UNIVERSIDADE FEDERAL FLUMINENSE
ESCOLA DE ENGENHARIA
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES

JEAN RIBEIRO DAMASCENO

EDITEC: EDITOR GRÁFICO DE TEMPLATES DE COMPOSIÇÃO PARA FACILITAR A AUTORIA DE APLICAÇÕES NCL PARA TV DIGITAL INTERATIVA

NITERÓI
2010

JEAN RIBEIRO DAMASCENO

EDITEC: EDITOR GRÁFICO DE TEMPLATES DE COMPOSIÇÃO PARA FACILITAR A AUTORIA DE APLICAÇÕES NCL PARA TV DIGITAL INTERATIVA

Dissertação apresentada ao Curso de Mestrado em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre. Área de Concentração: Sistemas de Telecomunicações.

Orientadora: Profª. Dra. Débora Christina Muchaluat Saade

Niterói

2010

JEAN RIBEIRO DAMASCENO

EDITEC: EDITOR GRÁFICO DE TEMPLATES DE COMPOSIÇÃO PARA FACILITAR A AUTORIA DE APLICAÇÕES NCL PARA TV DIGITAL INTERATIVA

Dissertação apresentada ao Curso de Mestrado em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre. Área de Concentração: Sistemas de Telecomunicações.

Aprovada em 23 de setembro de 2010.

BANCA EXAMINADORA

_____

Profa. Dra. Débora Christina Muchaluat Saade, D.Sc. / IC-UFF
(Orientadora)

_____

Profa. Dra. Cláudia Maria Lima Werner, D.Sc. / COPPE-UFRJ

_____

Prof. Dr. Esteban Walter Gonzalez Clua, D.Sc. / IC-UFF

_____

Prof. Dr. Murilo Bresciani de Carvalho, D.Sc. / TET-UFF

Niterói

2010

Esta dissertação é dedicada

primeiramente e sempre a Deus por tudo;

e aos meus pais José Damasceno Filho e Asenath Ribeiro Damasceno,

pelo amor, apoio e incentivo constantes.

Agradecimentos

Agradeço a todos que de alguma forma contribuíram para a realização deste trabalho.

Aos meus familiares, irmãs, avós, tias, tios etc., em especial aos meus tios Roger e Judith, pelos pensamentos positivos e incentivadores.

A todos os que integram ou já integraram a equipe do Laboratório MídiaCom e aos demais amigos de faculdade (UFF), pelos longos anos de convívio, em especial ao Joel Santos e ao Felipe Rolim pela ajuda neste trabalho e ao Diego Passos, por perder seu precioso tempo inúmeras vezes para me ajudar nas mais diversas tarefas.

A todos os professores e funcionários do Departamento de Engenharia de Telecomunicações, pelos ensinamentos e pela ajuda.

A profª Dra. Débora Christina Muchaluat Saade, pelo seu apoio e paciência em me orientar e ser em todos os momentos uma pessoa agradável e compreensível.

A UFF, CAPES, CNPq e FAPERJ, pelos auxílios concedidos.

Mais uma vez e sempre a Deus por tudo e também aos meus Pais.

Resumo

Este trabalho apresenta o EDITEC, um editor gráfico de *templates* de composição hipermídia baseado na linguagem XTemplate 3.0. O EDITEC foi projetado para a criação de *templates* usando uma abordagem gráfica visual de fácil uso e aprendizado. Ele fornece diversas opções para representar estruturas de iteração e uma interface gráfica para a criação de expressões XPath básicas. O sistema provê um ambiente amigável com múltiplas visões, dando ao autor um completo controle do *template* de composição durante o processo de autoria. *Templates* de composição podem ser usados em programas NCL para embutir semântica espaço-temporal em contextos NCL e facilitar a produção de conteúdo interativo para o Sistema Brasileiro de TV Digital. A produção de aplicações interativas não é um processo simples. As ferramentas de autoria ajudam a reduzir o esforço necessário para o desenvolvimento dessas aplicações.

Palavras-chave

EDITEC, NCL, XTemplate, XPath, autoria multimídia, editor gráfico, *templates* de composição.

Abstract

This work presents EDITEC, a graphical editor for hypermedia composite templates based on the XTemplate 3.0 language. EDITEC was designed for the creation of templates using a visual user-friendly graphical approach. It provides several options for representing iteration structures and a graphical interface for creating basic XPath expressions. The system provides a friendly environment with multiple views, giving the user a complete control of the composite template during the authoring process. Composite templates can be used in NCL programs for embedding spatio-temporal semantics into NCL contexts and making the production of interactive content easier for the Brazilian Digital TV System. The production of interactive applications is not a simple task. Authoring tools help reducing the effort required to develop those applications.

Keywords

EDITEC, NCL, XTemplate, XPath, multimedia authoring, graphical editor, composite templates.

Sumário

Lista de Acrônimos

API: Application Programming Interface

BST-OFDM: Band Segmented Transmission - Orthogonal Frequency Division Multiplexing

DOM: Document Object Model

GUI: Graphical User Interface

HE-AAC: High Efficiency - Advanced Audio Coding

IPTV: Internet Protocol Television

ISDB-T: Terrestrial Integrated Services Digital Broadcasting

ITU: International Telecommunication Union

MPEG: Moving Picture Experts Group

NCL: Nested Context Language

NCM: Nested Context Model

SBTVD: Sistema Brasileiro de Televisão Digital

SMIL: Synchronized Multimedia Integration Language

TVDI: TV Digital Interativa

URI: Uniform Resource Identifier

W3C: World Wide Web Consortium

XML: eXtensible Markup Language

XMT: eXtensible MPEG-4 Textual

XPATH: XML Path Language

XSLT: eXtensible Stylesheet Language Transformations

Lista de Figuras

Lista de Listagens

Lista de Tabelas

Capítulo 1

Introdução

Baseado no padrão japonês ISDB-T (*Terrestrial Integrated Services Digital Broadcasting*) (ISDB-T, 1998), o governo brasileiro definiu, recentemente, o seu sistema de televisão digital interativa. Ele é chamado de SBTVD (Sistema Brasileiro de Televisão Digital) (SBTVD, 2010), e adiciona inovações desenvolvidas por pesquisadores de universidades brasileiras. Além de proporcionar melhor qualidade de áudio e vídeo, a TV digital também proporciona interação entre o telespectador e a programação de TV.

Com a TV interativa, o usuário deixa de ter um papel passivo de telespectador e passa a ter um papel ativo, permitindo-lhe atuar com o programa transmitido, tomando ações que podem interferir na forma e no conteúdo exibidos na televisão. Abre-se um leque de possibilidades, tais como compras através da TV, acesso à Internet, aplicações bancárias na TV, exibição de publicidade de forma personalizada, de acordo com preferências e características do usuário, dentre outras.

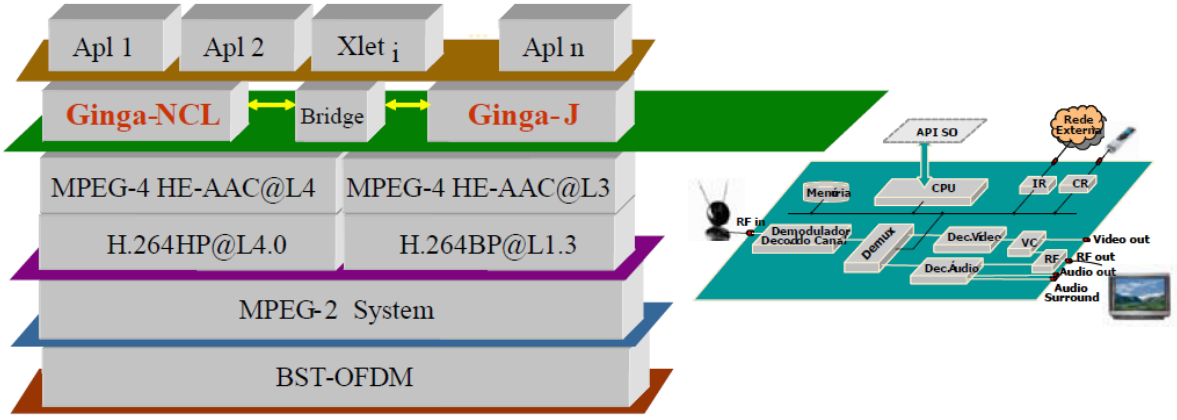


Figura 1: Arquitetura de referência do Sistema Brasileiro de TV Digital.

Fonte (Soares e Castro, 2008)

Semelhante a projetos de edificações, a melhor forma de lidar com um sistema complexo, como é o caso de um sistema de TV digital interativa, é através da definição de sua arquitetura. Uma arquitetura mostra os principais elementos de um sistema, explicitando suas interações e escondendo os detalhes menos importantes sob o ponto de vista adotado. A Figura 1 apresenta a arquitetura do Sistema Brasileiro de TV Digital, onde são exibidas as

camadas e as tecnologias utilizadas no padrão, incluindo seu *middleware* próprio, de nome Ginga (ABNT, 2007; Soares et al., 2007). A ideia central da arquitetura em camadas é cada uma oferecer serviços para a camada superior e usar os serviços oferecidos pela inferior.

As camadas são caracterizadas da seguinte forma:

- **Serviços, aplicações e conteúdo:** camada responsável pela captura e formatação dos sinais de áudio e vídeo, bem como a implementação de serviços interativos na forma de software para serem executados em uma ou mais entidades de hardware (Alencar, 2007).
- ***Middleware*:** camada de software que oferece um serviço padronizado para as aplicações, escondendo as peculiaridades e heterogeneidades das camadas inferiores. O *middleware* permite que os aplicativos gerados pelas emissoras sejam compatíveis com todas as plataformas de recepção desenvolvidas para o padrão de TV digital, facilitando, assim, a portabilidade das aplicações (ABNT, 2007; Soares et al., 2007).
- **Compressão:** camada responsável pela remoção de redundâncias nos sinais de áudio e vídeo, reduzindo assim a taxa de bits necessária para transmitir essas informações (Alencar, 2007). No receptor, essa camada decodifica os fluxos de áudio e vídeo recebidos. O sistema brasileiro adotou o padrão MPEG-4 HE AAC para compressão de áudio e o padrão H.264 para compressão de vídeo.
- **Multiplexação:** camada responsável por gerar um único fluxo de transporte contendo vídeo, áudio e aplicações dos diversos programas a serem transmitidos, baseada no padrão MPEG-2 Systems (Alencar, 2007).
- **Transmissão/Recepção:** também denominada de camada física, é a camada responsável por levar as informações digitais da emissora até a casa do telespectador (Mendes, 2007). O sistema brasileiro adotou a mesma tecnologia de transmissão utilizada no sistema japonês de TV digital, baseada em BST-OFDM.

No contexto de TV digital interativa, a camada de *middleware* recebe destaque especial, pois é a camada onde as aplicações de TV interativa podem ser executadas. No padrão brasileiro de TV digital, a principal inovação foi justamente nessa camada, com a adoção do *middleware* Ginga.

Nos *middlewares* para TV digital, as aplicações podem ser divididas, dependendo da forma em que elas são escritas, em duas categorias: procedural (imperativa) e declarativa. No

sistema brasileiro, as aplicações procedurais são escritas usando a linguagem Java e as aplicações declarativas são escritas usando a linguagem NCL (*Nested Context Language*) (Soares e Barbosa, 2009), baseada em XML (XML, 2008). Para isso, o *middleware* Ginga é subdividido em dois subsistemas principais interligados, Ginga-J (Filho et al., 2007), para aplicações procedurais Java, e Ginga-NCL (ABNT, 2007; Soares et al., 2007), para aplicações declarativas NCL. Dependendo das funcionalidades requeridas no projeto de cada aplicação, um paradigma de programação é mais adequado que o outro.

