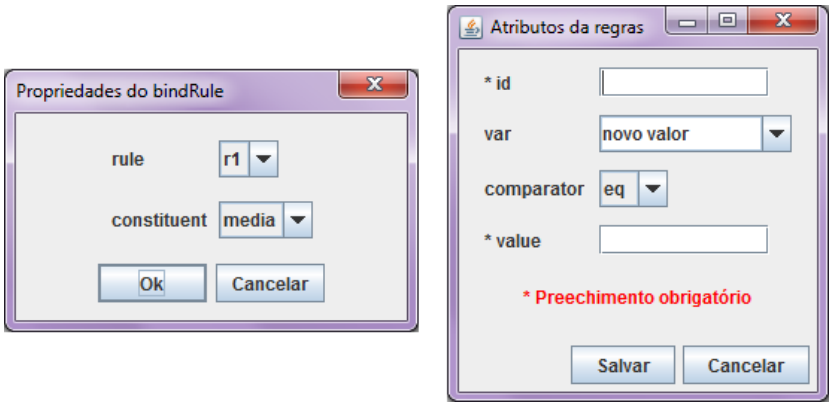


Figura 57: Janelas para criação dos elementos <bindRule> e <rule>.

As interfaces dos componentes de um switch não podem ser diretamente referenciadas por algum elo do corpo do documento, a não ser que estas possuam portas relacionando-as. O elemento <switchPort> é responsável por esta relação. Diferentemente do elemento <port> de um contexto, que só pode fazer referência a um componente, a porta de um switch pode fazer referência a diversos componentes de um switch através do elemento <mapping>. A Figura 58 exibe a janela de edição do <switchPort> apresentado na Figura 56.

Figura 58: Janela de edição do elemento <switchPort>.

Para mapear os componentes do switch, basta clicar em 'Adicionar mapeamento'. Uma nova janela será aberta e apresentará todos os componentes e suas respectivas interfaces

para que o autor faça o mapeamento. Na Figura 58 a opção ‘Adicionar mapeamento’ está desabilitada, pois todos os componentes do switch já foram mapeados no <switchPort>. O primeiro elemento <mapping> mapeia a mídia ‘mPreco1’, na sua interface ‘a1’, enquanto o segundo mapeia a mídia ‘mComprado1’.

Pode-se perceber pela Figura 56, que a segunda aba agora apresenta o código NCL do switch e não mais do corpo do documento. Percebe-se também que, apesar do elemento <switch> em NCL também permitir a definição de contextos e switches, os botões para criação destes elementos não estão habilitados nesta versão do plugin. Sendo assim, os componentes de um switch criado com este plugin serão todos objetos de mídia.

Além de switches, elementos do tipo <context> podem ser criados. Um contexto é representado por um círculo roxo tracejado na visualização estrutural de seu contexto pai, considerando a hierarquia de contextos do documento. Um duplo clique sobre ele ou sua seleção na árvore exibe sua visão estrutural, representando graficamente seus componentes. A Figura 59 exibe esta representação e sua janela de edição. O contexto ‘ctx1’ possui duas mídias, uma porta e um elo e seu código pode ser visualizado na segunda aba da janela.

Figura 59: Visualização do elemento <context> e sua janela de edição.

O objetivo principal da exibição de uma árvore com os componentes do corpo (região A da Figura 52) é ajudar o autor a localizar os elementos do contexto. Na região B, os elementos NCL podem ser livremente movimentados pelo autor, mas à medida que mais elementos são inseridos no contexto, a exibição gráfica destes pode ficar um pouco confusa. Por isso, ao selecionar algum elemento na árvore, o mesmo será destacado em B, facilitando sua localização.

Como o plugin oferece uma visão dos elementos do corpo, sua lista de interesse contempla mídias, portas, elos, contextos e switches. Entretanto, alguns elementos do cabeçalho, como conectores e regras por exemplo, também são de interesse deste plugin, uma vez que elos e switches precisam dessas informações, e, portanto, devem ser incluídos na sua lista de interesse.

Apesar do plugin receber informações de alguns elementos do cabeçalho, o mesmo não permite sua alteração, com exceção da base de regras. Entretanto, modificações neles implicam, em alguns casos, alterações nos elementos do corpo. As devidas alterações são realizadas, mantendo-se a consistência do documento.

Maiores detalhes sobre a implementação deste plugin podem ser encontrados no Anexo F.

5.4.1 TESTE DE USABILIDADE

O teste de usabilidade que avaliou o uso do plugin de visão estrutural foi o mesmo teste realizado para avaliar o NEXT, apresentado no Capítulo 4, Seção 4.4, ou seja, com um grupo de 15 usuários composto por alunos de graduação e de doutorado em computação. Como os alunos tiveram que criar uma aplicação completa, eles precisaram criar os elementos do corpo do documento através do uso do plugin de visão estrutural.

A aplicação teve como objetivo explorar a criação de todos os elementos que o corpo de um documento NCL pode ter a fim de verificar a usabilidade do plugin na criação destes, assim como a criação de regras para uso em um switch. Foi pedido, ainda, que os alunos criassem contextos para cada conjunto produto/preço a fim de validar a criação destes e estruturar o documento NCL.

A Tabela 21 apresenta as perguntas relacionadas ao plugin de visão estrutural. Os resultados foram separados de acordo com o conhecimento do aluno sobre a linguagem NCL. Assim sendo, os alunos que marcaram a opção 2 ou menor na questão sobre conhecimento da linguagem (9 alunos), são representados na coluna ‘Baixo’ (baixo conhecimento de NCL) e os outros (6 alunos), na coluna ‘Bom’ (bom conhecimento de NCL). A tabela apresenta a média para cada resposta, assim como o desvio padrão.

Tabela 21: Teste de usabilidade do plugin de visão estrutural no NEXT.

Pergunta	Média / Desvio padrão		Conclusão do resultado	
	Baixo	Bom	Baixo	Bom
Com relação à facilidade para compreender como usar o plugin de visão estrutural, que nota você daria à mesma?	4 / 0,7	4 / 0,8	Fácil	Fácil
Com que frequência você precisou utilizar a opção de ajuda do plugin de visão estrutural?	2 / 1,8	2 / 1,6	Pouco	Pouco
Como você classifica as imagens utilizadas para representar os elementos do contexto?	4 / 0,7	4 / 0,9	Muito Bom	Muito Bom
Como você classifica a forma de exibição dos elementos do contexto?	4 / 0,7	4 / 0,5	Muito Bom	Muito Bom
Com relação à facilidade para criar o elemento <bind> nos elos, que nota você daria à mesma?	4 / 1	4 / 1,2	Fácil	Fácil
Com relação à facilidade para criar regras, que nota você daria à mesma?	3 / 1,2	3 / 0,5	Fácil	Fácil

Seguem algumas observações e sugestões indicadas pelos alunos sobre o plugin estrutural: o plugin reorganiza os elementos do corpo quando o mesmo é fechado e reaberto. Ao ser reaberto, seria bom manter as posições em que os elementos se encontravam quando o mesmo foi fechado; na janela de atributos do elemento porta, o campo “id” some quando se altera o componente “Mídia”; exibir o “id” do elo sobre a figura do mesmo; indicar os papéis nos binds; na janela de criação dos binds, permitir a criação de parâmetros no elo quando o mesmo for criado e não apenas em sua edição.

5.4.2 LIMITAÇÕES DO PLUGIN

Apesar do plugin de visão estrutural ser responsável pela criação de contextos e de seus elementos filhos, ele também permite a construção de regras, viabilizando a utilização de switches no documento sendo criado. A linguagem NCL permite a criação de regras simples e compostas, entretanto, esta versão do plugin permite apenas a criação de regras simples.

Uma outra limitação deste plugin é a não possibilidade de criação dos elementos <context> e <switch> como filhos de um elemento <switch>.

Um elo de um documento, elemento <link>, permite a criação de elementos <bind>, que fazem referência a um dos papéis do conector que está sendo utilizado pelo elo. Entretanto, NCL também permite a criação de elementos <bind> utilizando papéis que não foram declarados no conector. Neste caso, o papel é declarado implicitamente como um papel

do tipo <attributeAssessment>, onde os atributos *eventType* e *attributeType* recebem os valores *attribution* e *nodeProperty*, respectivamente. Esta funcionalidade também não foi implementada pelo plugin.

Estas limitações serão tratadas em trabalhos futuros.

CAPÍTULO 6 – PLUGIN PARA USO DE TEMPLATES

Como os plugins mencionados no Capítulo 5 exigem conhecimento da linguagem NCL, sua utilização por autores sem esse conhecimento fica complicada. A fim de atender, também, a autores sem conhecimento de NCL, foi desenvolvido um plugin que permite a utilização de templates.

Com este plugin é possível criar um documento NCL indicando apenas as mídias que farão parte do documento, sem a necessidade de especificar características de funcionamento da aplicação. Qualquer template de composição criado com a linguagem XTemplate 3.0 [Dos Santos e Muchaluat-Saade 2011] pode ser utilizado por este plugin.

Além de permitir seu uso por autores sem conhecimento de NCL, este plugin também facilita a criação de aplicações que necessitam de grande quantidade de código, muitas vezes até repetitivo, em sua especificação. Um exemplo de como a utilização de um template pode ajudar na elaboração dessas aplicações é dado a seguir.

Considere um vídeo com duração de 1 hora e com 50 legendas para serem exibidas durante a apresentação do mesmo. Os momentos de iniciar e de encerrar a exibição de cada uma destas legendas devem estar sincronizados com o vídeo, ou melhor, com as âncoras do vídeo. Por exemplo, para este vídeo, devemos criar uma âncora para cada intervalo de tempo em que uma das legendas deve aparecer. O documento NCL deve definir ainda dois elos para cada par âncora/legenda do vídeo, um para iniciar a exibição da legenda e outro para finalizar sua exibição.

Esta aplicação, apesar de simples, possui muitas linhas de código NCL, que são bastante repetitivas. E, nestes casos, o autor tende a “copiar” e “colar” os trechos repetidos, trocando-se apenas o valor dos elementos que especificam cada âncora e cada legenda. Entretanto, caso o autor se esqueça de trocar o valor de algum elemento nesta ação de “copiar” e “colar”, erros podem ser facilmente gerados pelo fato do código ser muito extenso. Por outro lado, esse exemplo de documento é muito simples de ser especificado, caso um template que defina a sincronização das âncoras do vídeo com as legendas seja utilizado.

Outra questão importante é ajudar o autor a entender o que um template faz, ou seja, que tipo de aplicação pode ser criada ao se utilizar determinado template. Isso é importante pois o plugin oferecerá ao autor uma biblioteca com diversos templates disponíveis. Cada template especificado em XTemplate possui um arquivo RDF (*Resource Description Framework*) [W3C 2001] associado, cujo objetivo é explicar a funcionalidade de um template [Dos Santos e Muchaluat-Saade 2011]. O plugin para uso de templates exibe este texto

explicativo e, além disso, exibe também uma representação gráfica do template, a qual apresenta ao autor como as mídias estarão dispostas para o telespectador no documento final.

Este capítulo apresenta o funcionamento do plugin para uso de templates. Será explicado como é criada a representação gráfica de um template e como é gerado o documento NCL final. Os resultados do teste de usabilidade e limitações deste plugin encontram-se no final do capítulo.