1.1 Motivação

Como mencionado anteriormente, no Sistema Brasileiro de TV Digital, o ambiente declarativo Ginga-NCL utiliza a linguagem NCL para o desenvolvimento de aplicações interativas. Ginga-NCL também foi adotado como padrão internacional ITU para serviços de IPTV na recomendação H.761 (ITU, 2009). Com a difusão do uso da linguagem, o número de aplicações NCL tende a crescer, sendo importante o desenvolvimento de ferramentas que auxiliem em seu desenvolvimento. Com o uso de NCL para TV digital, os telespectadores podem ter acesso a uma programação interativa e de maior qualidade. Exemplos de aplicações são programas de governo eletrônico (*t-Government*), de saúde (*t-Health*), além de outros programas não-lineares, onde é possível que haja uma continuação escolhida pelo telespectador, ou mesmo um *quiz* realizado em programas de auditório.

A autoria de aplicações declarativas mais complexas para TV digital que possuem muitos objetos de mídia, a necessidade de se definir um número grande de relacionamentos entre eles, bem como a especificação de características de apresentação, acaba introduzindo mais dificuldade em sua autoria, pois os documentos NCL ficam maiores, tornando o surgimento de erros mais frequente. Neste cenário, o reúso de especificações genéricas de programas e o oferecimento de editores gráficos facilitam bastante a autoria, possibilitando aos autores abstrair o máximo possível da complexidade de programar.

A linguagem XTemplate 3.0 (Santos, 2009) apresenta-se como uma forma de se definir tais estruturas genéricas através do conceito de *templates* de composição, permitindo o reúso de especificações estruturais em documentos distintos, mas que têm características estruturais semelhantes.

Templates de composição descrevem, de uma forma genérica, todo o conteúdo de uma composição, definindo componentes genéricos e os relacionamentos entre eles. O documento

cuja composição utiliza um *template* fica responsável apenas por indicar os elementos que serão utilizados. Depois que o documento que referencia o *template* é processado, esses elementos recebem a mesma lógica definida para os componentes genéricos do *template*. Assim, o usuário de um *template* só precisa especificar as mídias que serão usadas no documento. Os leiautes espaciais e as relações temporais bem como as características de interatividade dos documentos são automaticamente gerados pelo uso do *template*, os usuários não precisam se preocupar com essas definições, tornando a criação de documentos NCL uma tarefa mais fácil. Com a linguagem XTemplate, pode ser criado um conjunto de *templates* de composição com diferentes semânticas, não se limitando a apenas a um conjunto pré-definido, como é o caso, por exemplo, da linguagem SMIL (*Synchronized Multimedia Integration Language*) (SMIL, 2005) que oferece apenas três tipos de composições, sequencial, paralelo e exclusivo.

Um problema que persiste é que a criação de *templates* usando a abordagem textual através da linguagem XTemplate, baseada em XML, não é uma tarefa tão simples para usuários inexperientes. O usuário precisa dominar bem a sintaxe e semântica da linguagem XTemplate e ter uma boa capacidade de abstração, além da dificuldade no entendimento da estrutura do documento conforme a quantidade de informação cresce. A edição textual permite alta flexibilidade e grande poder de edição, mas por outro lado, consome muito tempo. Para certos usuários, o tempo gasto especificando o documento na forma puramente textual torna o processo de autoria trabalhoso.

Assim, a disponibilidade de uma ferramenta gráfica para a autoria de *templates* é desejável. Em um editor gráfico, o uso de ícones para rotular as ações de edição permite que o autor estabeleça uma associação entre as imagens de componentes da interface gráfica com eventos de edição, ao invés de guardar nomes de elementos XML e comandos da linguagem textual. Entretanto, o design de tal ferramenta não é uma tarefa simples. O desenvolvimento de uma ferramenta que seja simples e fácil de usar e, ao mesmo tempo, seja flexível o bastante para permitir a construção de diversos *templates* é um desafio.

Um editor gráfico também oferece facilidade de reúso. Pequenas partes dos documentos podem ser facilmente identificadas (através de figuras) e reaproveitadas na elaboração de um novo documento.

Alguns requisitos, levantados a partir da análise de diversos trabalhos relacionados e das características da linguagem XTemplate, são desejáveis em uma ferramenta gráfica de edição de *templates* de composição, tais como:

- várias visões do documento – a apresentação de um documento multimídia em diferentes visões, como por exemplo as visões estrutural e temporal, fornece ao usuário uma maior intuição no desenvolvimento de suas aplicações, dando a ele uma visão global prévia do documento, na qual o autor pode ter um retorno visual mais imediato durante a edição;
- definição de componentes genéricos – um componente genérico é utilizado para representar um conjunto de elementos. Quando é definida uma lógica para esse componente genérico, todos os elementos que ele representa também recebem essa mesma lógica;
- definição de estruturas de repetição – para atribuir uma lógica de um componente genérico a um conjunto de elementos, faz-se necessária a definição de estruturas de repetição, exigindo uma abordagem gráfica para representar essas possíveis estruturas;
- descrição semântica do *template* – para facilitar o uso do *template* em diferentes documentos NCL, é interessante oferecer uma descrição semântica do *template*, indicando sua função principal e o papel de cada um de seus componentes genéricos;
- definição de expressões XPath – para a construção de um *template* de composição é necessário fazer uso de expressões XPath (XPath, 1999). Deste modo, é importante ter opções para construir essas expressões;
- portabilidade em diferentes plataformas – a portabilidade em diferentes plataformas permite que o usuário utilize o editor em diferentes plataformas de hardware e sistemas operacionais;
- uma interface flexível com capacidade de realizar edições de estruturas genéricas mais complexas – depois de levantados todos os requisitos para a construção de *templates* de composição NCL, fica evidente a importância de oferecer uma interface flexível que ofereça todas essas facilidades e permita a criação de *templates* de forma interativa.

1.2 Objetivos

Considerando as informações anteriores e visando a atender os requisitos desejáveis de uma ferramenta gráfica de edição de *templates* de composição, esta dissertação propõe o

EDITEC (Damasceno et al., 2010), um editor gráfico de *templates* de composição NCL baseado na linguagem XTemplate 3.0, que permite a criação de *templates* de forma interativa. O objetivo do EDITEC é auxiliar a criação de *templates* principalmente por usuários mais leigos em XTemplate. O usuário tem menos esforço usando o sistema do que teria se os *templates* fossem construídos diretamente de forma textual.

O objetivo é oferecer um ambiente gráfico amigável com múltiplas visões/janelas que ajudam os autores a criar *templates* eficientemente. Diversas visões integradas facilitam uma melhor visão global do documento, dando ao autor um completo controle do documento durante a edição.

Um objetivo mais pontual deste trabalho é o de detalhar o modelo gráfico criado para representar estruturas de iteração, que permite a criação de diversos *templates*, de forma fácil e interativa, com as mais variadas semânticas espaço-temporais. Esse modelo, desenvolvido especialmente para o EDITEC, é usado para criar, de forma gráfica, diversos tipos de relacionamentos entre componentes de um *template*. O EDITEC analisa a estrutura gráfica do *template* com as opções escolhidas, processa-a e gera o código XTemplate correspondente em XML.

Um outro objetivo importante requerido na ferramenta de edição gráfica de *templates* de composição é a de proporcionar a construção de uma grande variedade de *templates* sem que o autor precise saber ou definir expressões XPath de forma textual. Para isso, esta dissertação também apresenta uma interface gráfica amigável para criação de expressões XPath. Ela é usada no EDITEC, com objetivo de substituir a definição textual de expressões XPath básicas por construções interativas e intuitivas da lógica das expressões XPath através de interface gráfica.

1.3 Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma. O Capítulo 2 apresenta alguns trabalhos relacionados, onde diversas ferramentas de edição gráfica são analisadas com o objetivo de identificar requisitos importantes que um editor gráfico de *templates* de composição NCL possa vir a oferecer.

O Capítulo 3 aborda as linguagens NCL e XTemplate 3.0, que são as linguagens foco nesta dissertação. Os detalhes mais importantes dos principais elementos das duas linguagens

são exibidos e, posteriormente, é apresentado um exemplo de uso de um *template* de composição.

O Capítulo 4 apresenta o EDITEC, um editor gráfico de *templates* de composição, onde são exibidas as três visões disponíveis para edição, estrutural, de leiaute e textual. Além disso, o capítulo apresenta um modelo usado para representar graficamente estruturas genéricas com diferentes opções de iteração, uma interface gráfica desenvolvida para criar expressões XPath básicas e diversos exemplos de construção de *templates* no ambiente de autoria EDITEC.

O Capítulo 5 apresenta uma visão geral da implementação das principais partes do ambiente EDITEC, que é desenvolvido em linguagem Java. O capítulo também comenta algumas funcionalidades ainda não disponíveis na implementação atual que devem ser acrescentadas futuramente. Além disso, o capítulo também apresenta um estudo de usabilidade da ferramenta EDITEC, realizado com um grupo de treze usuários com nível de conhecimento variado em NCL e XTemplate, incluindo usuários avançados e iniciantes. O objetivo do estudo foi traçar, através de um questionário, indicadores de facilidade de uso do editor e verificar se ele facilita o entendimento dos conceitos da linguagem XTemplate.

O Capítulo 6 traz comparações com os principais trabalhos relacionados da área, apresenta as considerações finais, salientando as contribuições da dissertação, e propõe possíveis trabalhos futuros a serem realizados.

O Apêndice A exibe códigos de programas NCL e XTemplate de alguns exemplos que são apresentados durante o texto.

Capítulo 2

Trabalhos Relacionados

Este capítulo apresenta diversas ferramentas de edição gráfica, analisadas com o objetivo de levantar requisitos importantes para o desenvolvimento de uma ferramenta gráfica de edição de *templates* de composição que serve de auxílio na criação de aplicações interativas para TV digital. As principais funcionalidades de cada ferramenta são analisadas, ressaltando os pontos fortes e fracos a serem observados. Também é feita uma comparação entre as ferramentas, com o objetivo de resumir suas principais funcionalidades.

2.1 LimSee2

O LimSee2 (Deltour et al., 2005; LimSee2, 2010) é uma ferramenta de código aberto para autoria de documentos SMIL (SMIL, 2005), desenvolvida pelo *Institut National de Recherche en Informatique* - INRIA (INRIA, 2010), que oferece várias visões integradas (temporal, de leiaute, textual etc.). Ele provê duas visões mais aprimoradas (visões de leiaute e temporal) e duas visões mais simples (visões XML e textual). Seus conteúdos são automaticamente atualizados quando ocorrem mudanças no documento SMIL. A Figura 2 apresenta a interface gráfica do ambiente LimSee2.

Na Figura 2, as regiões 1 (um) e 3 (três) representam a visão XML, na qual o documento SMIL é organizado na forma de uma árvore. A visão XML é dividida em duas partes: a porção de cima representa a árvore XML do documento SMIL, chamada de “visão em árvore” ou “de estrutura”, e a porção de baixo representa os atributos do elemento selecionado atualmente, chamada de “visão de atributos”.

Na região 2 (dois), é exibida a visão mais sofisticada, a visão de leiaute. Ela produz imagens, fotos em movimento, estilo de texto, etc. A visão também simula a reprodução de uma apresentação, incluindo animações e diversos efeitos, tais como a transparência. Na região 4 (quatro), é apresentada a visão temporal, onde os usuários podem ajustar a sincronização de mídias em uma linha de tempo, através de movimentação e sincronização de caixas de objetos cujo comprimento está relacionado à sua duração. Além disso, nessa visão é

possível especificar um determinado instante de tempo para visualizar na visão de leiaute o estado da apresentação.