6.1 REPRESENTAÇÃO GRÁFICA DE UM TEMPLATE

Além de permitir a utilização de templates, este plugin tem como objetivo ajudar o autor a escolher um template que atenda às suas necessidades. Para isso, além de facilitar o preenchimento do documento, ele é responsável pela criação de uma imagem gráfica para mostrar ao autor como será o leiaute criado com o uso do template escolhido.

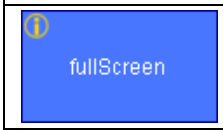
Para criar a representação gráfica do template, o plugin realiza a leitura do documento XTemplate e cria um objeto Java, do qual é possível extrair informações sobre o template. A partir destas informações, o plugin gera a visualização gráfica do mesmo.

Para esta representação gráfica, foi definido que um template é composto por uma ou mais telas. Uma tela pode ser entendida como o conjunto de nós que serão exibidos ao mesmo tempo e, provavelmente, em posições diferentes na tela da TV. Toda vez que uma nova mídia é apresentada ou finalizada, de acordo com a especificação do template, uma nova tela é criada para o template. Cada tela do template exibe as regiões criadas no mesmo, onde o autor deverá indicar as mídias que farão parte do documento NCL. A Figura 60 exemplifica o conceito de tela para este plugin. As Telas 1 e 2 são diferentes pois o botão azul não está sendo exibido na Tela 2.

Figura 60: Diferentes telas na visão do plugin de templates.

Um template de composição hipermídia especifica componentes genéricos, descrevem como estes estão relecionados e também define a posição espacial e características de apresentação dos mesmos, como comentado no Capítulo 3. O plugin de templates interpreta as especificações XML do template e gera símbolos gráficos de acordo com suas definições. A Tabela 22 apresenta os símbolos gráficos utilizados pelo plugin.

Tabela 22: Símbolos usados para gerar as telas do template.

Imagem	Significado
	Um retângulo azul representa um nó cujo conteúdo deverá ser escolhido pelo autor.
	Um retângulo vermelho indica um nó NCL específico que já está definido no template e cujo conteúdo não poderá ser redefinido.
	Um retângulo sombreado representa um conjunto de nós, ou seja, vários nós poderão ser exibidos na mesma região, em momentos distintos. O número máximo de nós também é definido pelo template. Neste caso, o autor pode indicar conteúdos diferentes para preencher o template.
	A imagem da letra "i" sobre um componente indica que o mesmo permite interatividade, ou seja, o telespectador poderá usar o controle remoto para interagir com a aplicação.
	A nota musical indica que um componente de áudio é utilizado no template. Assim como os retângulos, a nota pode ser azul ou vermelha e pode ou não ser sombreada.

A Figura 61, mostra como seria o template com leiaute das duas telas apresentadas na Figura 60.

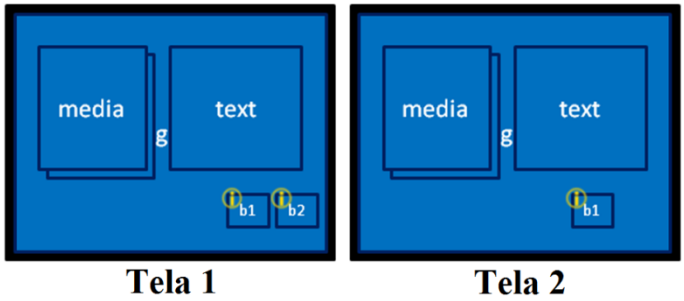


Figura 61: Representação gráfica do template para o documento representado pela Figura 60.

Já explicado o conceito de telas, a seguir apresenta-se como é determinada a quantidade de telas e as posições dos componentes nas mesmas para um template.

O documento XTemplate é lido com o uso da API aXT [Silva 2011], a qual, assim como a aNaa, é responsável pela criação de um objeto Java que contém informações sobre os elementos do template.

Uma vez de posse do objeto aXT que representa o template, é possível descobrir a URI do documento NCL que é referenciado pelo template. Este documento possui a base de descritores importada pelo template e, consequentemente, a base de regiões. Uma vez encontrado este documento, o mesmo é lido com o uso da API aNaa, que cria um outro objeto Java, desta vez para fornecer as informações necessárias sobre as posições de cada

componente do template. A relação entre os componentes e suas posições na tela se dá através do atributo *descriptor* do elemento <component> do template, o qual especifica um descritor, que, por sua vez, especifica uma região. A região contém as posições nas quais o componente será apresentado.

O próximo passo é considerar as informações do vocabulário do template. Através do mesmo, é possível conhecer quais são os componentes do template, suas interfaces e quantidades mínimas e máximas de cada um. Para cada componente, é criado um objeto Java que armazena estas informações e as informações sobre sua posição para, posteriormente, ser representado em uma das telas do template. Além disso, através do vocabulário, também é possível verificar quais conectores serão utilizados nos elos do template.

Após conhecer todos seus componentes, inicia-se a leitura do corpo do template. Quando algum componente do vocabulário é especificado pelo próprio template, utiliza-se o elemento <media>, assim como em um documento NCL, para especificá-lo. Depois de verificar todos os elementos <media> do template, é possível definir a cor de cada componente do vocabulário na representação do template, ou seja, vermelho para os que já estão especificados, e azul para os que devem ser especificados no documento NCL que usar o template.

Os próximos elementos a serem considerados são os elos do template. Através deles é possível conhecer as relações entre os componentes do template, descobrir quais elementos podem ser selecionados com uso do controle remoto e, inclusive, por qual tecla do controle.

Ao analisar um elo, todos os seus elementos <bind> são considerados, possibilitando a coleta de informações sobre as relações entre os componentes do template. Por exemplo, suponha o elo apresentado na Figura 62. Este indica que ao selecionar o componente ‘fig1’, deve-se parar a apresentação dos componentes ‘fundo’ e ‘fig1’ e iniciar a apresentação dos componentes ‘texto1’ e ‘b1’.

```
<link xtype="onKeySelectionStopStart">
    <bind role="onSelection" select="child::*[@xlabel='fig1']">
        <bindParam name="keyCode" value="ENTER"/>
    </bind>
    <bind role="stop" select="child::*[@xlabel='fundo']/>
    <bind role="stop" select="child::*[@xlabel='fig1']/>
    <bind role="start" select="child::*[@xlabel='texto1']/>
    <bind role="start" select="child::*[@xlabel='b1']/>
</link>
```

Figura 62: Exemplo de elo em um template.

Além de verificar que o componente ‘fig1’ permite interação através da tecla ‘ENTER’ do controle remoto, também é possível verificar a criação de uma nova tela para o

template ao selecionar ‘fig1’. A Figura 63 exibe estas telas. A Tela 1 apresenta os componentes ‘fig1’ e ‘fundo’, enquanto a Tela 2 apresenta os componentes ‘texto1’ e ‘b1’.

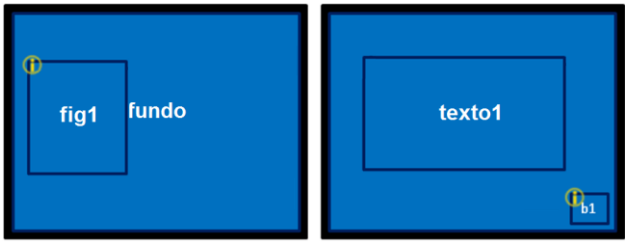


Figura 63: Telas formadas pelo elo da Figura 62.

Para definir como será a primeira tela do template, deve-se verificar os elementos <port> do mesmo, os quais indicam os primeiros componentes a serem iniciados. Além disso, para completar esta tela inicial, deve-se verificar se algum elo inicia a apresentação de outro(s) componente(s) quando o componente relacionado a uma porta for iniciado. Após determinar a primeira tela do template, faz-se uma varredura dos elos, verificando se ocorre ou não a formação de novas telas. Essa iteração ocorre até que todos os elos tenham sido verificados. Os elementos <port> e <link> encontrados dentro de elementos <for-each> também são analisados.

Depois de realizar esta leitura do template, é possível determinar suas possíveis telas, que componentes o autor pode escolher, suas quantidades mínimas e máximas e se suas interfaces, caso existam, precisam ou não ser definidas pelo autor.

Até agora, apenas foi mostrado como se dá a representação de templates que possuem leiaute, ou seja, que importam uma base de descritores de um documento NCL. Entretanto, também existem templates que não especificam leiaute algum.

Como a representação de um template foi feita baseando-se nas posições de seus componentes, esta representação é impossível para um template que não indica descritores e regiões. Então, como alternativa para esses templates, decidiu-se representá-los através de sua visão temporal ao invés de sua visão de leiaute, ou seja, representar ao autor quando ocorrerá a apresentação de seus componentes ao longo do tempo. Sendo assim, para criação da tela de um template sem leiaute, utiliza-se os símbolos gráficos exibidos na Tabela 23.

Fazendo uso dos símbolos para representação de templates sem leiaute, a Figura 64 mostra como seria a representação gráfica de um template sequencial e de um template paralelo.

Tabela 23: Símbolos usados para gerar as telas do template sem leiaute.

Imagem	Significado
	A elipse representa um componente e a letra ‘i’ indica que o mesmo permite interatividade.
	A letra “n” entre parênteses indica que um número ilimitado de componentes poderão ser utilizadas no template. Se essa quantidade for limitada, o valor máximo será exibido no lugar de “n”.
	O símbolo de reticências na horizontal indica que vários componentes serão exibidos sequencialmente, ou seja, um após o outro.
	O símbolo de reticências na vertical indica que vários componentes serão exibidos paralelamente, ou seja, ao mesmo tempo.

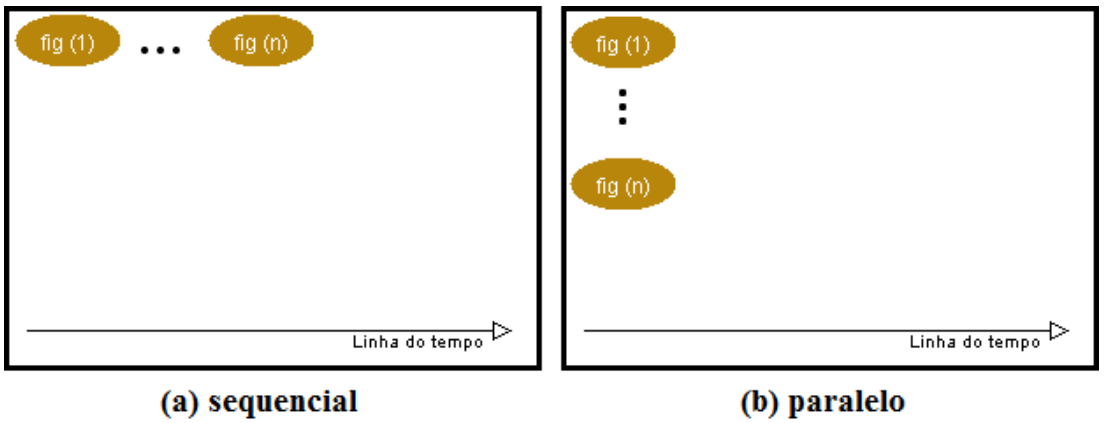


Figura 64: Telas de um template sequencial (a) e de um template paralelo (b).

6.2 PLUGIN PARA USO DE TEMPLATES DE COMPOSIÇÃO

Ao ser inserido no NEXT, o plugin cria uma pasta para representar sua base de templates. Nesta base o autor poderá inserir novos templates, permitindo que os mesmos sejam lidos e disponibilizados pelo plugin.

Inicialmente, o plugin realiza a leitura de todos os templates encontrados em sua base de templates e gera a representação gráfica de cada um. A Figura 65 apresenta a janela de exibição dos templates encontrados na base. Do lado esquerdo, é exibida uma lista de telas ilustrando a primeira tela de cada um dos templates. Ao selecionar uma das telas na lista, serão exibidas, ao lado direito, todas as diferentes telas do template selecionado, assim como o texto explicativo do mesmo.

Após selecionar um template, o autor deverá preenchê-lo com as mídias que irão compor o documento NCL. A Figura 66 exibe a tela de preenchimento do template, especificando que nós de mídia serão incluídos no documento NCL final. Ela ilustra o template de um quiz, que pode ser inicializado se o telespectador pressionar o botão azul do controle remoto, enquanto um vídeo de introdução está sendo apresentado. Para este template,

o autor deverá especificar o nó de vídeo, a pergunta e quatro opções de resposta para cada pergunta do quiz. Cada pergunta e suas opções de resposta são apresentadas em telas distintas, que devem ser duplicadas conforme o número de perguntas do quiz específico sendo criado.

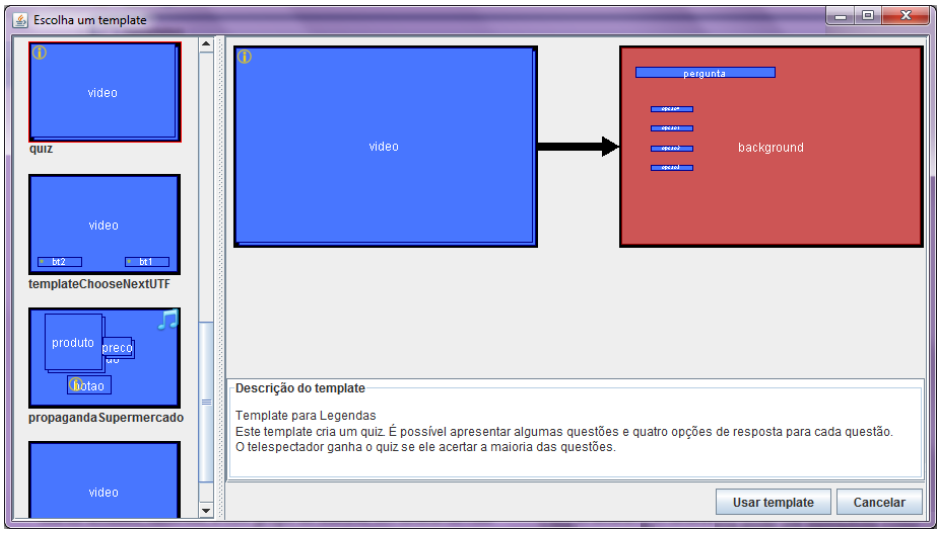


Figura 65: Janela de exibição dos templates.

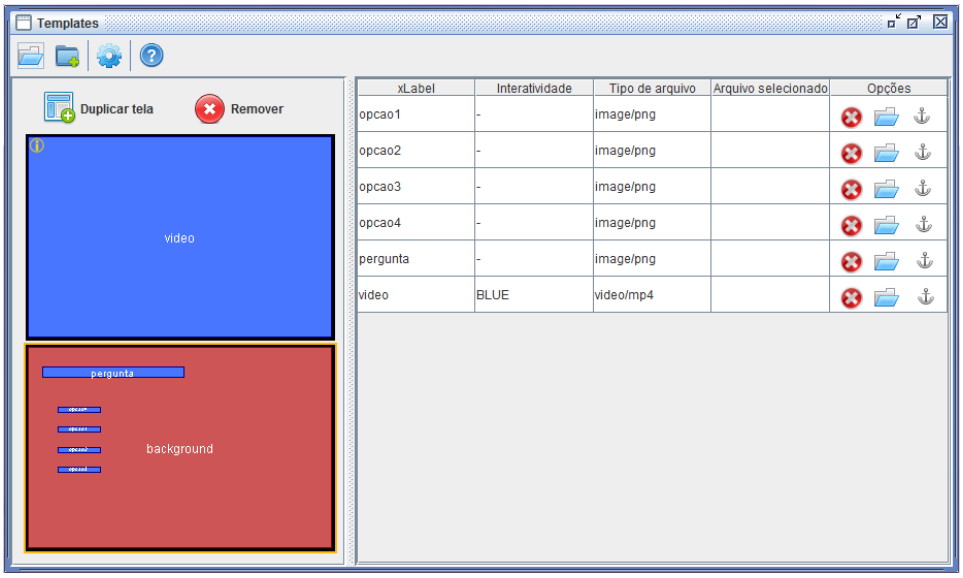


Figura 66: Tela de preenchimento do template.

No lado esquerdo da Figura 66, uma lista exibe todas as telas que compõem o template escolhido. Como algumas delas podem ser duplicadas ou removidas (aquelas que apresentam retângulos ou notas sombreados), botões com estas funcionalidades são oferecidos ao autor. No exemplo, a tela de pergunta deve ser duplicada para cada pergunta diferente que irá compor o quiz. Vale ressaltar que apenas telas que foram duplicadas podem ser removidas, já que excluir uma tela do próprio template poderia causar alguma falha no documento.

Ao selecionar uma das telas do template, uma tabela é exibida no lado direito. Esta tabela apresenta informações sobre os componentes da tela, os quais devem ser preenchidos

com mídias escolhidas pelo autor. Ela informa a identificação do componente (xlabel) e o tipo de mídia que deve ser inserida, conforme definido no template. O caminho da mídia selecionada é exibido na coluna “Arquivo selecionado”. Percebe-se pela Figura 66 que os componentes predefinidos pelo template (desenhados com cor vermelha) não são exibidos na tabela.

A coluna “Opções” oferece três botões, o primeiro e o segundo estão relacionados à escolha da mídia, enquanto o terceiro permite a criação de âncoras na mídia a ser utilizada. Alguns templates requerem a especificação de âncoras e, apesar de existirem diversos tipos de âncoras, esta versão do plugin só possibilita a criação de âncoras em mídias contínuas, onde o autor pode escolher seu início e fim.

Uma das principais características de uma aplicação para TV digital é permitir a interatividade, ou seja, o telespectador poderá interagir com a aplicação exibida. A letra “i”, presente em alguns componentes das telas do template, indica que o mesmo permite interatividade. Esta interatividade é possível clicando-se em alguma tecla do controle remoto, a qual também deve ser definida pelo template. A coluna “Interatividade” da tabela de preenchimento do template, informa qual tecla do controle remoto deve ser utilizada para prover interatividade. Caso esta coluna esteja preenchida com "-", significa que o componente não permite interatividade.

A funcionalidade dos botões exibidos no topo da Figura 66 são apresentados na Tabela 24.

Tabela 24: Funcionalidade dos botões do plugin para uso de templates.

Botão	Funcionalidade
	Abrir template
	Adicionar template
	Criar aplicação NCL
	Exibir menu de ajuda

Após realizar a escolha de todas as mídias que farão parte do documento e criar todas as âncoras necessárias, o autor deve clicar no botão “Criar aplicação NCL”. Após clicar nesse botão, o plugin cria um documento NCL estendido para utilizar templates, conforme explicado na Seção 3.2, o qual é processado pelo processador desenvolvido em [Dos Santos 2012], criando o documento NCL final. Este documento é então enviado para o NEXT e está pronto para ser utilizado por outros plugins ou para rodar na máquina virtual.

É importante ressaltar que, antes de criar o documento NCL estendido e, consequentemente, o documento NCL final, o plugin também é responsável por verificar se todas as mídias definidas no template existem, isto é, se as mesmas encontram-se no local especificado pelo template, se todas as mídias que deveriam ser escolhidas pelo autor foram definidas e se as mesmas apresentam a extensão conforme definido pelo template, assim como se suas quantidades mínima e máxima, definidas pelo template, estão sendo respeitadas. Essas verificações também são realizadas com relação às âncoras do documento. Dessa forma, garante-se que o documento final não esteja inconsistente.

Maiores detalhes sobre a implementação deste plugin podem ser encontrados no Anexo C.

6.3 TESTE DE USABILIDADE

O teste de usabilidade para avaliar a usabilidade do plugin de templates foi realizado com um grupo de dez alunos de computação que não conheciam a linguagem NCL (quatro de doutorado e seis de graduação) e oito alunos que já a conheciam (alunos de doutorado, mestrado e graduação). O perfil dos usuários para este teste era de usuários que realmente não conheciam a linguagem, ainda assim, decidiu-se incluir no teste usuários com conhecimento de NCL para que estes também pudessem contribuir com sugestões para o plugin.

Após uma breve explicação sobre a linguagem NCL e sobre o conceito de templates, os usuários tiveram que criar uma aplicação com a utilização do plugin de templates. Nenhuma explicação sobre o funcionamento do plugin foi dada. Eles apenas foram avisados que o plugin possui um menu de ajuda, o qual poderia ser consultado durante o desenvolvimento da aplicação.

Os usuários deveriam usar um template para criar uma propaganda de supermercado, onde eles deveriam sincronizar a imagem de três produtos e seus respectivos preços com âncoras de um áudio, as quais também deveriam ser criadas pelos alunos. As mídias a serem utilizadas também foram disponibilizadas. O código do template utilizado está disponível no Anexo A.

Após a conclusão da aplicação, os alunos responderam um questionário com perguntas relacionadas especificamente ao plugin. O questionário era composto por perguntas que buscavam verificar o grau de compreensão e facilidade de uso do plugin. As perguntas eram objetivas e as opções de resposta variavam numa escala de seis opções (de zero a cinco, onde zero era a pior opção e cinco a melhor).

A Tabela 25 apresenta o resultado obtido, o qual foi separado de acordo com o conhecimento do usuário sobre a linguagem NCL. Assim sendo, os usuários que marcaram a opção 2 ou menor na questão sobre conhecimento da linguagem (10 usuários), são representados na coluna 'Baixo' (baixo conhecimento de NCL) e os outros (8 usuários), na coluna 'Bom' (bom conhecimento de NCL). A tabela apresenta a média das respostas e também o desvio padrão.

Tabela 25: Perguntas e média dos resultados obtidos no questionário sobre a usabilidade do plugin de templates.

Pergunta	**Média / Desvio padrão**		**Conclusão do resultado**	
	Baixo	Bom	Baixo	Bom
Como você classifica o seu conhecimento em NCL?	0 / 0,3	4 / 0,7	Nenhum	Muito alto
Você já criou alguma aplicação NCL?	Nunca criaram	Sim, todos	-	-
Qual é o seu entendimento sobre o que são templates e sua utilidade?	4 / 0,8	4 / 1	Bom	Bom
Com relação à facilidade para compreender o plugin, que nota você daria ao mesmo?	4 / 0,8	3 / 0,9	Fácil	Mediano
Com que frequência você precisou utilizar a opção de ajuda do plugin para criar a aplicação?	1 / 1,1	1 / 1,4	Muito pouco	Muito pouco
Você conseguiu criar a aplicação NCL final com uso do plugin?	Sim, todos	Sim, todos	-	-
Como você classifica as imagens utilizadas para representar cada tela do template?	4 / ,07	4 / 1,2	Boas	Boas
Você consegue entender a funcionalidade do template através das telas exibidas e da sua descrição?	5 / 0,5	4 / 0,5	Perfeita-mente	Compre-ende bem
Como você classifica a tabela utilizada para definir as mídias que farão parte da aplicação?	4 / 0,8	4 / 0,7	Muito boa	Muito boa
Como você classifica a usabilidade na criação de âncoras nas mídias da aplicação?	4 / 0,8	3 / 1,2	Fácil	Mediano

Não houve diferenças significativas no uso do plugin entre os usuários que já conheciam a linguagem e os que não conheciam. Além de responderem o questionário, os alunos também contribuíram com sugestões para o aprimoramento do plugin. Algumas dessas sugestões, as quais ficam como trabalho futuro, são: melhorar a indicação de quando uma âncora pode ser criada na mídia e exibir a imagem da mídia escolhida no lugar dos retângulos.