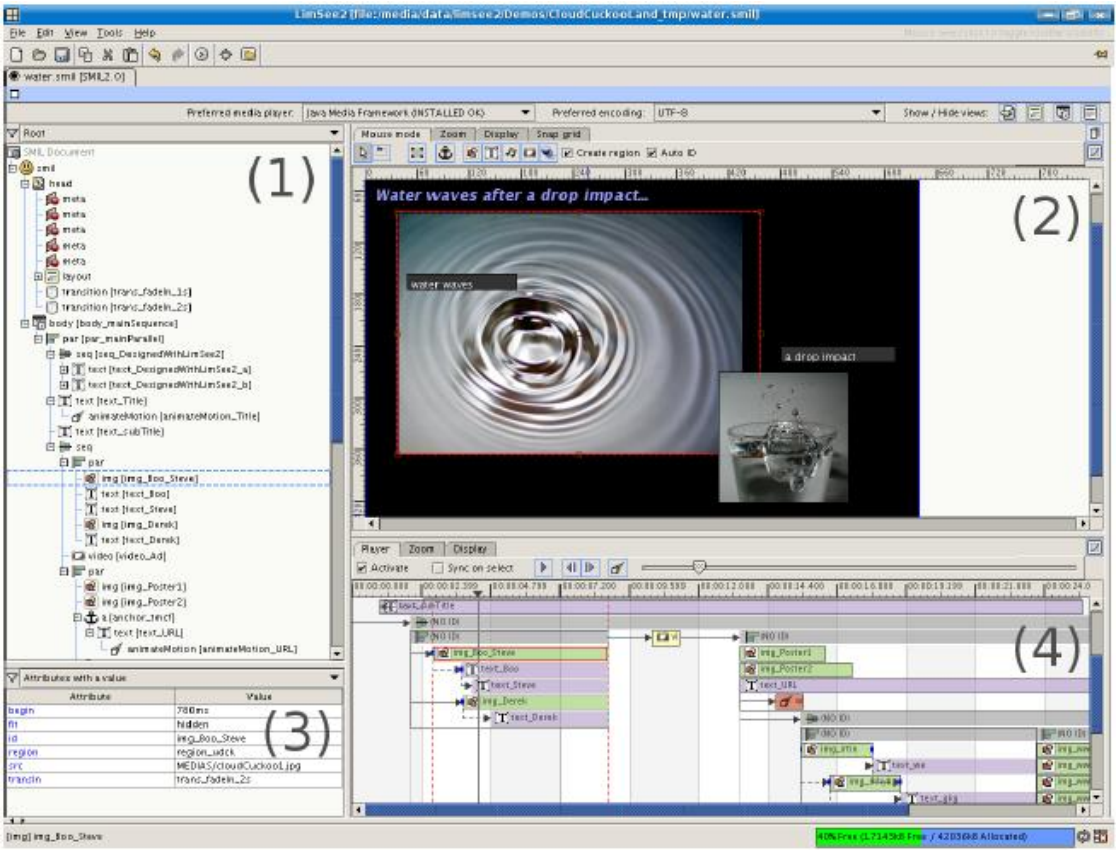


Figura 2: Ambiente de autoria LimSee2.

Fonte (Deltour e Roisin, 2006)

A visão textual fica escondida por padrão, porque é usada raramente, e somente por usuários mais experientes. Essa visão de código permite que usuários mais experientes controlem o código fonte da apresentação SMIL sem serem restringidos pelo alto nível da visão XML.

O LimSee2 privilegia a especificação da sincronização temporal multimídia à interatividade.

2.2 LimSee3

O LimSee3 (LimSee3, 2010) é uma ferramenta de autoria multimídia de código aberto, baseada em *templates* e desenvolvida em linguagem Java. Em (Deltour e Roisin, 2006), o modelo LimSee3 é proposto. Esse modelo usa *templates* para ajudar na autoria de documentos multimídia. O objetivo do LimSee3 é que usuários experientes interajam com

todas as características da plataforma e usuários básicos interajam com *templates* específicos e interfaces simplificadas. Ele define uma linguagem de autoria, independente das linguagens existentes, focando na estrutura lógica de um documento e não na semântica da linguagem alvo. Com essa abordagem, LimSee3 provê grande flexibilidade e reúso. O *template* criado com essa linguagem descreve uma hierarquia genérica entre seus componentes, sendo a sincronização temporal feita com o uso de elementos que fazem referência a composições SMIL. A Figura 3 apresenta o LimSee3.

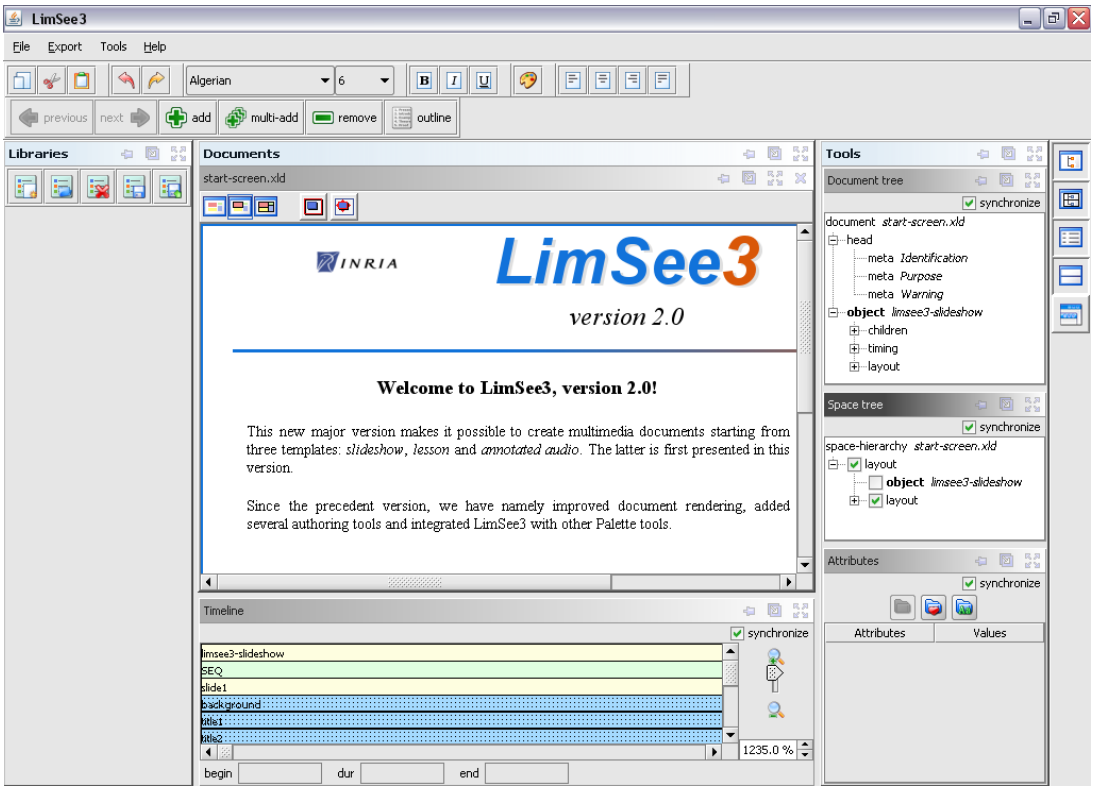


Figura 3: Ambiente de autoria LimSee3.

Embora os nomes possam sugerir versões diferentes de um mesmo editor, LimSee3 não é uma continuação direta ou uma nova versão do LimSee2. LimSee2 é um editor SMIL e, como tal, é dependente da linguagem base. LimSee3 objetiva autoria independente de linguagem, baseado em uma estruturação lógica de documentos multimídia. Ele pode produzir documentos baseados no padrão SMIL, mas seu modelo interno não é dependente do SMIL, sendo ele mais baseado na semântica dos objetos do que em sua real apresentação. Os objetos definem seus próprios comportamentos temporal, espacial e interativo. Um documento pode ser visto como uma árvore de objetos.

No LimSee3, um documento é criado a partir da instanciação de um *template*, em um tipo de aplicação guiada. Um *template* pode ser visto, no LimSee3, como um "documento com buracos" chamados de zonas e o processo de instanciação consiste no preenchimento dos

“buracos” com conteúdo de mídia, sendo possível, durante o processo de autoria de *templates*, a adição de frases do tipo “Por favor, adicione uma imagem aqui” (“*Please add image here*”) para ajudar os usuários no uso do *template*. É possível trancar partes do *template* para prevenir que usuários editem uma parte específica. Isso permite, por exemplo, guiar mais fortemente usuários inexperientes restringindo seu acesso a somente as partes do documento que fazem sentido para eles. LimSee3 não provê uma distinção clara entre o *template* e sua instância, já que o autor deve editar um *template* diretamente para preencher suas lacunas.

LimSee3 organiza mídias em bibliotecas de mídias. Depois de escolhido um *template*, o usuário pode, por exemplo, através de cliques e arrastes do mouse, mover suas mídias específicas das bibliotecas para as zonas definidas no *template*. Ele permite a manipulação, através de múltiplas visões, de todos os elementos definidos no modelo (documentos, objetos compostos, detalhes de tempo e leiaute, relações...).

A finalidade de uma biblioteca é fornecer uma visão uniforme de alguns tipos de conteúdo de mídia local e remoto. Bibliotecas não contêm os componentes, elas meramente os referenciam, assim um conteúdo apresentado em várias bibliotecas não é fisicamente duplicado e um conteúdo remoto não é baixado para armazenagem local.

2.3 Composer

O Composer (Guimarães, 2007) é uma ferramenta de autoria implementada em Java, que fornece várias visões integradas para a criação de documentos NCL. Tendo como objetivo a construção de programas audiovisuais interativos por usuários com pouco conhecimento da linguagem NCL. Ele possui licenciamento duplo, com licenças GPLv2 e comercial.

A ferramenta Composer permite ao usuário trabalhar com as visões estrutural, temporal, textual e de leiaute. Sempre que é feita alguma alteração em uma visão, ela é refletida nas demais visões. A Figura 4 apresenta as diversas visões em que o usuário pode trabalhar.

A visão estrutural abstrai as principais entidades (Soares e Barbosa, 2009) definidas em NCL. Nessa visão, o autor pode criar, editar e deletar objetos de mídia que compõem um documento, contextos (conjunto de objetos e seus relacionamentos) e elos (relacionamentos entre objetos). Na visão de leiaute, o autor pode criar, editar e remover regiões espaciais associadas com a exibição inicial dos objetos. A visão temporal abstrai os relacionamentos

temporais entre objetos de mídia, fornecendo uma solução gráfica a esses relacionamentos. Essa visão permite a representação e simulação de eventos imprevisíveis, como interações do telespectador e adaptações de conteúdo. Outra visão importante é a textual, que permite a edição do código NCL.

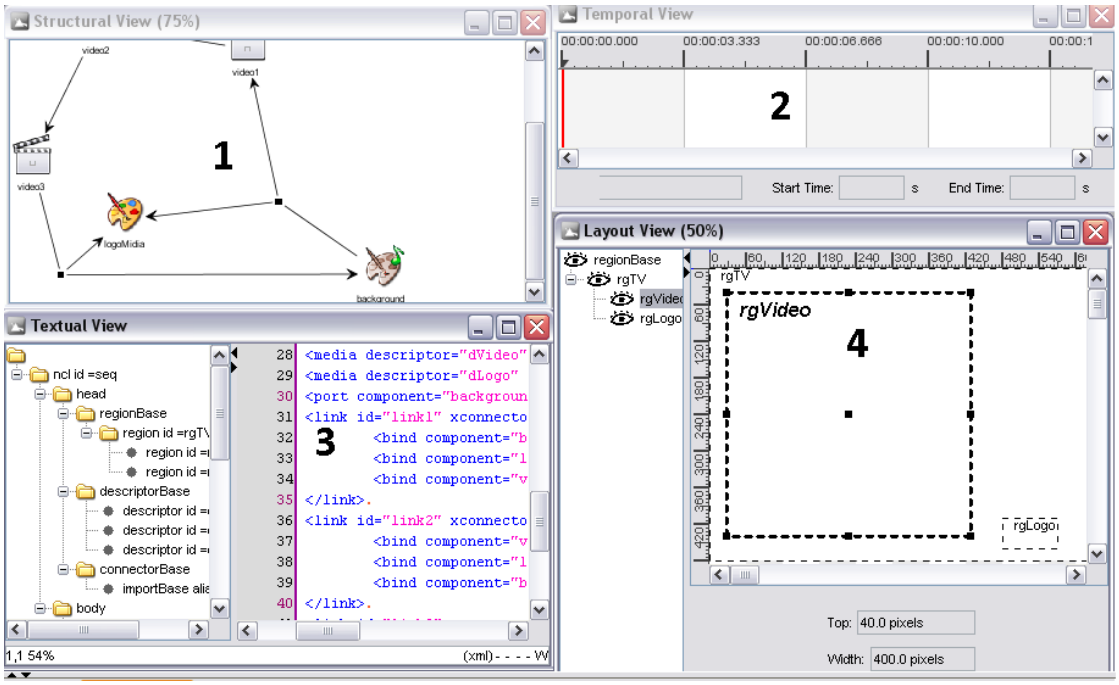


Figura 4: Ferramenta de autoria Composer.