6.4 LIMITAÇÕES DO PLUGIN

Como mencionado no Capítulo 4, Seção 4.5, algumas limitações da API aNaa acarretam algumas limitações ao NEXT. O mesmo ocorre para o plugin de uso de templates.

A API aXT, utilizada para realizar a leitura do template, estende a API aNaa e, por isso, possui também as mesmas limitações dessa. Uma destas limitações não permite abrir um documento NCL que possui apenas um cabeçalho, o qual possui uma base de regras. Assim sendo, templates que utilizam switches não podem ser utilizados no plugin, uma vez que não será possível abrir o documento NCL que possui a base de descritores e, consequentemente, os componentes responsáveis por representar os switches serão ignorados.

Como uma nova versão da API aNaa está em andamento, uma nova versão da aXT também deve ser criada a fim de ser compatível com a esta nova versão e permitir o uso de templates que utilizam switches.

Este capítulo apresentou o plugin do editor NEXT que permite o uso de templates de composição para criação de documentos NCL. De acordo com resultados de testes de usabilidade, diferente dos outros plugins comentados no Capítulo 5, o plugin para uso de templates permite a criação de documentos multimídia por usuários que não têm conhecimento da linguagem NCL. Sendo assim, o editor NEXT pode ser utilizado por autores com diferentes habilidades e conhecimentos sobre a linguagem NCL, o que representa uma das principais contribuições desta dissertação.

CAPÍTULO 7 – CONCLUSÃO

O cenário atual de TV digital requer metodologias fáceis e rápidas para a criação de aplicações multimídia interativas, de forma que autores com diferentes perfis possam contribuir para seu desenvolvimento. Com o objetivo de minimizar as dificuldades relacionadas à criação de aplicativos, são oferecidos os sistemas de autoria multimídia. Tais sistemas visam facilitar o trabalho de autoria oferecendo um ambiente gráfico para criação e edição de documentos multimídia.

Esta dissertação apresentou uma proposta de editor gráfico para autoria multimídia baseada nas linguagens NCL e XTemplate. O editor proposto, chamado NEXT - *NCL Editor supporting XTemplate*, oferece a possibilidade de uso de templates de composição, especificados com a linguagem XTemplate 3.0, para a criação de documentos NCL. Templates de composição definem estruturas genéricas que podem ser reutilizadas em documentos distintos, facilitando bastante sua criação. Além da proposta do editor, este trabalho identificou um conjunto de requisitos importantes a serem oferecidos por editores gráficos multimídia, a partir do estudo de diversos trabalhos relacionados. O editor NEXT foi desenvolvido de forma a atender os requisitos identificados como importantes.

Pode-se ressaltar como principal contribuição desta dissertação, um editor gráfico multimídia que permite a edição de qualquer documento NCL 3.0 com suporte ao uso de templates. Nenhum outro editor gráfico para documentos NCL oferece o uso de templates para criação de documentos multimídia. Além disso, o editor é extensível através da inserção de novas funcionalidades com a instalação de plugins, está disponível em dois idiomas (inglês e português), pode ser utilizado em diferentes plataformas e oferece a análise estrutural e comportamental do documento multimídia, permitindo a identificação de erros de autoria que possam ser cometidos pelo autor do documento.

No contexto deste trabalho, quatro plugins foram desenvolvidos para uso no NEXT: editor de conectores, visão de leiaute, visão estrutural e plugin para uso de templates. Todos oferecem um ambiente gráfico visando facilitar o desenvolvimento de aplicações NCL. Além disso, os dois primeiros também podem ser utilizados como aplicações stand-alone, ou seja, fora do ambiente do NEXT.

O plugin de conectores permite a especificação de conectores causais, os quais são utilizados pelos elos de um documento NCL. O plugin de visão de leiaute facilita a criação de bases de regiões, regiões e descritores. Já o plugin de visão estrutural permite a definição do corpo do documento, assim como de seus elementos. E, por fim, o plugin de templates

permite o uso de templates de composição para a criação de documentos NCL, inclusive por autores sem conhecimento da linguagem. Através do uso destes plugins, é possível criar um documento NCL completo, definindo todos os seus elementos básicos.

Testes de usabilidade também foram realizados a fim de avaliar o uso desses plugins por usuários com e sem conhecimento da linguagem NCL.

A Seção 7.1 apresenta as principais características e contribuições do editor desenvolvido e realiza uma comparação entre o mesmo e os trabalhos relacionados, apresentados no Capítulo 2. Na Seção 7.2, são apresentados os trabalhos futuros.

7.1 COMPARAÇÃO COM TRABALHOS RELACIONADOS

NEXT foi desenvolvido em Java a fim de possibilitar sua utilização em diferentes plataformas e está atualmente disponível em dois idiomas. Além disso, acrescentar novas opções de idiomas não é uma tarefa complicada, uma vez que o mesmo foi desenvolvido a fim de prover esta facilidade.

NEXT permite a edição de qualquer documento NCL independentemente de onde este tenha sido criado e fornece a análise do documento, ajudando o autor a criar aplicações consistentes.

O editor permite que autores sem nenhum conhecimento da linguagem NCL possam criar aplicações através da utilização de templates de composição criados com a linguagem XTemplate 3.0. Neste caso, o autor apenas precisará informar quais mídias farão parte da aplicação final. Seu plugin para uso de templates também gera automaticamente uma representação gráfica dos mesmos, facilitando a compreensão do template por parte do autor.

NEXT permite a inclusão de novas funcionalidades através de plugins, possibilitando seu uso por diferentes perfis de autores. Plugins podem ser facilmente desenvolvidos por programadores Java uma vez que eles precisam seguir poucas regras, simples e bem definidas.

Considerando os requisitos identificados no Capítulo 2, pode-se dizer que o NEXT atende os seguintes requisitos: permite seu uso por diferentes perfis de autores, é extensível, fornece portabilidade, possibilita o uso de templates, oferece suporte a templates genéricos, oferece diferentes visões do documento e realiza a análise de consistência estrutural e comportamental do documento. Dentre os requisitos apontados, somente a simulação do documento multimídia ainda não é contemplada na implementação atual do NEXT.

A Tabela 26 lista os requisitos básicos apresentados no Capítulo 2 e faz uma comparação do NEXT com os editores estudados, também apresentados no Capítulo 2.

Tabela 26: Comparação do NEXT com os editores apresentados no Capítulo 2.

Requisitos / Editores	ASCCT	DEMAIS	Composer 3	LAMP	EDITEC	T-B MHP AT	MMPE	Berimbau	ISB Designer	NEXT
Diferentes perfis de autores	✓		✓	?			✓	✓		✓
Extensibilidade	?		✓			✓			✓	✓
Portabilidade	?	✓	✓	✓	✓	?			✓	✓
Simulação		✓		✓			✓			
Uso de templates	✓	✓		✓		✓	✓			✓
Suporte a templates genéricos	✓			✓	✓	✓				✓
Diferentes visões		✓	✓	✓	✓	✓	✓		✓	✓
Análise de consistência	+/-		+/-			?				✓

✓ contemplado pela ferramenta
? não foi possível definir
+/- atende parcialmente

7.2 TRABALHOS FUTUROS

Como trabalho futuro, outros plugins podem ser desenvolvidos para estender as funcionalidades do editor, oferecendo mais facilidade na criação de documentos. Um plugin que ofereça uma visão temporal é de grande ajuda no desenvolvimento de aplicações por auxiliar o autor a compreender o acontecimento dos eventos ao longo do tempo. Uma visão textual editável também agradaria bastante a autores que já conhecem bem a linguagem NCL. Além desses, outro trabalho importante é a simulação da execução de um documento NCL.

Além do desenvolvimento de novos plugins, vale ressaltar a importância de criar uma nova versão do NEXT, onde este passe a utilizar a nova versão da API aNaa. Em sua nova versão, a API resolve os problemas de referências não suportados pela versão atualmente utilizada no NEXT. Desta forma, seria possível abrir documentos NCL que fazem referências a outros documentos, resolvendo algumas limitações existentes na implementação atual.

Os testes de usabilidade do editor e de seus plugins permitiram coletar ideias de refinamentos para os mesmos. Os usuários contribuíram com sugestões para aperfeiçoar o uso da ferramenta, as quais devem ser analisadas e, sendo viáveis, implementadas. Ainda assim, novos testes podem ser realizados para reavaliar a qualidade da interface dos plugins com o

usuário. O estudo realizado em [Jeffries et al. 1991] mostra que o método proposto por [Nielsen e Molich 1990] tem apresentado bons resultados e por isso pode ser considerado.

Um outro importante trabalho futuro é a integração do editor EDITEC ao NEXT. EDITEC foi desenvolvido como uma aplicação Java stand-alone e permite a edição gráfica de templates de composição. Através da adaptação do EDITEC como um plugin do NEXT, será possível criar graficamente novos templates de composição utilizando o NEXT.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABNT 2007] ABNT NBR 15606-2:2007 standard. Digital terrestrial television - Data coding and transmission specification for digital broadcasting - Part 2: Ginga-NCL for fixed and mobile receivers - XML application language for application coding. 2007.

[Advanced Distributed Learning 2006] Advanced Distributed Learning (ADL) (2006) Sharable Content Object Reference Model (SCORM®) 2004, 3rd edn.

[Araújo 2012] Araújo, E. C. Projetando Aplicações para TVDI através de Storyboards Interativos. Dissertação de Mestrado. PUC-Rio, Rio de Janeiro, 2012.

[Bailey et al. 2001] Bailey, B. P., Kostan, J. A., Carlis, J. V. – DEMAIS: Designing Multimedia Applications with Interactive Storyboards. International Multimedia Conference, páginas 241-250, Ottawa , 2001.

[Berimbau iTV Author 2011] Berimbau iTV Author, http://www.batuque.tv/, acessado em novembro de 2011.

[Celentano e Gaggi 2003] Celentano, A. e Gaggi O. – Template-based generation of multimedia presentations. International Journal of Software Engineering and Knowledge Engineering, Vol. 13, Nº 4, 419-445, 2003.

[Chiao et al. 2006] Chiao H., Hsu K., Chen Y. e Yuan S. A template-based MHP authoring tool. In Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, página 138, Seoul, 2006.

[Damasceno et al. 2010] Damasceno, J. R., Santos, J. A. F e Saade, D. C. M. – EDITEC: Editor Gráfico de Templates de Composição para Facilitar a Autoria de Programas pata TV Digital Interativa. WebMedia, Belo Horizonte, 2010.

[Deltour e Roisin 2006] Deltour R. e Roisin C. The LimSee3 Multimedia Authoring Model. In Proceedings of the 2006 ACM symposium on Document engineering, páginas 173-175, Amsterdã, 2006.

[Dos Santos e Muchaluat-Saade 2011] Dos Santos, Joel; Muchaluat-Saade, Débora. XTemplate 3.0: spatio-temporal semantics and structure reuse for hypermedia compositions Multimedia Tools and Applications.Springer Netherlands, 2011. Issn: 1380-7501. Doi: 10.1007/s11042-011-0732-2.

[Dos Santos et al. 2012]

Dos Santos, J. A. F., Silva, J. V., Vasconcelos, R., Schau, R., Werner, C. e Muchaluat-Saade, D. C. aNa: API for NCL Authoring. WebMedia - Workshop de Ferramentas e Aplicações, São Paulo, 2012.

[Dos Santos 2012] Dos Santos, J. A. F. Multimedia and hypermedia document validation and verification using a model driven approach. Dissertação de Mestrado. Universidade Federal Fluminense, Niterói, 2012.

[Extensible Markup Language 2011] Extensible Markup Language (XML) 1.1 (Second Edition) - W3C Recommendation 16 August 2006, Setembro 2006, http://www.w3.org/TR/2006/REC-xml11-20060816/, acessado em abril de 2011.

[Jeffries et al. 1991] Jeffries, R., Miller, J.R., Wharton, C., Uyeda, K.M. User interface evaluation en the real world: a comparison of four techniques. ACM, 1991.

[Hong e Landay 2000] Hong, J.I. e J. A. Landay. - SATIN: A toolkit for Informal Ink-based Applications. User Interface Software and Technology, páginas 63-72, 2000.

[IANA 2012] IANA – MIME Media Types, http://www.iana.org/assignments/media-types/index.html, acessado em junho de 2012.

[International Telecommunication Union 2009] International Telecommunication Union (ITU), recommendation H.761, Nested Context Language (NCL) and Ginga-NCL for IPTV services, available in http://www.itu.int/itu-t/workprog/wp_item.aspx?isn=6186, 2009.

[Java 2012] Java swing, http://docs.oracle.com/javase/6/docs/technotes/guides/swing/, acessado em maio de 2012.

[Jokela et al. 2008] Jokela T., Lehikoinen J. T., Korhonen H. Mobile multimedia presentation editor: enabling creation of audio-visual stories on mobile devices. In ACM Special Interest Group on Computer-Human Interaction, páginas 63-72, Florença, 2008.

[Lima et al. 2010] Lima, B. S., Azevedo, R. G., Moreno, M. F. e Soares, L. F. – Composer 3: Ambiente de autoria extensível, adaptável e multiplataforma. WebMedia – Workshop de TV Digital Interativa (WTVDI), 2010.

[López et al. 2008] López, M. R., Redondo, R. P. D., Vilas, A. F., Arias, J. J. P, Nores, M. L., Duque, J. G., Solla, A. G. and Cabrer, M. R. T-MAESTRO and its authoring tool: using adaptation to integrate entertainment into personalized t-learning. Multimedia Tools and Applications, 40(3):409–451, 2008.

[Muchaluat-Saade e Soares 2002] Muchaluat-Saade D. C. e Soares L. F. G., XConnector e XTemplate: Estendendo XLink para Aumentar Expressividade e Reuso, VIII Simpósio Brasileiro de Sistemas Multimídia e Hipermídia - SBMídia2002, Fortaleza, Outubro de 2002.

[Multimedia Home Plataform 2010] Multimedia Home Plataform - MHP, http://www.mhp.org/, acessado em junho de 2010.

[NCL Composer 2012] NCL Composer, http://composer.telemidia.puc-rio.br/, acessado em maio de 2012.

[Nielsen e Molich 1990] Nielsen, J., Molich, R. Heuristic evaluation of user interfaces. In: EMPOWERING PEOPLE - CHI'90 CONFERENCE Proceedings. New York: ACMPress, 1990.

[Silva 2011] Silva, F. S.,TVDI – Pesquisas em Televisão Digital Interativa. Relatório de Iniciação Científica, Departamento de Ciência da Computação da Universidade Federal Fluminense, 2011.

[Silva e Muchaluat-Saade 2011] Silva, J. V. e Muchaluat-Saade D. C., Editor Gráfico de Conectores Hipermídia para Linguagem NCL 3.0. WebMedia, Florianópolis, 2011.

[Silva e Muchaluat-Saade 2012] Silva, J. V. e Muchaluat-Saade D. C., NEXT – Editor Gráfico para Autoria de Documentos NCL com Suporte a Templates de Composição. WebMedia, São Paulo, 2012.

[Soares e Barbosa 2009] Soares, L. F. G e Barbosa S. D. J. Programando em NCL 3.0. Editora Campus/Elsevier, 2009.

[Soares e Rodrigues 2005] Soares, L. F. G. e Rodrigues, R. F. Nested Context Model 3.0 Part 1 - NCM Core. Monografias em ciência da computação, Departamento de Informática, PUC-Rio, Rio de Janeiro, Maio 2005.

[Soares e Rodrigues 2006] Soares, L. F. G. e Rodrigues, R. F. (2006) "Nested Context Model 3.0 Part 8 – NCL (Nested Context Language) Digital TV Profiles." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, no 35/06. Rio de Janeiro. Outubro de 2006. ISSN0103-9741.

[Soares et al. 2007] Soares, L. F. G., Rodrigues, R. F. e Moreno, M. F. "Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System", Journal of the Brazilian Computer Society, no. 1; Vol. 13; Mar. 2007.

[Synchronized Multimedia Integration Language 2010] Synchronized Multimedia Integration Language - SMIL, recomendação do W3C, http://www.w3.org/TR/SMIL/, acessado em dezembro de 2010.

[W3C 1999a] W3C World-Wide Web Consortium. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath, 1999. W3C Recommendation.

[W3C 1999b] W3C World-Wide Web Consortium. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt, 1999. W3C Recommendation.

[W3C 2001] W3C World-Wide Web Consortium. W3C Semantic Web Activity. http://www.w3.org/2001/sw/, 2001.

ANEXO A – TEMPLATE ‘PROPAGANDA DE SUPERMERCADO’

```
<?xml version="1.0" encoding="UTF-8"?>
<xtemplate id="propagandaSupermercado" descrption="Xtemplate Propaganda de Supermercado"
        xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xmlns:xt='http://www.midiacom.uff.br/gtvd/XTemplate30/XTemplateENCL'
        xsi:schemaLocation='http://www.midiacom.uff.br/gtvd/XTemplate30/XTemplateENCL
http://www.midiacom.uff.br/gtvd/XTemplate30/profiles/XTPENCL.xsd'>

<head>
        <connectorBase>
                <importBase alias="connBase" documentURI="connectorBase.ncl"/>
        </connectorBase>
        <descriptorBase>
                <importBase alias="descBase" documentURI="descriptorBase.ncl"/>
        </descriptorBase>
</head>

<vocabulary>
        <component xlabel="fundo" xtype="image/png" descriptor="descBase#dpTV" maxOccurs="1"/>
        <component xlabel="audio" xtype="audio/mp3" descriptor="descBase#dpAudio"
            maxOccurs="1">
                <port xlabel="prodX" minOccurs="3" maxOccurs="3"/>
        </component>
        <component xlabel="produto" xtype="image/png" descriptor="descBase#dpProd"
            minOccurs="3" maxOccurs="3">
                <port xlabel="bounds"/>
        </component>
        <component xlabel="preco" xtype="image/png" descriptor="descBase#dpPrec"
            maxOccurs="3"/>
        <component xlabel="botao" xtype="image/png" descriptor="descBase#dpBotao"
            maxOccurs="1"/>
        <component xlabel="compras" maxOccurs="1">
                <port xlabel="compProd1"/>
                <port xlabel="compProd2"/>
                <port xlabel="compProd3"/>
        </component>

        <connector src="connBase#onBeginStartN" xlabel="onBeginStartN" maxOccurs="unbounded"/>
        <connector src="connBase#onEndStopN" xlabel="onEndStopN" maxOccurs="unbounded"/>
        <connector src="connBase#onKeySelectionPresentingSet" xlabel="onKeySelectionPresentingSet"
            maxOccurs="unbounded"/>
        <connector src="connBase#onEndTestStartN" xlabel="onEndTestStartN"
            maxOccurs="unbounded"/>
        <connector src="connBase#onEndTestSetStartN" xlabel="onEndTestSetStartN"
            maxOccurs="unbounded"/>
        <connector src="connBase#onEndStopN" xlabel="onEndStopN" maxOccurs="unbounded"/>
</vocabulary>

<body>
        <port id="pFundo" component="fundo"/>

        <media id="compras" xlabel="compras" type="application/x-ginga-settings">
                <property name="compProd1" value ="false" xlabel="compProd1"/>
                <property name="compProd2" value ="false" xlabel="compProd2"/>
                <property name="compProd3" value ="false" xlabel="compProd3"/>
        </media>
```

```
<!-- INICIO -->
<link xtype="onBeginStartN">
        <bind role="onBegin" select="child::*[@xlabel='fundo']"/>
        <bind role="start"   select="child::*[@xlabel='audio']"/>
        <bind role="start"   select="child::*[@xlabel='botao']"/>
</link>

<!-- EXIBE PRODUTOS -->
<variable name="i" select="1"/>
<for-each select="child::*[@xlabel='audio']/child::area[@xlabel='prodX']">
        <link xtype="onBeginStartN">
                <bind role="onBegin" select="current()"/>
                <bind role="start"   select="child::*[@xlabel='produto'][position() = $i]"/>
                <bind role="start"   select="child::*[@xlabel='preco'][position() = $i]"/>
        </link>
        <link xtype="onEndStopN">
                <bind role="onEnd" select="current()"/>
                <bind role="stop"   select="child::*[@xlabel='produto'][position() = $i]"/>
                <bind role="stop"   select="child::*[@xlabel='preco'][position() = $i]"/>
        </link>
<variable name="i" select="$i + 1"/>
</for-each>

<!-- COMPRA PRODUTOS -->
<link xtype="onKeySelectionPresentingSet">
        <bind role="onSelection" select="child::*[@xlabel='botao']">
                <bindParam name="keyCode" value="ENTER"/>
        </bind>
        <bind role="isPresenting" select="child::*[@xlabel='produto'][position() = 1]"/>
        <bind role="set"
            select="child::media[@xlabel='compras']/child::property[@xlabel='compProd1']">
                <bindParam name="var" value="true"/>
        </bind>
</link>

<link xtype="onKeySelectionPresentingSet">
        <bind role="onSelection" select="child::*[@xlabel='botao']">
                <bindParam name="keyCode" value="ENTER"/>
        </bind>
        <bind role="isPresenting" select="child::*[@xlabel='produto'][position() = 2]"/>
        <bind role="set"
            select="child::media[@xlabel='compras']/child::property[@xlabel='compProd2']">
                <bindParam name="var" value="true"/>
        </bind>
</link>

<link xtype="onKeySelectionPresentingSet">
        <bind role="onSelection" select="child::*[@xlabel='botao']">
                <bindParam name="keyCode" value="ENTER"/>
        </bind>
        <bind role="isPresenting" select="child::*[@xlabel='produto'][position() = 3]"/>
        <bind role="set"
            select="child::media[@xlabel='compras']/child::property[@xlabel='compProd3']">
                <bindParam name="var" value="true"/>
        </bind>
</link>
```

```
<variable name="j" select="1"/>
<for-each select="child::*[@xlabel='produto']">
        <property name="bounds" xlabel="bounds"/>
        <variable name="j" select="$j + 1"/>
</for-each>

<!-- APRESENTA PRODUTOS COMPRADOS -->
<link xtype="onEndStopN">
        <bind role="onEnd"
            select="child::*[@xlabel='audio']/child::area[@xlabel='prodX'][position() = 3]"/>
        <bind role="stop" select="child::*[@xlabel='botao']"/>
</link>

<link xtype="onEndTestSetStartN">
        <bind role="onEnd" select="child::*[@xlabel='botao']"/>
        <bind role="test"
            select="child::*[@xlabel='compras']/child::property[@xlabel='compProd1']">
                <bindParam name="var" value="true"/>
        </bind>
        <bind role="set" select="child::*[@xlabel='produto'][position() =
            1]/child::*[@xlabel='bounds']">
                <bindParam name='loc' value='1%, 15%, 60%, 90%'/>
        </bind>
        <bind role="start" select="child::*[@xlabel='produto'][position() = 1]"/>
</link>

<link xtype="onEndTestSetStartN">
        <bind role="onEnd" select="child::*[@xlabel='botao']"/>
        <bind role="test"
            select="child::*[@xlabel='compras']/child::property[@xlabel='compProd2']">
                <bindParam name="var" value="true"/>
        </bind>
        <bind role="set" select="child::*[@xlabel='produto'][position() =
            2]/child::property[@xlabel='bounds']">
                <bindParam name='loc' value='47%, 15%, 60%, 90%'/>
        </bind>
        <bind role="start" select="child::*[@xlabel='produto'][position() = 2]"/>
</link>

<link xtype="onEndTestSetStartN">
        <bind role="onEnd" select="child::*[@xlabel='botao']"/>
        <bind role="test"
            select="child::*[@xlabel='compras']/child::property[@xlabel='compProd3']">
                <bindParam name="var" value="true"/>
        </bind>
        <bind role="set" select="child::*[@xlabel='produto'][position() =
            3]/child::property[@xlabel='bounds']">
                <bindParam name='loc' value='99%, 15%, 60%, 90%'/>
        </bind>
        <bind role="start" select="child::*[@xlabel='produto'][position() = 3]"/>
</link>

<link xtype="onEndStopN">
        <bind role="onEnd" select="child::*[@xlabel='audio']"/>
        <bind role="stop" select="child::*[@xlabel='botao']"/>
        <bind role="stop" select="child::*[@xlabel='fundo']"/>
        <bind role="stop" select="child::*[@xlabel='produto'][position() = 1]"/>
        <bind role="stop" select="child::*[@xlabel='produto'][position() = 2]"/>
        <bind role="stop" select="child::*[@xlabel='produto'][position() = 3]"/>
```

```
        </link>
</body>

<constraints>
        <constraint select="count(child::media[@xlabel='produto']) = 3" description="O número de
            produtos deve ser igual a três"/>
        <constraint select="count(child::media[@xlabel='preco']) = 3" description="O número de preços
            deve ser igual a três"/>
        <constraint select="count(descendant::area[@xlabel='prodX']) = 3" description="O número de
            âncoras (prodX) do áudio deve ser igual a três"/>
</constraints>
</xtemplate>
```