O Composer utiliza o Ginga-NCL Emulator (Ginga-NCL, 2010), que é uma ferramenta disponibilizada para testes de aplicações NCL. Apesar do Ginga-NCL Emulator apresentar problemas de fidelidade na apresentação de documentos NCL, pois não oferece suporte à transparência, efeitos de transição, entre outros, a integração do Composer com essa ferramenta facilita a depuração da aplicação, aumentando o poder do editor em desenvolver aplicações com um maior nível de interatividade. O Composer não oferece o uso de *templates* para autoria.

Um problema no Composer é que há grande necessidade de salvar o projeto em vários arquivos de backup, pois o corrompimento de arquivos é frequente.

2.4 Lamp

O Lamp (Gaggi e Celentano, 2006; Celentano e Gaggi, 2003) é um ambiente de autoria que contém um editor visual para autoria de apresentações, um simulador de execução para testar o comportamento da apresentação e um formatador para exibir e executar a

apresentação completa. Ele permite a definição de *templates* de apresentação que podem ser instanciados com diferentes dados.

O principal componente do ambiente de autoria é o editor visual baseado em uma notação gráfica, na qual os nós são objetos de mídia e as arestas são relacionamentos de sincronização entre eles. Todos os aspectos da apresentação são definidos pela manipulação gráfica. O editor visual também provê facilidades para desenhar o leiaute da tela. O ambiente de autoria visual é desenvolvido em linguagem Java. O editor visual possui duas visões: uma para definir os canais e o leiaute da apresentação e outra para definir o comportamento temporal. Os canais podem ser de dois tipos: regiões, por exemplo, áreas da tela, e canais de áudio. As Figuras 5 e 6 apresentam as duas visões do editor visual.

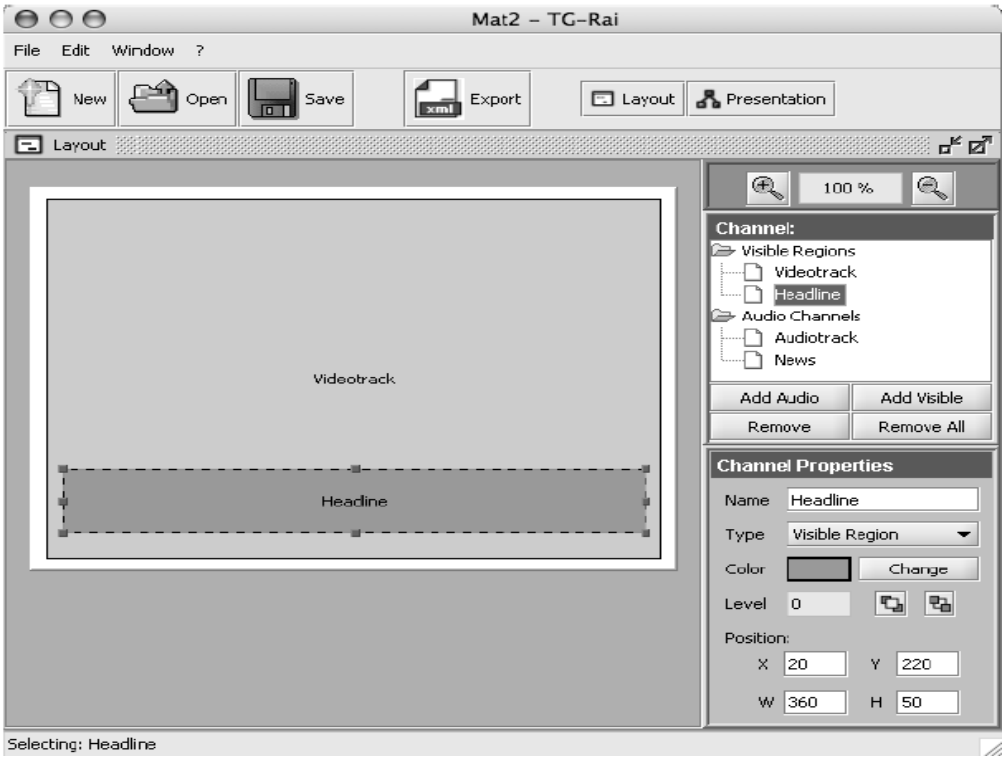


Figura 5: O leiaute e definição de canais de uma nova apresentação.

Fonte (Gaggi, 2006)

No modelo usado pelo Lamp, são definidas cinco relações de sincronização básicas e mais dois tipos de controle de mídia chamados de marcador de tempo e objeto de interação do usuário **(**Gaggi, 2006**)**. Na barra de ferramentas da Figura 6, em branco perto dos botões de zoom, são mostradas as relações de sincronização básica.

O Lamp faz uso da mesma linguagem para definição tanto de documentos quanto de *templates*. O resultado da apresentação é salvo em um documento XML, que pode suportar várias propostas, como a tradução para SMIL ou tradução para outras linguagens e geração

automática de apresentações a partir de *templates*. Entretanto, existem diferenças com relação à SMIL, o que pode afetar a tradução.

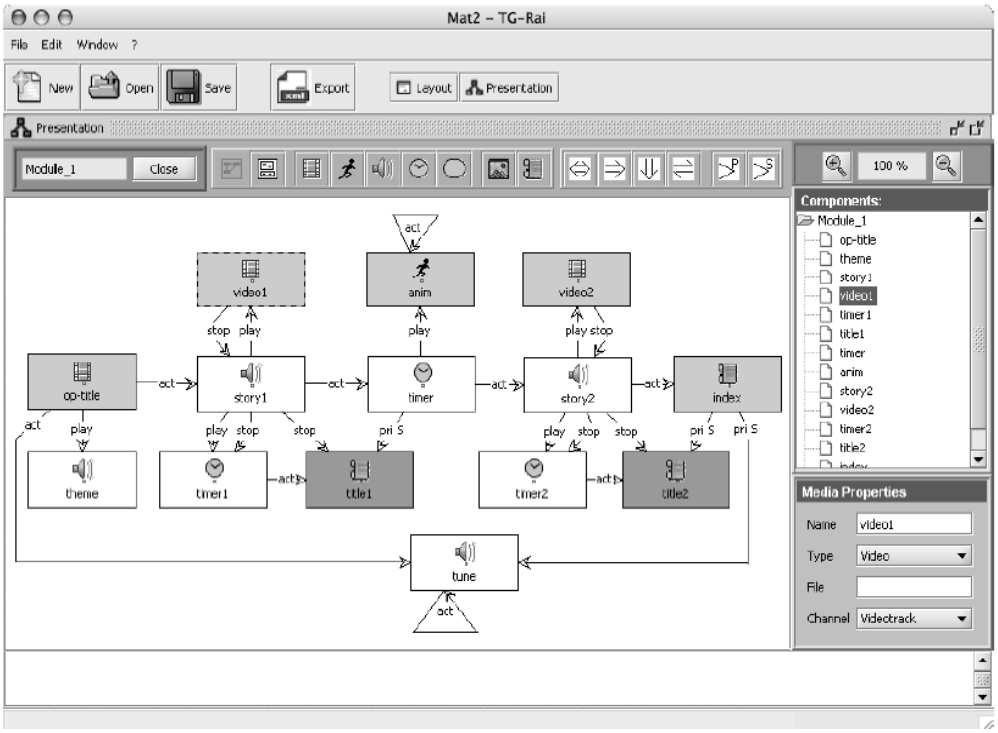


Figura 6: Sincronização gráfica de uma nova apresentação.

Fonte (Gaggi, 2006)

O documento XML da apresentação gerada organiza os dados em 3 seções: a seção de leiaute, a seção de componentes e a seção de relacionamentos. A seção de leiaute descreve o leiaute espacial dos canais, a seção de componentes contém informações sobre os objetos de mídia, e a seção de relacionamentos descreve a sincronização entre as mídias. Essa solução permite a definição de relações espaço-temporais entre objetos de mídia sem conhecer qualquer informação sobre suas localizações ou duração.

2.5 LuaComp

O LuaComp (Souza Júnior, 2009) é uma ferramenta de autoria desenvolvida em Linguagem Java que explora as funcionalidades do LuaOnTV (Souza Júnior, 2009), um framework para utilização de componentes gráficos para entrada e saída de dados em aplicações interativas. Os componentes gráficos disponíveis são utilizados na maioria dos ambientes de desenvolvimento de interface. Alguns componentes disponíveis são: *Button*,

CheckBox, *CheckGroup*, *ComboBox*, *Label*, *List*, *NumberField*, *RadioButton*, etc. O uso desses objetos diminui a necessidade de implementar códigos procedurais.

O LuaComp possui várias visões e permite a criação e utilização de *templates* em arquivos XML. O objetivo da ferramenta é agilizar a criação de aplicações interativas para TV digital, abstraindo do autor toda, ou pelo menos parcialmente, a complexidade de se programar em NCL e Lua. Ele cria aplicações interativas para TV digital tomando por base a abordagem de orientação a objetos. A Figura 7 apresenta a interface do LuaComp.

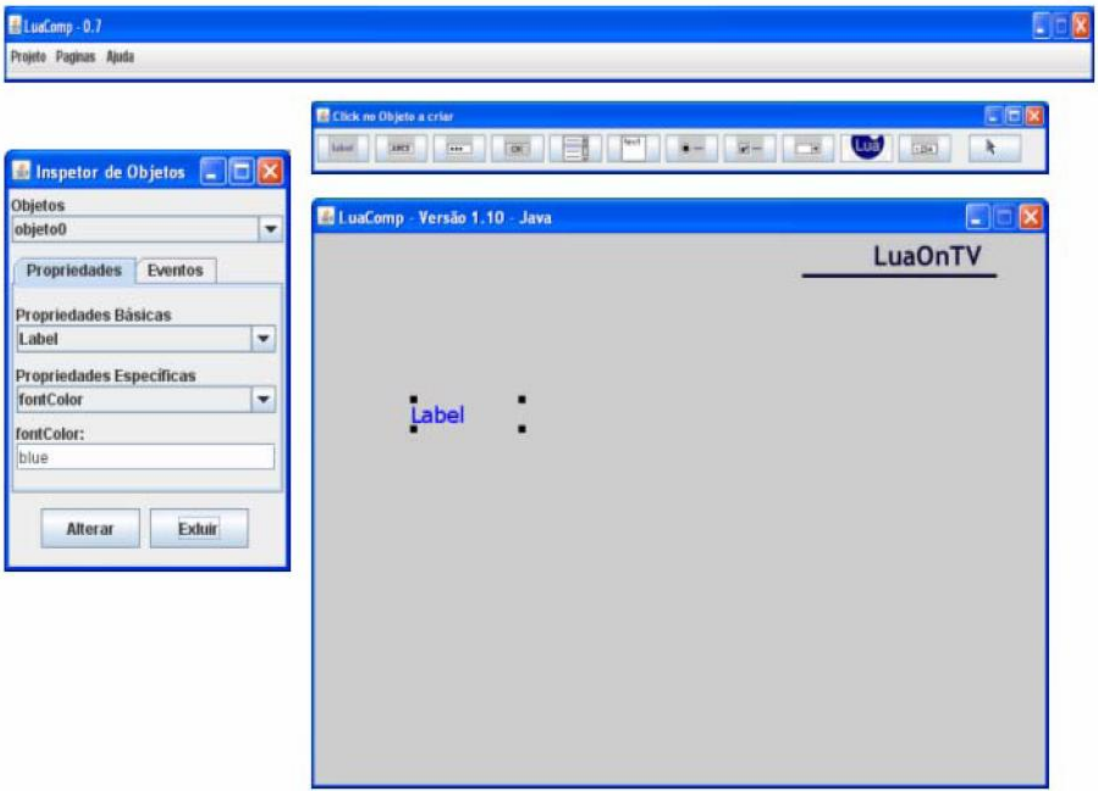


Figura 7: Interface do LuaComp.

Fonte (Souza Júnior, 2009).

Figura 8: Componentes do LuaComp 1.0.

Fonte (Souza Júnior, 2009).

O LuaComp, além de disponibilizar os componentes gráficos (ver Figura 8), também disponibiliza as visões textuais, de leiaute e estrutural. Essas visões são apresentadas na Figura 9.

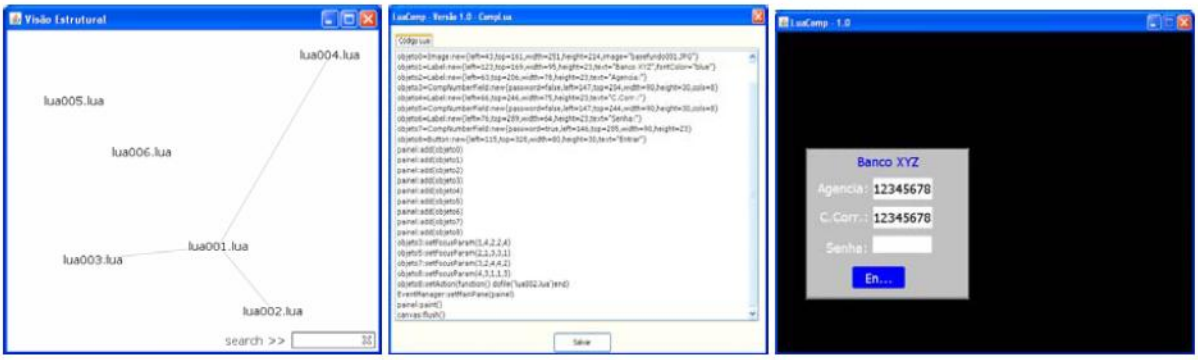


Figura 9: LuaComp: Visões estrutural, textual e de leiaute.

Fonte (Souza Júnior, 2009).

A visão estrutural permite que o autor crie, salve e altere seu projeto de páginas (telas), facilitando a manutenção. A visão de leiaute é a principal visão da ferramenta, pois é a base para a construção do design da página e permite a edição no formato *drag and drop* – arrastar e soltar. A visão textual serve como alternativa para modificações e adaptações feita por um autor experiente e conhecedor tanto de NCL quanto de Lua.

A arquitetura do LuaComp registra todo o projeto em arquivos XML. Mesmo possuindo um recurso interessante de exportação das aplicações para o formato XML, que favorece a integração com outras ferramentas de autoria, o ambiente ainda está estritamente ligado ao LuaOnTV. Isto faz com que a integração dessa ferramenta com outras soluções seja limitada pelo uso do LuaOnTV (Prota, 2010).

2.6 Contextual Ginga

O Contextual Ginga (Carvalho e Ferraz, 2010; Contextual Ginga, 2010) é uma ferramenta desenvolvida em linguagem Java para autoria de aplicações interativas para TV digital sensíveis ao contexto, executadas na plataforma Ginga-NCL. Uma aplicação é dita sensível ao contexto se a mesma utiliza informações sobre o contexto de entidades relevantes ao seu domínio para prover informações ou serviços a seus usuários. Tipicamente, informações de contexto incluem a localização, identidade ou estado de pessoas ou objetos físicos ou computacionais. Com sensibilidade ao contexto, as informações mais relevantes para um determinado usuário ficam em primeira instância e são mais facilmente acessíveis. No cenário de TV digital, informações contextuais estão sendo utilizadas principalmente para

personalização de conteúdo, oferecendo recomendação de programas e exibição de propagandas específicas para o usuário.

A seguir são apresentados alguns cenários de uso da TV Digital aprimorados com funcionalidades sensíveis ao contexto (Neto e Ferraz, 2006):

- Notificações: O contexto do mundo que cerca o usuário é notificado na tela quando pertinente e enriquecedor para sua experiência. Por exemplo, exibição de notificações sobre programas relacionados ao seu perfil, ou de amigos assistindo ao mesmo evento de TV, algo que poderia gerar interessantes interações sociais.
- Gravação de conteúdo: gravação automática de conteúdo, inclusive feita sem que o usuário tenha conhecimento prévio, para que seja utilizado posteriormente de acordo com o perfil do telespectador.
- Recomendação de Programas: Os telespectadores que se sentirem entediados na frente de televisores poderão receber sugestões de programas que sejam de acordo com o perfil do usuário, diminuindo a probabilidade do usuário realizar *zapping* (sequência de mudanças rápidas de um canal para outro).
- Propaganda: Funcionalidades de gravação de conteúdo (PVR – *Personal Video Recorder*) é um fator que dificulta aceitação de propaganda por usuários. Entretanto, o modelo de propaganda sensível ao contexto é uma ferramenta poderosa para entregar anúncios personalizados baseados no contexto do usuário.

Das categorias de informações contextuais levantadas por (Morse et al., 2000) que podem ser utilizadas como contexto em TV digital, o Contextual Ginga permite a utilização das categorias *Who*, *When* e *Where*. A categoria *Who* possui propriedades que definem a faixa etária e o sexo do usuário. A categoria *When* possui propriedades que definem a data e a hora inicial e final, e o dia da semana que determinam quando o usuário representado por uma dada "persona"[1] está assistindo à TV. A categoria *Where* possui como propriedades o país, o estado, a cidade e o código postal. Na ferramenta, essas três categorias determinam uma "persona".

A interface gráfica do Contextual Ginga foi construída a partir da API (*Application Programming Interface*) Swing e da API AWT (Sun, 2010), ambas nativas do próprio Java. A

[1] "Persona" é o conceito utilizado no desenvolvimento de produtos com design centrado no usuário. Ele representa de forma imaginária o usuário através de suas características principais, devendo ser construído a partir de dados especificados sobre pessoas reais.

sua arquitetura é composta pelo tratamento da interface gráfica, leitura e escrita dos arquivos salvos pela ferramenta, e pela geração do código da aplicação. Os dados de um projeto são armazenados em arquivos XML e divididos em quatro tipos: um de características do projeto, um para os componentes de uma tela, um para as "personas" e um último para as transições. O conjunto de elementos gráficos que devem formar uma aplicação deve estar representado na ferramenta dentro de um mesmo projeto. Esses elementos são as telas que reúnem um conjunto de componentes (*Text*, *Image* ou *Video*) que são exibidos por vez. As telas são interligadas através de transições que determinam os possíveis fluxos entre elas, que podem ser visualizados. Os possíveis fluxos dependem do contexto do usuário da aplicação.

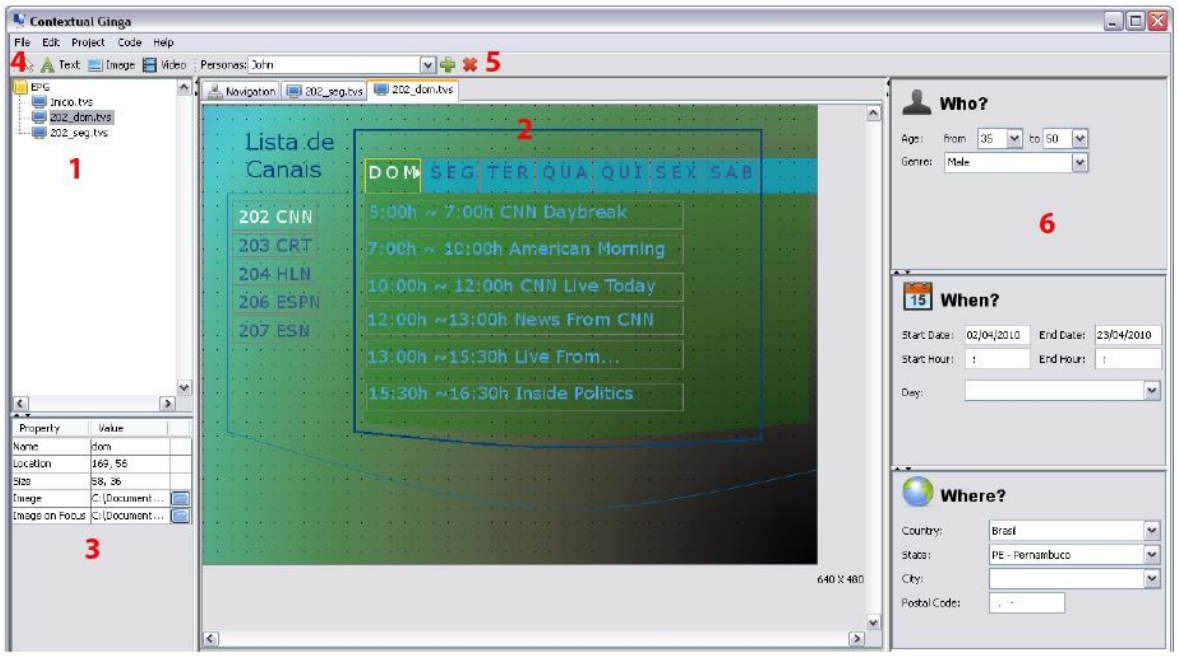


Figura 10: Contextual Ginga.

Fonte (Carvalho e Ferraz, 2010).

A Figura 10 exibe uma imagem da ferramenta com suas principais áreas destacadas. Na área 1, são exibidos o nome do projeto e as telas pertencentes a ele. Na área 2, é exibida a tela selecionada ou a navegação entre telas para a "persona" selecionada. Na área 3, são exibidas as propriedades e seus valores para um componente selecionado. Na área 4, são exibidos os tipos de componentes que podem ser adicionados às telas. Na área 5, é exibida a lista das "personas" do projeto. Por fim, na área 6, são exibidas as três categorias (*who*, *when* e *where*) das características de contexto que podem ser alteradas para a "persona" selecionada.

A ferramenta também oferece a possibilidade de visualizar graficamente, para uma dada "persona", as transições existentes entre as telas de maneira similar a um diagrama de estados.

2.7 SCO Creator Tool

O SCO Creator Tool (López et al., 2008) é uma ferramenta para criar conteúdo de ensino adaptativo para telespectadores de TV digital interativa, sem que seja necessário saber programação. Ela é utilizada para criar objetos de *t-learning* auto-adaptativos, chamados de SCO (*Sharable Content Object*). A principal característica desses objetos é ser capaz de mudar seus comportamentos de acordo com as características do estudante (telespectador), tornando mais agradável o processo de aprendizado via TV interativa.

Um SCO contém três componentes principais: um *template* Java, vários arquivos de configuração e um arquivo de adaptação. O *template* Java contém as funcionalidades do SCO, por exemplo, uma classe Java com algum espaço para texto, um vídeo, uma figura e alguns botões de controle. Cada arquivo de configuração é um arquivo XML que especifica o comportamento do SCO para uma opção concreta. Esse arquivo indica os objetos colocados nos espaços do *template* bem como as propriedades desses objetos, tais como cor, posição, etc. Existem tantos arquivos de configuração quanto os diferentes comportamentos possíveis que o SCO pode adotar. O arquivo de adaptação é um arquivo XML que contém as regras de adaptação. Essas regras indicam ao SCO, de acordo com a preferência e nível de conhecimento do usuário, qual é o comportamento mais apropriado a ser adotado. Assim, é definido o arquivo de configuração a ser usado, ou seja, o que vai ser exibido ao usuário. Essas regras são resolvidas de acordo com os valores atuais dos parâmetros de adaptação. Podem ser citados como exemplos de parâmetro de adaptação: um tipo de recurso (áudio, vídeo, etc.), um esporte preferido, um tipo de programa de TV, o gênero de um filme, o gênero de uma música, etc.

Esses SCOs podem ser manualmente criados programando o *template* Java e escrevendo os arquivos XML correspondentes aos arquivos de configuração e adaptação. O objetivo do SCO Creator é permitir essa criação sem que o criador de conteúdo saiba programação ou sintaxe de arquivos XML. Os *templates* Java são criados por programadores experientes.