ANEXO B – QUESTIONÁRIOS DOS TESTES DE USABILIDADE

QUESTIONÁRIO SOBRE O PLUGIN PARA A CRIAÇÃO DE CONECTORES NA LINGUAGEM NCL

1) Como você classifica o seu conhecimento na linguagem NCL?

( )	( )	( )	( )	( )	( )
Nenhum	Muito pouco	Pouco	Bom	Alto	Muito alto

2) Como você classifica o seu conhecimento em conectores NCL?

( )	( )	( )	( )	( )	( )
Nenhum	Muito pouco	Pouco	Bom	Alto	Muito alto

3) Você já criou algum aplicativo utilizando a linguagem NCL?

( ) Sim ( ) Não (passe para a pergunta 5)

4) Com que frequência você cria os conectores que utiliza nas suas aplicações?

( )	( )	( )	( )	( )	( )
Nunca	Raramente	Poucas vezes	De vez em quando	Muitas vezes	Sempre

5) Com relação ao tempo para compreender o plugin, que nota você daria ao mesmo?

( )	( )	( )	( )	( )	( )
Não compreendi	Demorei muito	Demorei	Razoável	Rápido	Muito rápido

6) Com relação ao grau de usabilidade (facilidade de uso), que nota você daria ao plugin?

( )	( )	( )	( )	( )	( )
Péssimo	Ruim	Razoável	Bom	Muito bom	Excelente

7) Com relação ao grau de desempenho (tempo de processamento, falhas do programa), que nota você daria ao plugin?

( )	( )	( )	( )	( )	( )
Péssimo	Ruim	Razoável	Bom	Muito bom	Excelente

8) Que nota você daria para a importância (necessidade) do plugin?

( )	( )	( )	( )	( )	( )
Desnecessário	Pouco útil	Indiferente	Útil	Muito útil	Indispensável

9) Com relação às figuras utilizadas para representar os papéis, que nota você daria em relação à clareza das mesmas? Alguma sugestão?

( )	( )	( )	( )	( )	( )
Não são claras	Pouco claras	Razoáveis	Claras	Bastante claras	Excelentes

10) Você conseguiu criar os conectores da lista?

( ) Sim, todos ( ) A maioria ( ) Poucos ( ) Não

Sugestões:

QUESTIONÁRIO NEXT

1) Como você classifica seu conhecimento na linguagem NCL?

Considere: zero indica nenhum conhecimento e 5 conhecimento avançado.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

2) Você já criou alguma aplicação NCL?

( ) Sim ( ) Não

PLUGIN DE REGIÕES E DESCRITORES

3) Com relação à facilidade para compreender como usar o plugin, que nota você daria a mesma?

Considere: zero indica que não compreendeu e 5 compreendeu facilmente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

4) Com relação à criação de regiões, como você classifica a usabilidade do plugin?

Considere: zero indica muito complicado e 5 muito fácil.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

5) Com relação à criação de descritores, como você classifica a usabilidade do plugin?

Considere: zero indica muito complicado e 5 muito fácil.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

6) Como você classifica a janela de criação/edição de descritores?

Considere: zero indica péssimo e 5 excelente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Se você marcou a opção 3, ou menor, indique o que não ficou bom ou como poderia melhorar.

PLUGIN VISÃO ESTRUTURAL

7) Com relação à facilidade para compreender como usar o plugin, que nota você daria a mesma?

Considere: zero indica que não compreendeu e 5 compreendeu facilmente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

8) Com que frequência você precisou utilizar a opção de ajuda?

Considere: zero indica que não utilizou e 5 utilizou com muita frequência.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

9) Como você classifica as imagens utilizadas para representar os elementos do contexto?

Considere: zero indica péssimo e 5 excelente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

10) Como você classifica a forma de exibição dos elementos do contexto?

Considere: zero indica péssimo e 5 excelente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Se você marcou a opção 3, ou menor, indique o que não ficou bom ou como poderia melhorar.

_____

11) Com relação à facilidade para criar o elemento bind nos elos, que nota você daria a mesma?

Considere: zero indica que não compreendeu e 5 compreendeu facilmente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Se você marcou a opção 3, ou menor, indique o que não ficou bom ou como poderia melhorar.

_____

12) Com relação à facilidade para criar regras, que nota você daria a mesma?

Considere: zero indica que não compreendeu e 5 compreendeu facilmente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

NEXT

13) Você conseguiu criar a aplicação NCL completa?

( ) Sim ( ) Não

14) Com relação à importância do repositório de mídias, que nota você daria a mesma?

Considere: zero indica irrelevante e 5 fundamental.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

15) Sua aplicação funcionou na máquina virtual conforme o esperado?

( ) Sim ( ) Não

Sugestões:

_____

QUESTIONÁRIO SOBRE O PLUGIN PARA A CRIAÇÃO DE APLICAÇÕES NCL COM O USO DE TEMPLATES

1) Como você classifica seu conhecimento na linguagem NCL?

Considere: zero indica nenhum conhecimento e 5 conhecimento avançado.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

2) Você já criou alguma aplicação NCL?

( ) Sim ( ) Não

3) Qual é o seu entendimento sobre o que são templates e sua utilidade?

Considere: zero indica que não compreendeu e 5 compreende perfeitamente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

4) Com relação à facilidade para compreender como usar o plugin, que nota você daria a mesma?

Considere: zero indica que não compreendeu e 5 compreendeu facilmente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

5) Com que frequência você precisou utilizar a opção de ajuda do plugin para criar a aplicação?

Considere: zero indica que não utilizou e 5 utilizou com muita frequência.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

6) Você conseguiu criar a aplicação NCL final com uso do plugin?

( ) Sim ( ) Não

7) Como você classifica as imagens utilizadas para representar cada tela do template?

Considere: zero indica péssimo e 5 excelente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Se você marcou a opção 3, ou menor, indique o que não ficou bom ou como poderia melhorar.

_____

8) Você consegue entender a funcionalidade do template através das telas exibidas e da sua descrição?

Considere: zero indica que não entendeu e 5 que entendeu perfeitamente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Se você marcou a opção 3, ou menor, indique o que não ficou bom ou como poderia melhorar.

_____

9) Como você classifica a tabela utilizada para definir as mídias que farão parte da aplicação?

Considere: zero indica péssimo e 5 excelente.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Se você marcou a opção 3, ou menor, indique o que não ficou bom ou como poderia melhorar.

_____

10) Como você classifica a usabilidade na criação de âncoras nas mídias da aplicação?

Considere: zero indica muito complicado e 5 muito fácil.

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Sugestões:

_____

ANEXO C – IMPLEMENTAÇÃO DO PLUGIN PARA USO DE TEMPLATES

Este anexo descreve, com maiores detalhes, a implementação de cada um dos pacotes do plugin para uso de templates.

PACOTE MYPLUGIN

O pacote *myPlugin* possui as classes obrigatórias que todo plugin deve implementar, como explicado no Capítulo 5, Seção 5.1. Ele é composto pelas classes *PluginInfo* e *StartPlugin*. A primeira possui as informações de que o NEXT precisa: o apelido do plugin e seus elementos NCL de interesse. Apesar de não tratar nenhuma alteração que ocorra no documento NCL, já que este plugin apenas cria um novo documento a partir de um template, seu elemento de interesse foi definido como *Element.Doc*.

A classe *StartPlugin* é responsável pela exibição da janela principal do plugin, conforme especificado na criação de plugins. Ela cria a pasta 'Templates' onde os templates adicionados pelo usuário são armazenados e fornece as opções (abrir, adicionar e criar aplicação) para utilizá-los.

Quando o usuário deseja escolher um template, esta classe realiza a leitura de cada um dos templates encontrados na pasta 'Templates', cria sua representação gráfica e insere em uma lista que possui todos os templates. Esta lista é exibida para o usuário e o mesmo deve escolher o template a ser utilizado.

PACOTE TEMPLATE

O pacote *template* é responsável por criar a representação gráfica de um template. Ele é composto por nove classes, explicadas a seguir.