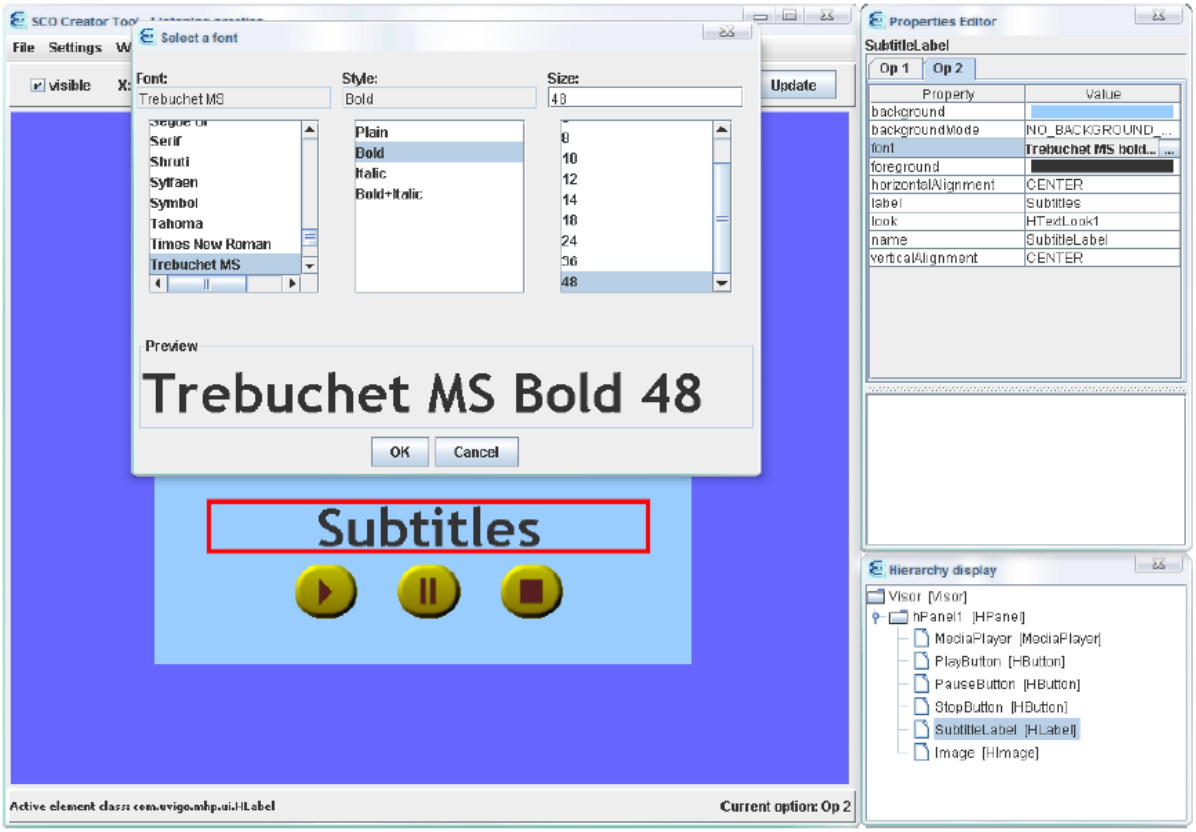


Figura 11: SCO Creator Tool.

Fonte (López et al., 2008)

A Figura 11 apresenta o SCO Creator Tool. O primeiro passo no processo de criação de SCOs auto-adaptativos é escolher o *template* apropriado dentre os que foram previamente criados. Logo que o *template* é escolhido, podem ser distinguidas três áreas sobre a tela (ver Figura 11). No lado esquerdo, é exibido o *template*, onde todos os seus objetos (*buttons*, *video-players*, *audio-players*, etc.) podem ser selecionados pelo mouse a fim de mudar as suas propriedades (estilo de fonte, figura de fundo, nome, etc.). No topo do canto direito, o criador de conteúdo pode mudar os valores das propriedades dos objetos selecionados. Finalmente, na parte de baixo do canto direito, os objetos podem ser hierarquicamente visualizados, o que permite facilmente selecionar os objetos que estão contidos em outros.

Depois de especificar cada opção, o criador de conteúdo pode definir as regras de adaptação, a fim de indicar ao SCO, de acordo com as características de cada usuário, qual o comportamento que ele deve exibir ao usuário. A fim de tornar essa tarefa mais fácil, a ferramenta possui um editor de regras de adaptação, exibido na Figura 12. Nele, o criador de conteúdo poderia ser capaz de indicar as características do usuário alvo para cada opção usando uma expressão lógica. No exemplo apresentado nessa figura, a opção 1 poderia ser oferecida a um usuário com nenhuma deficiência visual, enquanto que a opção 2 é pretendida

para usuários com visão desabilitada. O programa converte a expressão lógica em código XML e os criadores não necessitam saber a sintaxe dos arquivos de adaptação.

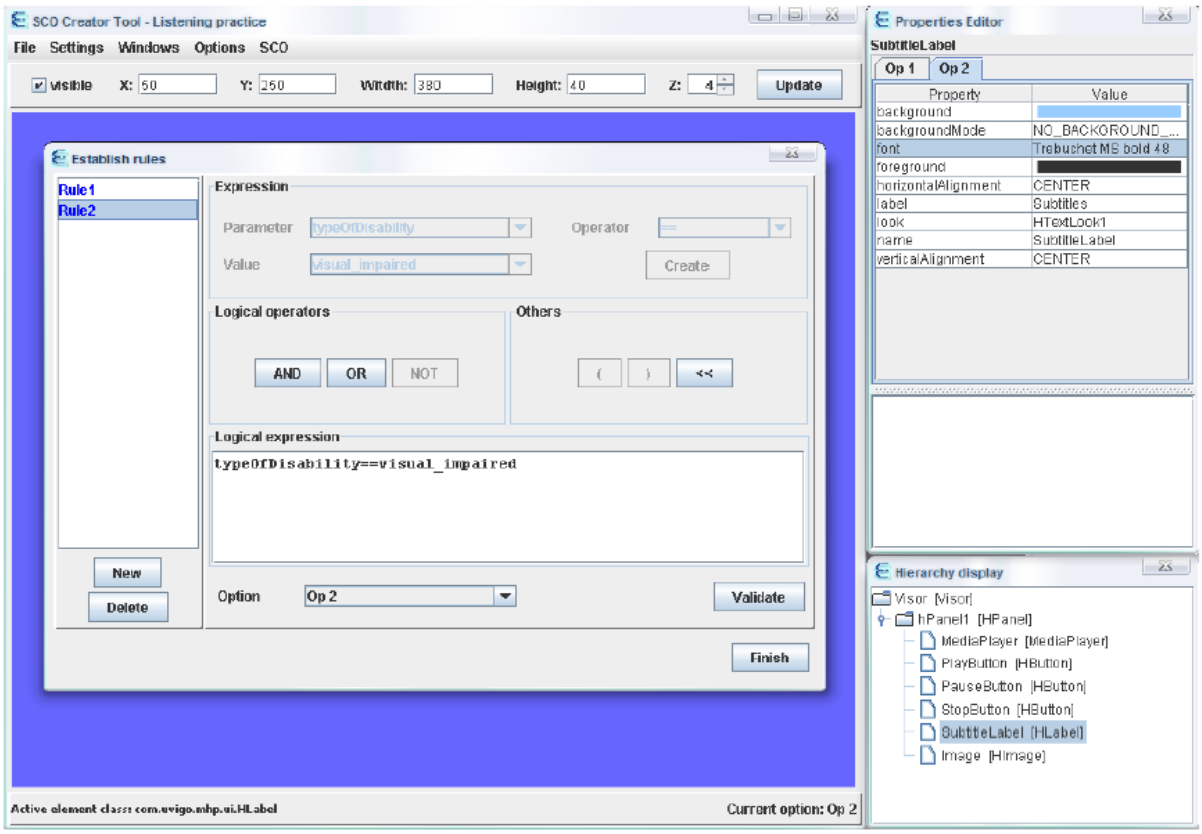


Figura 12: SCO Creator Tool: regra de adaptação.

Fonte (López et al., 2008)

2.8 Análise Comparativa

Após o estudo dos diversos recursos oferecidos pelas ferramentas de autoria, podem ser destacadas as suas principais funcionalidades. Assim, a Tabela 1 apresenta as principais características das ferramentas discutidas neste capítulo, que servem de base para a elaboração de uma ferramenta de edição gráfica de *templates* de composição. Lembrando que o tempo de maturidade de uma ferramenta e a sua demanda é um ponto importante para que ela se torne mais estável e agregue mais recursos a serem oferecidos aos seus usuários.

Tabela 1: Análise comparativa das ferramentas de autoria.

Ferramenta	**Contexto Principal**	**Principais Características**	**Linguagem Base**	**Foco**	**Permite o Uso ou Criação de *Templates***
LimSee2	web	- Sincronismo de mídias; - Visões de leiaute, temporal, XML e de código.	SMIL e XML.	Sincronismo temporal	Não
LimSee3	web	- Árvore XML dos elementos do documento; - Bibliotecas de mídias; - Uma linha de tempo e leiaute por objeto.	Linguagem própria, baseada em SMIL.	Criação e uso de *Templates*	Sim
Composer	TV Digital	- Visões temporal, de leiaute, estrutural e textual; - Sincronização das visões; - Emulador Ginga-NCL.	NCL	Sincronismo temporal/espacial	Não, mas pretende implementar no futuro.
Lamp	web	- Visões de leiaute e gráfica; - Simulador de Apresentações; - Cinco operadores de sincronização e mais dois tipos de controle de mídias.	Linguagem própria, baseada em XML.	Sincronização gráfica.	Sim
LuaComp	TV Digital	- Possibilita importar e criar *templates* de páginas e projetos; - Visões de leiaute, estrutural e textual; - Inserção de componentes gráficos; - Facilita a integração com outras ferramentas (exporta aplicação para formato XML)	NCL, XML e Lua	Reúso de componentes	Sim

Contextual Ginga	TV Digital	- Geração de código nas linguagens NCL e Lua; - Abstração para o usuário dos conceitos das linguagens de programação; - Visualização gráfica de transições entre telas.	NCL e Lua	Inserção de sensibilidade ao contexto nas aplicações.	Não
SCO Creator Tool	TV Digital	- Visão hierárquica; - Visão de propriedades; - Editor de regras de adaptação; - Conteúdo auto-adaptativo	Java e XML	*Templates*; Ensino via TV com conteúdo adaptativo	Sim

Capítulo 3

Templates de Composição para NCL

Um documento hipermídia pode usar composições para aprimorar sua estrutura e organização conforme necessário ou desejado. Algumas linguagens de autoria de documentos hipermídia também embutem semântica às composições, tornando a criação de documentos hipermídia uma tarefa mais simples, pois não é preciso definir relacionamentos específicos entre os componentes. Podemos citar como exemplo de composições com semântica, as composições sequencial, paralela e exclusiva da linguagem SMIL. No caso da linguagem NCL, as composições (chamadas de contextos) não possuem semântica embutida, logo a especificação dos relacionamentos (representados por elos) entre os componentes se torna necessária (Santos, 2009). Uma forma de embutir semântica em contextos NCL é fazer uso de *templates* de composição.

Este capítulo faz uma breve apresentação das linguagens NCL e XTemplate, que são as linguagens foco nesta dissertação. Os detalhes mais importantes dos principais elementos das duas linguagens são exibidos e, posteriormente, é apresentado um exemplo de uso de um *template* de composição.