A classe *Template* é responsável por iniciar a leitura dos documentos xml, rdf e ncl, que representam o template, a descrição do template e as bases utilizadas pelo template, respectivamente. A leitura do documento xml é feita com a API aXT [Silva 2011], a qual cria um objeto Java com informações sobre os elementos do template. A leitura do rdf permite extrair a descrição do template e é realizada pela classe *TemplateDescription*. Esta descrição é exibida ao usuário com o intuito de ajudá-lo a compreender o objetivo do template. E, por fim, a leitura das bases NCL utilizadas pelo template é feita com a API aNaa [Dos Santos 2012].

Após realizar a leitura do template com a aXT e, consequentemente, ter informações sobre todos os elementos, um objeto para cada componente do vocabulário é criado com a classe *VocabularyComp*. Este objeto possui todas as informações sobre o componente como, por exemplo, se ele pode ou não ser escolhido pelo usuário ou se ele possui interfaces, cujas informações é mantida pelos objetos da classe *Interface*. Através da classe *Picture*, que estende um *JComponent*, a representação gráfica do componente é criada, ou seja, o tamanho, posição e cor do retângulo, caso seja um template com layout, são definidos.

O próximo passo é determinar as telas do template, para isso utiliza-se a classe *DiscoverScreens*, que analisa todos os elos do template a fim de descobrir suas telas. A primeira tela é composta pelos componentes que estão relacionados às portas do documento e, a partir destes, as informações de cada elo definem se uma nova tela deve ser gerada. Os objetos da classe *MyLink* armazenam as informações necessárias dos elos para determinar se novas telas devem ser geradas.

Depois de determinadas todas as telas do template, cria-se a representação gráfica de cada uma utilizando-se a classe *Screen*. Esta classe também estende um *JComponent*, o qual recebe os objetos *JComponent* criados por *Picture*, relativos aos componentes que formam a referida tela.

Por fim, para exibir a representação final do template com todas suas telas, a classe *TempScreen*, que estende um *JPanel*, organiza todas as telas, lado a lado, separadas por uma seta, indicando a possível ordem de exibição das telas.

A Figura 67 apresenta o diagrama de classes do pacote *template*.

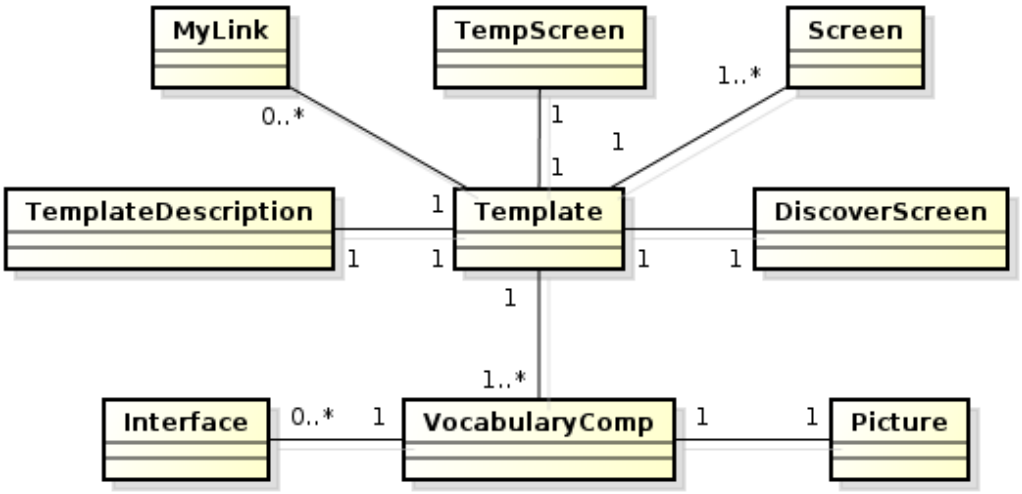


Figura 67: Diagrama de classes do pacote *template*.

PACOTE FILLTEMPLATE

O pacote *fillTemplate* é responsável por oferecer a interface de preenchimento do template. Ou seja, ele exibe uma lista com todas as telas do template escolhido, assim como os botões para duplicar e remover telas. Ao selecionar uma das telas, a tabela correspondente à tela selecionada é exibida para que o usuário possa preenchê-la, escolhendo as mídias para cada componente da tela.

Duas classes são responsável por oferecer a interface de preenchimento, uma para templates com layout, *FillTemplate*, e outra para templates sem layout, *FillTemplateNoLayout*. A diferença entre elas está na tabela de preenchimento que é exibida. Em um template com layout é possível, muitas vezes, duplicar telas, já em um template sem layout isso não é possível, o que ocorre é a duplicação de componentes. Logo, para um template sem layout existe apenas uma tabela, a qual exibe todos os componentes do template, inclusive aqueles duplicados pelo usuário.

A Figura 68 apresenta o diagrama de classes do pacote *fillTemplate*.

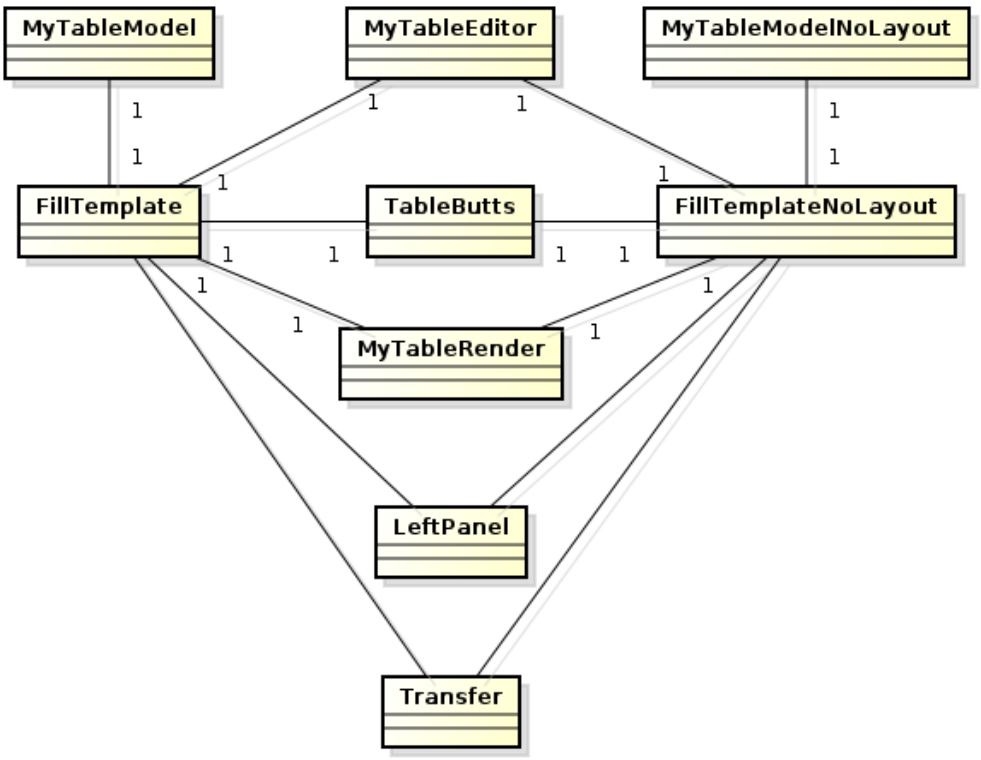


Figura 68: Diagrama de classes do pacote *fillTemplate*.

A classe *LeftPanel* é responsável pela exibição das diversas telas e pelas funcionalidades dos botões de duplicar e remover telas. *MyTableEditor*, *MyTableModel*, *MyTableModelNoLayout* e *MyTableRender* são responsáveis pelas tabelas de preenchimento. Já *TableButts* controla as ações do botões encontrados nas tabelas. A classe *Transfer* é

responsável por tratar as mídias que são arrastadas do repositório e liberadas na tabela de preenchimento do template, a mesma foi definida de acordo com as regras de definição de plugins.

PACOTE NCLDOCUMENT

O pacote *nclDocument* é composto apenas pela classe *NCLDocument*, a qual utiliza o processador de templates [Dos Santos 2012] para criar o documento NCL padrão. Esta classe também é responsável por substituir as importações das bases no documento final pela definição das próprias bases, uma vez que a API aNaa não trata importações nos documentos NCL.

PACOTE PROCESSOR

Os pacotes *processor.base* e *processor.documents*, desenvolvido em [Dos Santos 2012], possuem as classes responsáveis por transformar um documento NCL que faz referência a templates em um documento NCL padrão, próprio para rodar na máquina virtual.

PACOTE COMMON

O *common* é composto por seis classes, as quais fornecem métodos auxiliares que são utilizados por outras classes do plugin. As classes são: *Constantes*, *Languages*, *MyButton*, *MySpinnerModel*, *RendererList* e *TemplateException*.

A classe *Constantes* implementa alguns métodos e constantes utilizados em diversas outras classes. *Languages* é responsável por exibir os textos utilizados pelo plugin no idioma que está sendo utilizado pelo NEXT. As classes *MyButton* e *MySpinnerModel* estendem, respectivamente, as classes *JButton* e *AbstractSpinnerModel* do Java para atender às necessidades do plugin, assim como a classe *RendererList*, que implementa a classe *ListCellRenderer* do Java.

A classe *TemplateException* estende *Exception* com o objetivo de passar informações sobre exceções e/ou erros que possam ocorrer para as classes superiores, para que essas possam tratá-las ou informar ao usuário sobre o problema ocorrido.

ANEXO D – IMPLEMENTAÇÃO DO PLUGIN DE CONECTORES

Este anexo descreve, com maiores detalhes, a implementação de cada um dos pacotes do plugin de conectores.

PACOTE MYPLUGIN

O pacote *myPlugin* é constituído pelas classes obrigatórias, *StartPlugin* e *PluginInfo*, e pelas classes *Main* e *Connector*. A classe *Main* permite o uso do plugin como uma aplicação *stand alone*, sendo responsável pela inicialização da mesma, e, no caso de ser utilizada como plugin, a classe *StartPlugin* inicia o mesmo.

StartPlugin é responsável por exibir e atualizar a árvore de conectores, assim como os botões para criação e remoção destes. Quando o usuário cria um novo conector, ou seleciona algum na árvore, a representação gráfica do mesmo é exibida na área central do plugin. Esta representação gráfica é criada pela classe *Connector*.

Connector exibe todos os papéis pré-definidos pela linguagem NCL e permite que o usuário arraste-os e libere-os sobre o desenho do conector, desta forma, o conector vai sendo definido. Além disso, esta classe também é responsável por exibir mensagens de ajuda ao usuário durante a construção do conector.

A Figura 69 apresenta o diagrama de classes do pacote *myPlugin*.

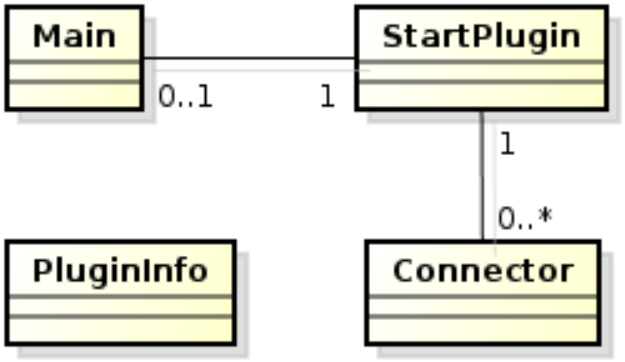


Figura 69: Diagrama de classes do pacote *myPlugin* do plugin de conectores.