3.1 A Linguagem NCL

A Linguagem NCL (*Nested Context Language*) (Soares e Barbosa, 2009) é a linguagem declarativa adotada no Sistema Brasileiro de TV Digital, para o desenvolvimento de aplicações interativas declarativas. Ela é baseada no modelo NCM (*Nested Context Model*), o qual é descrito em detalhes em (Soares e Rodrigues, 2005). Como toda linguagem de autoria hipermídia, NCL foca no sincronismo entre mídias (textos, imagens, vídeos, áudio e etc.), não se preocupando com a definição do conteúdo das mídias em si. NCL define nós de mídia, para representar objetos de mídia, nós de contexto, para representar composições de nós e relacionamentos entre eles, e elos, que representam os relacionamentos de sincronismo e de interatividade entre nós de mídia ou de contexto. Um elo NCL utiliza um conector hipermídia (Muchaluat-Saade e Soares, 2002), que define uma relação genérica entre nós, sem

se preocupar com os nós que efetivamente participarão dessa relação. Um conector define um conjunto de papéis, onde cada papel identifica a função de um participante na relação.

Por ser uma linguagem declarativa, a linguagem NCL não tem como foco funções de uma linguagem de programação tradicional, como cálculos matemáticos ou a definição de estruturas de repetição. Quando esse tipo de tarefa é necessário, o autor do programa NCL emprega scripts Lua (ABNT, 2007), definidos como nós de mídia para esse fim.

NCL é uma linguagem baseada no padrão XML cujo elemento raiz, que representa um documento, tem como filhos o cabeçalho e o corpo. O cabeçalho apresenta especificações genéricas de relações e definições de apresentação que serão utilizadas em um documento, e o corpo, define as mídias, os contextos e os elos.

No cabeçalho, são definidas as bases de regiões, de descritores e de conectores, entre outras. A base de regiões é responsável por definir onde as mídias serão apresentadas na tela, como por exemplo, a posição na tela em que um determinado vídeo ou imagem será apresentado. A base de descritores define como as mídias serão apresentadas, como por exemplo, o volume de uma mídia de áudio, qual a região utilizada, uma borda na região quando ela está com o foco, etc. A base de conectores define os conectores que serão usados nos elos posteriormente definidos no corpo do documento. Por ser uma definição genérica, um mesmo conector pode ser reusado em diferentes elos.

No corpo do documento NCL, são definidos os nós e os elos que definem os relacionamentos entre os nós. Além disso, o corpo pode ainda definir composições (nós de contexto), que são utilizadas para estruturar logicamente o documento, elementos de controle de conteúdo (switches), que são utilizados para a apresentação alternativa de conteúdo e pontos de interfaces dos nós de contextos (portas) e dos nós de mídia (âncoras e propriedades), dentre outros.

Um elo é formado por um conector, que define a semântica espaço-temporal, e por um conjunto de *binds*, que relacionam cada papel do conector a um ponto de interface de um nó (porta, âncora ou propriedade). Por exemplo, se fizéssemos uma comparação de um elo NCL com a seguinte frase “O professor ensina ao aluno”, ‘professor’ e ‘aluno’ seriam os nós e ‘ensina ao’ seria o conector. De acordo com esse conector alguém tem o papel de ensinar e outro alguém tem o papel de aprender. A idéia do conector é definir a semântica do relacionamento, mas sem dizer quem serão os participantes deste relacionamento, permitindo assim a reutilização do conector em outros relacionamentos (elos) a partir da definição de novos nós. Outro exemplo seria “O pai ensina ao filho”. Neste outro elo, o conector é o mesmo da frase anterior ‘ensina ao’, mas os participantes são outros. Se em um documento

NCL, tivéssemos uma mídia vídeo e uma mídia áudio e um conector causal hipermídia do tipo “onBeginStart”, poderíamos criar um elo NCL do tipo “Quando o vídeo começar, inicie o áudio”.

3.2 A Linguagem XTemplate 3.0

A linguagem XTemplate 3.0 (Santos, 2009; Santos e Muchaluat-Saade, 2009) provê suporte para definição de *templates* de composição hipermídia. *Templates* de composição descrevem de uma forma genérica todo o conteúdo de uma composição, um contexto em NCL, definindo componentes genéricos e os relacionamentos entre eles. O documento cuja composição utiliza um *template* fica responsável apenas por indicar os nós que serão utilizados, simplificando a autoria de documentos hipermídia.

Para que um documento hipermídia usando *templates* possa ser executado em uma implementação padrão do formatador da linguagem alvo (neste trabalho, NCL), é necessário o processamento deste documento, transformando-o em um documento completo (sem *templates*). A Figura 13 ilustra essa ideia.

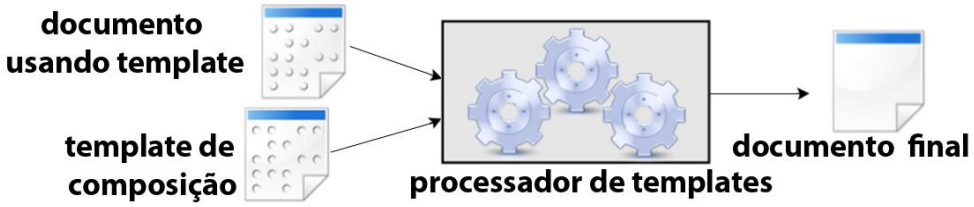


Figura 13: Utilização de *templates* na autoria de documentos hipermídia.

Com o uso de *templates* de composição, torna-se possível o reúso de especificações estruturais e espaço-temporais em diferentes contextos NCL. Além disso, facilita-se o trabalho do autor de aplicações hipermídia, uma vez que não há a necessidade de se definir toda a sincronização entre os elementos de um documento hipermídia, somente os componentes que serão utilizados.

A linguagem XTemplate, assim como NCL, é baseada em XML. A definição de um *template* de composição é feita em quatros partes principais, cabeçalho, vocabulário, corpo e restrições. No vocabulário, são definidos os componentes genéricos do *template*, onde cada um deles é identificado por um atributo *xlabel*. Este atributo deve ser usado pelos nós do contexto NCL que usa o *template*. Um nó de um contexto NCL, que utiliza um *template*, deve definir um atributo *xlabel* com mesmo valor do *xlabel* do elemento genérico correspondente definido no *template*. No corpo do *template*, são definidos, através de construções XSLT e

XPath (XSLT, 1999; XPath, 1999), os relacionamentos genéricos (elos) entre os componentes (nós) do *template*. Finalmente, a parte de restrições permite a definição de restrições adicionais sobre os componentes do *template* (nós e elos) utilizando a linguagem XPath, que são usadas para garantir a consistência do documento final gerado, evitando erros no programa criado.

3.3 Exemplo de uso de um *template*

A Figura 14 ilustra um exemplo de *template* de composição. Neste exemplo, o *template* de composição especifica a sincronização entre um objeto de vídeo, uma imagem (logo) e diversos objetos de texto (*subtitle*). As relações do tipo "L" especificam que "o início da apresentação de um objeto causa o início da apresentação de outro objeto", já as relações do tipo "P" especificam que "o fim da apresentação de um objeto causa o fim da apresentação de outro objeto".

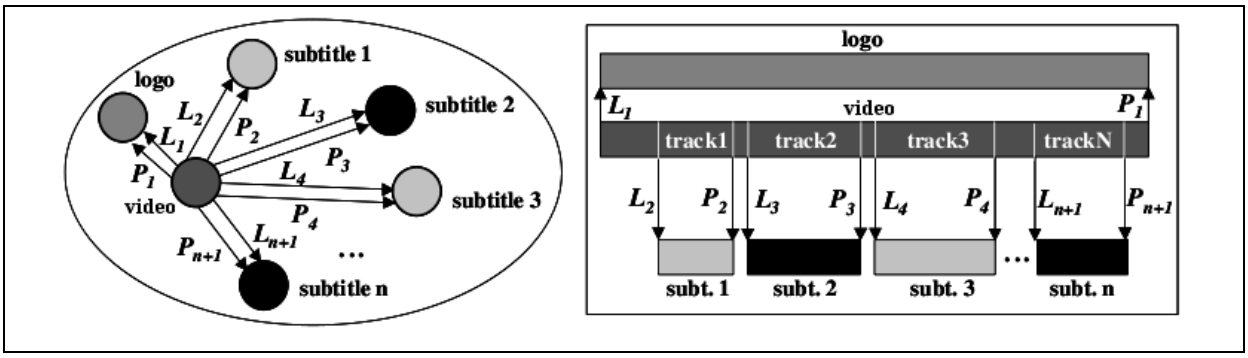


Figura 14: Visões Estrutural e Temporal de um *template* de composição.

Fonte (Muchaluat-Saade, 2003)

Podemos perceber que os relacionamentos entre um trecho de vídeo (*track*) e uma legenda (*subtitle*) se repetem, sendo necessária a definição de dois elos a cada trecho de vídeo, um do tipo "L" e outro do tipo "P". Como esses relacionamentos ("L" e "P") a cada trecho de vídeo são semelhantes, mudando apenas os participantes dos relacionamentos, podemos criar um *template* que especifica um relacionamento genérico do tipo "L" e outro do tipo "P" entre um trecho de vídeo e uma legenda e reusá-los nos demais trechos de vídeo ($track_i$) e legenda ($subtitle_i$). Assim, todo o conjunto de relacionamentos de sincronização é especificado no *template* e o documento que o utiliza só precisa definir o nó de vídeo específico, seus trechos e os nós de texto com as legendas, facilitando bastante a autoria de documentos. O nó de mídia logo não precisa ser especificado, pois é inserido pelo próprio *template*. Quando o

template define nós específicos, não é necessário que o usuário os especifique no documento que usa o *template*. O uso de *templates* torna mais fácil a tarefa de autoria, pois o autor pode especificar com um único *template* de composição o que seria especificado com o uso de vários elos.

Nesse exemplo, além da sincronização do vídeo com os objetos de texto, definições de apresentação também são feitas através da importação de bases de regiões e descritores definidas pelo *template*.

O *template* também especifica restrições. Cada restrição contém uma expressão XPath definindo a restrição e uma descrição da restrição sendo definida. Para este exemplo, as restrições do *template* comparam as quantidades dos dois tipos de elos "L" e "P" e de âncoras do vídeo com textos de legenda, restringindo que as suas quantidades devam ser iguais. Além disso, o *template* também restringe que todas as mídias definidas devem utilizar um dos três rótulos definidos no *template* (*video*, *logo* ou *subtitle*). A Listagem 1 apresenta esse exemplo de *template*.

Listagem 1: Exemplo de *template* "vídeo com legendas".

```
1.  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2.  <xtemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xt="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL" description="Este
    template e  usado para..." id="xtemplate"
    xsi:schemaLocation="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL
    http://www.midiacom.uff.br/gtvd/XTemPlate30/profiles/XTPENCL.xsd">
3.   <head>
4.              <!--importacao das bases de conectores e descritores -->
5.              <connectorBase>
6.                      <importBase alias="connBase" documentURI="connBase.ncl"/>
7.              </connectorBase>
8.
9.              <descriptorBase>
10.                     <importBase alias="presBase" documentURI="presBase.ncl"/>
11.             </descriptorBase>
12. </head>
13.
14. <vocabulary>
15.             <!-- definicao dos componentes e de seus pontos de interface-->
16.             <component descriptor="presBase#d_video" maxOccurs="1" minOccurs="1" xlabel="video" >
17.                     <port xlabel="track"/>
18.             </component>
19.
20.             <component descriptor="presBase#d_legenda" maxOccurs="unbounded" xlabel="subtitle" />
21.             <component descriptor="presBase#d_logo" maxOccurs="1" xlabel="logo" type="image/gif" />
22.
23.             <!-- definicao dos tipos de relacoes que uma composicao pode ter-->
24.             <connector src="connBase#onBeginStart" xlabel="start"/>
25.             <connector src="connBase#onEndStop" xlabel="stop"/>
```

```
26.  </vocabulary>
27.
28.  <body>
29.          <!--ponto de interface da composicao -->
30.          <port id="port" select="child::*[@xlabel = 'video']"/>
31.          <media id="logoMidia" xlabel="logo" src="medias/logo.gif" type="image/gif"/>

32.          <!--definicao dos relacionamentos entre o video e o no' logo -->
33.          <link xtype="start">
34.                  <bind role="onBegin" select="child::*[@xlabel='video']"/>
35.                  <bind role="start" select="child::*[@xlabel='logo']"/>
36.          </link>
37.
38.          <link xtype="stop">
39.                  <bind role="onEnd" select="child::media[@xlabel='video']"/>
40.                  <bind role="stop" select="child::media[@xlabel='logo']"/>
41.          </link>

42.        <!--definicao dos relacionamentos entre os trechos do video e as legendas correspondentes -->
43.          <variable name="i" select="1"/>
44.          <for-each select="child::*[@xlabel = 'video']/child::*[@xlabel = 'track']">
45.                  <link id="link1" xtype="start">
46.                          <bind role="onBegin" select="current()"/>
47.                          <bind role="start" select="child::*[@xlabel = 'subtitle'][position() = $i]"/>
48.                  </link>
49.                  <variable name="i" select="$i + 1"/>
50.          </for-each>
51.
52.          <variable name="j" select="1"/>
53.          <for-each select="child::*[@xlabel = 'video']/child::*[@xlabel = 'track']">
54.                  <link id="link2" xtype="stop">
55.                          <bind role="onEnd" select="current()"/>
56.                          <bind role="stop" select="child::*[@xlabel = 'subtitle'][position() = $j]"/>
57.                  </link>
58.          <variable name="j" select="$j + 1"/>
59.          </for-each>
60.  </body>
61.
62.  <constraints>
63.          <!--estas restricoes comparam as quantidades dos dois tipos de elos 'start' e 'stop', e de
             ancoras do video com textos de legenda, restringindo que as suas quantidades devam ser
             iguais -->
64.          <constraint select="count(child::link[@xtype='start']) = count(child::link[@xtype='stop'])"
             description="The number of links 'start' must be equal to the number of links 'stop'"/>
65.
66.          <constraint select="count(descendant::area[@xlabel='track']) =
             count(child::media[@xlabel='subtitle'])"    description="The number of tracks must be equal
             to the number of subtitles"/>
67.          <constraint select="(count(child::media[@xlabel='video']) +
             count(child::media[@xlabel='subtitle']) +   count(child::media[@xlabel='logo'])) =
             count(child::media)" description="All medias must be videos,          subtitles or logos"/>
68.  </constraints>
```

```
69.  </xtemplate>
```