PACOTE CONDITION

O pacote *condition* é constituído pelas classes *CondPanel*, *CondRole* e *CompoundCond*. *CondPanel* estende um *JPanel* e controla as opções que o usuário possui para criar um papel de condição. Ele exibe os papéis de condição criados no conector e oferece botões para criação/edição de condições compostas.

O *CondRole* possui as informações sobre um papel de condição, como, por exemplo, quais opções devem ser exibidas em sua janela de edição e que parâmetros são referenciados pelo papel, caso algum seja.

CompoundCond é responsável pelas informações referentes a uma condição composta. A classe conhece todos os papéis simples e/ou compostos que o formam, assim como o operador utilizado pela composição, AND ou OR.

PACOTE ACTION

O pacote *action* é constituído pelas classes *ActPanel*, *ActRole* e *CompoundAct*. *ActPanel* estende um *JPanel* e controla as opções que o usuário possui para criar um papel de ação. Ele exibe os papéis de ação criados no conector e oferece botões para criação/edição de ações compostas.

O *ActRole* possui as informações sobre um papel de ação, como, por exemplo, quais opções devem ser exibidas em sua janela de edição e que parâmetros são referenciados pelo papel, caso algum seja.

CompoundAct é responsável pelas informações referentes a uma ação composta. A classe conhece todos os papéis simples e/ou compostos que o formam, assim como o operador utilizado pela composição, SEQ ou PAR.

PACOTE PARAMETER

O pacote *parameter* é constituído pelas classes *ParamPanel* e *Param*. A primeira classe estende um *JPanel* que exibe os parâmetros do conector e permite que o usuário adicione novos parâmetros e, a segunda, representa o próprio parâmetro. Cada representação criada por *Param* possui um botão que possibilita a exclusão do próprio parâmetro.

PACOTE COMMON

O *common* é composto por quatro classes, as quais fornecem métodos auxiliares que são utilizados por outras classes do plugin. As classes são: *Constantes*, *Languages*, *MyButton*, e *RoleButton*. *Constantes* implementa alguns métodos e constantes utilizados em diversas outras classes. *Languages* é responsável por exibir os textos utilizados pelo plugin no idioma que está sendo utilizado pelo NEXT. *MyButton* e *RoleButton* estendem a classe *JButton* do Java, sendo a primeira utilizada para criar os botões de ação exibidos pela classe *StartPlugin*, e, a segunda, utilizada para representar os papéis pré-definidos, utilizados pela classe *Connector*.

ANEXO E – IMPLEMENTAÇÃO DO PLUGIN DE VISÃO DE LEIAUTE

Este anexo descreve, com maiores detalhes, a implementação de cada um dos pacotes do plugin de visão de leiaute.

PACOTE MYPLUGIN

O pacote *myPlugin* é constituído pelas classes: *StartPlugin*, *PluginInfo*, *Main*, *Region*, *RegionBase*, *Descriptor* e *DrawPanel*. A classe *Main* permite o uso do plugin como uma aplicação *stand alone*, sendo responsável pela inicialização da mesma, e, no caso de ser utilizada como plugin, a classe *StartPlugin* inicia o mesmo. *PluginInfo* possui as informações necessárias que todo plugin deve fornecer.

A classe *StartPlugin* cria a interface do plugin, exibe as árvores de bases de regiões e de descritores e implementa as funcionalidades dos botões para criação, remoção e edição das bases de regiões e de descritores, sendo também responsável pela exibição do código destas bases.

A classe *Region* cria a representação de cada região criada e é responsável por exibir a janela de edição dos atributos do elemento <region>. Sua representação gráfica muda de acordo com os valores atribuídos a seus atributos. Esta classe também permite a utilização do mouse, através da implementação dos métodos *MouseListener* e *MouseMotionListener* do Java, para definir os atributos de tamanho e localização da região.

Já a classe *RegionBase* possui informações sobre quais são suas regiões filhas e as exibe para o usuário através de um objeto da classe *DrawPanel*, a qual estende um *JPanel* que é composto pelas regiões da base.

Descriptor é responsável por exibir a janela de edição dos descritores, a qual apresenta ao usuário todas opções de atributos e parâmetros que um descritor pode ter. O elemento <descriptor> não possui uma representação gráfica e sua janela é exibida quando o mesmo é selecionado na árvore de descritores.

A Figura 70 apresenta o diagrama de classes do pacote *myPlugin*.

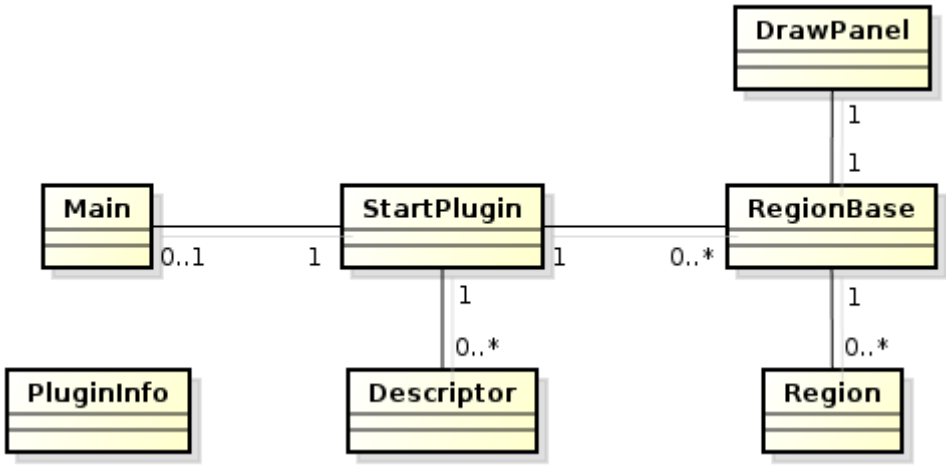


Figura 70: Diagrama de classes do pacote *myPlugin* do plugin de visão de leiaute.

PACOTE COMMON

O *common* é composto por classes que fornecem métodos auxiliares a outras classes do plugin. A classe *Constantes* possui métodos e constantes utilizados por outras classes. *Languages* é responsável por exibir os textos utilizados pelo plugin no idioma que está sendo utilizado pelo NEXT. As classes *MyButton* e *MySpinnerModel* estendem, respectivamente, as classes *JButton* e *AbstractSpinnerModel* do Java para atender às necessidades do plugin.

ANEXO F – IMPLEMENTAÇÃO DO PLUGIN DE VISÃO ESTRUTURAL

Este anexo descreve, com maiores detalhes, a implementação de cada um dos pacotes do plugin de visão estrutural.

PACOTE MYPLUGIN

O pacote *myPlugin* é constituído pelas classes: *StartPlugin*, *PluginInfo*, *Context*, *Port*, *Switch*, *SwtPort* e *Rule*. A classe *StartPlugin* é responsável pela inicialização do plugin, exibindo uma árvore com todos os elementos do corpo do documento, assim como a representação gráfica do contexto selecionado na mesma. *PluginInfo* possui as informações necessárias que todo plugin deve fornecer.

A classe *Context* estende um *JComponent* para representar o elemento <context>. O objeto desta classe possui informações sobre todos seus elementos filhos e utiliza um objeto da classe *DrawPanel*, que será explicado no pacote *common*, para exibir a representação gráfica dos filhos. Ele também é responsável por oferecer os botões de criação de novos elementos no contexto e por exibir o código NCL do mesmo.

A classe *Switch* estende a classe *Context*, assim ela também mantém informações sobre seus filhos e possui um 'mainPanel' para exibi-los. Ela também é responsável pela exibição da janela de edição dos atributos do switch. Já a classe *Rule* permite a criação de regras para serem associadas aos elementos de mídia do switch.

As classes *Port* e *SwtPort* representam as portas de um contexto e de um switch, respectivamente, e são responsáveis pela exibição gráfica destes elementos, assim como por suas janelas de edição.

A Figura 71 apresenta o diagrama de classes do pacote *myPlugin*.

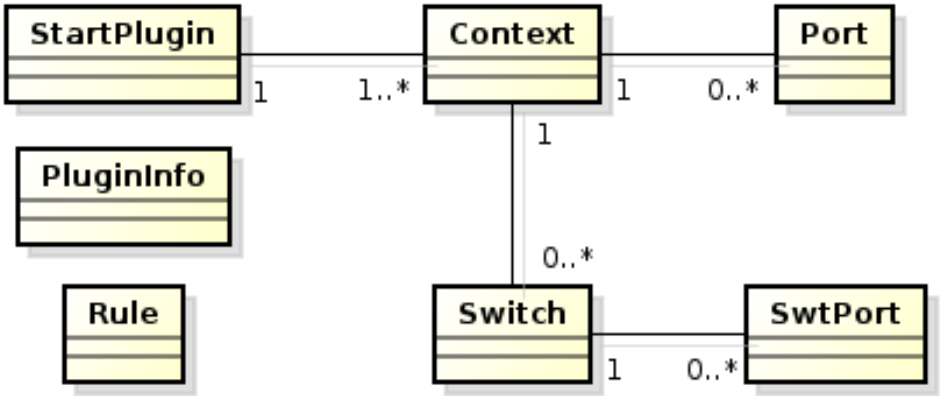


Figura 71: Diagrama de classes do pacote *myPlugin* do plugin de visão estrutural.

PACOTE MEDIA

O pacote *media* é utilizado para representar os elementos de mídia do documento. Ele é constituído pelas classes *Media* e *Interface*. *Media* estende um *JComponent* para representar o elemento graficamente. Ela também é responsável por exibir a janela de edição dos atributos da mídia e por manter informações referentes a outros elementos que fazem referência a mesma, caso existam. Já a classe *Interface* é responsável por exibir a janela de edição dos atributos das interfaces da mídia, caso existam.

PACOTE LINK

O pacote *link* representa os elos do documento. Ele é constituído pelas classes *Link* e *Bind*. A classe *Link* estende um *JComponent* para representar o elemento graficamente. Ela também é responsável por manter informações sobre seus elementos <bind> e janela de edição do elo. A classe *Bind* exibe a janela de edição dos atributos de um <bind> e mantém informações sobre os parâmetros do mesmo, caso existam. A representação gráfica do <bind> é feita pela classe *DrawPanel*.

PACOTE COMMON

O *common* é composto por classes que fornecem métodos auxiliares a outras classes do plugin. A classe *Constantes* possui métodos e constantes utilizados por outras classes. *Languages* é responsável por exibir os textos utilizados pelo plugin no idioma que está sendo utilizado pelo NEXT. As classes *MyButton* e *MySpinnerModel* estendem, respectivamente, as classes *JButton* e *AbstractSpinnerModel* do Java para atender às necessidades do plugin.

A classe *DrawPanel* estende um *JPanel* e é responsável por exibir os elementos filhos de um contexto ou de um switch. Ela implementa as classes *MouseListener* e *MouseMotionListener* do Java a fim de permitir que o usuário arraste e organize seus elementos. Esta classe também é responsável por exibir graficamente os elementos <bind>, já que a representação destes é um desenho no próprio *JPanel*. Entretanto, o objeto do tipo *Bind* é avisado quando o mesmo é selecionado pelo usuário, para que sua janela de edição seja exibida. *DrawPanel* também desenha as linhas que relacionam as portas de contextos e de switches a seus componentes.

A classe *Transfer* é responsável por tratar as mídias que são arrastadas do repositório e liberadas no *DrawPanel*, criando uma nova mídia no contexto/switch.