Um conjunto de elos é gerado quando esse *template* é usado por um contexto que possui um nó de vídeo e legendas relativas a cada trecho do vídeo. Após o processamento do *template* e da composição, é definido o sincronismo entre os trechos do vídeo e suas respectivas legendas, além do sincronismo entre o início e término do vídeo com a imagem *logo*. É importante ressaltar que a imagem *logo*, definida como instância de componente no *template*, também passa a ser contida pelo contexto, após o processamento do *template*.

A linguagem de autoria de documentos que usam *templates* necessita de algumas extensões para permitir o uso de *templates* de composição. Essas extensões são providas, no caso de NCL, pelo perfil EXTProfile, como apresentado na Linha 2 do exemplo exibido na Listagem 2. Com essas extensões, o documento cujas composições usam *templates* pode importá-los através do elemento *templateBase* declarado em seu cabeçalho (Linhas 6 a 8). Uma composição pode referenciar o apelido (atributo *alias*) de um *template* de composição através do seu atributo *xtemplate* (Linhas 12) e identificar seus componentes usando seus rótulos (atributo *xlabel*). Note que, em NCL, o corpo do documento e um contexto (elementos *body* e context) definem composições. A Listagem 2 apresenta um documento NCL usando o *template* "vídeo com legendas". Usando *templates*, a autoria torna-se similar à autoria de documentos usando SMIL, sendo que em SMIL o usuário fica limitado a apenas a três tipos básicos de composição "seq", "par" e "excl". Usando XTemplate, cada *template* de composição pode ser considerado um novo tipo de composição.

Listagem 2: Exemplo de documento que usa o *template* "vídeo com legendas".

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <ncl id="videoLegendas" xmlns="http://www.midiacom.uff.br/gtvd/XTemplate30/EXTProfile">
3.
4.    <head>
5.      <!--importacao do template-->
6.      <templateBase>
7.              <importBase documentURI="xtemplate.xml" alias="template"/>
8.      </templateBase>
9.    </head>
10.
11.  <!--composicao referenciando o apelido do template atraves do atributo 'xtemplate' -->
12.  <body xtemplate="template">
13.
14.     <!-- um elemento especifica um xlabel igual a de um componente generico do template, assim ele
            recebe a mesma logica definida, no template, para esse componente generico-->
15.     <media id="gols38s" src="medias/gols38s.mpeg" xlabel="video" >
16.             <area id="parte1" begin="0.4s"  end ="3s" xlabel="track"/>
17.             <area id="parte2" begin="3s"  end ="7s" xlabel="track"/>
```

```
18.             <area id="parte3" begin="9s"  end ="11s" xlabel="track"/>
19.             <area id="parte4" begin="11s"  end ="19s" xlabel="track"/>
20.             <area id="parte5" begin="21s"  end ="23s" xlabel="track"/>
21.             <area id="parte6" begin="23s"  end ="28s" xlabel="track"/>
22.     </media>
23.
24.     <media id="legenda1" src="medias/legenda1.html" xlabel="subtitle"/>
25.     <media id="legenda2" src="medias/legenda2.html" xlabel="subtitle"/>
26.     <media id="legenda3" src="medias/legenda3.html" xlabel="subtitle"/>
27.     <media id="legenda4" src="medias/legenda4.html" xlabel="subtitle"/>
28.     <media id="legenda5" src="medias/legenda5.html" xlabel="subtitle"/>
29.     <media id="legenda6" src="medias/legenda6.html" xlabel="subtitle"/>
30.  </body>
31. </ncl>
```

Note que, na Listagem 2, o documento não especifica a sincronização entre os componentes e nem especificações de apresentação, uma vez que tais informações são providas, nesse exemplo, pelo *template*. Quando o documento definido na linguagem alvo usa *templates* de composição, é necessário o processamento do mesmo para transformá-lo em um documento padrão, como explicado anteriormente (ver Figura 13). Após o processamento, o documento final possui o conteúdo adicional definido pelo *template*, além do conteúdo original. A representação da Listagem 3 apresenta o documento processado.

Listagem 3: Representação do documento processado que usa o *template* "vídeo com legendas".

```
1.      <?xml version="1.0" encoding="UTF-8"?>
2.      <!-- Powered by XTemplate. -->
3.      <ncl id="videoLegendas-processed" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
4.       <head>
5.              <connectorBase>
6.                      <importBase alias="connBase" documentURI="connBase.ncl"/>
7.              </connectorBase>
8.
9.              <descriptorBase>
10.                     <importBase alias="presBase" documentURI="presBase.ncl"/>
11.             </descriptorBase>
12.      </head>
13.
14.      <body>
15.              <!--definicao do no de midia video e de suas ancoras-->
16.             <media descriptor="presBase#d_video" id="gols38s" src="medias/gols38s.mpeg">
17.                     <area begin="0.4s" end="3s" id="parte1"/>
18.                     <area begin="3s" end="7s" id="parte2"/>
19.                     <area begin="9s" end="11s" id="parte3"/>
20.                     <area begin="11s" end="19s" id="parte4"/>
21.                     <area begin="21s" end="23s" id="parte5"/>
22.                     <area begin="23s" end="28s" id="parte6"/>
23.             </media>
```

```
24.             <!--definicao dos nos de midia legenda do documento-->
25.             <media descriptor="presBase#d_legenda" id="legenda1" src="medias/legenda1.html"/>
26.             <media descriptor="presBase#d_legenda" id="legenda2" src="medias/legenda2.html"/>
27.             <media descriptor="presBase#d_legenda" id="legenda3" src="medias/legenda3.html"/>
28.             <media descriptor="presBase#d_legenda" id="legenda4" src="medias/legenda4.html"/>
29.             <media descriptor="presBase#d_legenda" id="legenda5" src="medias/legenda5.html"/>
30.             <media descriptor="presBase#d_legenda" id="legenda6" src="medias/legenda6.html"/>
31.             <media descriptor="presBase#d_logo" id="logoMidia" src="medias/logo.gif"
                        type="image/gif"/>

32.
33.             <!—porta de inicio do documento-->
34.             <port component="gols38s" id="port1"/>
35.

36.             <!--definicao dos relacionamentos do documento -->
37.             <link id="link1" xconnector="connBase#onBeginStart">
38.                     <bind component="gols38s" role="onBegin"/>
39.                     <bind component="logoMidia" role="start"/>
40.             </link>
41.
42.             <link id="link2" xconnector="connBase#onEndStop">
43.                     <bind component="gols38s" role="onEnd"/>
44.                     <bind component="logoMidia" role="stop"/>
45.             </link>
46.
47.             <link id="link3" xconnector="connBase#onBeginStart">
48.                     <bind component="gols38s" interface="parte1" role="onBegin"/>
49.                     <bind component="legenda1" role="start"/>
50.             </link>
51.
52.             <link id="link4" xconnector="connBase#onBeginStart">
53.                     <bind component="gols38s" interface="parte2" role="onBegin"/>
54.                     <bind component="legenda2" role="start"/>
55.             </link>
56.
57.             <link id="link5" xconnector="connBase#onBeginStart">
58.                     <bind component="gols38s" interface="parte3" role="onBegin"/>
59.                     <bind component="legenda3" role="start"/>
60.             </link>
61.
62.             <link id="link6" xconnector="connBase#onBeginStart">
63.                     <bind component="gols38s" interface="parte4" role="onBegin"/>
64.                     <bind component="legenda4" role="start"/>
65.             </link>
66.
67.             <link id="link7" xconnector="connBase#onBeginStart">
68.                     <bind component="gols38s" interface="parte5" role="onBegin"/>
69.                     <bind component="legenda5" role="start"/>
70.             </link>
71.
72.             <link id="link8" xconnector="connBase#onBeginStart">
73.                     <bind component="gols38s" interface="parte6" role="onBegin"/>
```

```
74.                     <bind component="legenda6" role="start"/>
75.             </link>
76.
77.             <link id="link9" xconnector="connBase#onEndStop">
78.                     <bind component="gols38s" interface="parte1" role="onEnd"/>
79.                     <bind component="legenda1" role="stop"/>
80.             </link>
81.
82.             <link id="link10" xconnector="connBase#onEndStop">
83.                     <bind component="gols38s" interface="parte2" role="onEnd"/>
84.                     <bind component="legenda2" role="stop"/>
85.             </link>
86.
87.             <link id="link11" xconnector="connBase#onEndStop">
88.                     <bind component="gols38s" interface="parte3" role="onEnd"/>
89.                     <bind component="legenda3" role="stop"/>
90.             </link>
91.
92.             <link id="link12" xconnector="connBase#onEndStop">
93.                     <bind component="gols38s" interface="parte4" role="onEnd"/>
94.                     <bind component="legenda4" role="stop"/>
95.             </link>
96.
97.             <link id="link13" xconnector="connBase#onEndStop">
98.                     <bind component="gols38s" interface="parte5" role="onEnd"/>
99.                     <bind component="legenda5" role="stop"/>
100.            </link>
101.
102.            <link id="link14" xconnector="connBase#onEndStop">
103.                    <bind component="gols38s" interface="parte6" role="onEnd"/>
104.                    <bind component="legenda6" role="stop"/>
105.            </link>
106.    </body>
107.  </ncl>
```

Note que o documento especificado pelo autor de programas para TV digital é o apresentado na Listagem 2. O documento final representado na Listagem 3 é transparente para o autor que usa *templates* de composição, tornando a autoria de programas NCL mais simples. Comparando o documento apresentado na Listagem 2 e o documento final processado da Listagem 3, podemos perceber que o número de linhas de código XML que o usuário final especifica é bem menor que o que ele deveria especificar se estivesse construindo a sua aplicação NCL sem usar *templates*.

Uma vez que XTemplate não é uma linguagem de autoria de documentos completa, ela não especifica um documento hipermídia por si só, como NCL (Soares e Barbosa, 2009), XMT (ISO/IEC, 2005) e SMIL (SMIL, 2005) fazem. XTemplate pode ser usada de forma

integrada com outras linguagens, chamadas linguagens alvo de XTemplate, provendo reúso e semântica de composições às mesmas.

Como pode-se observar, o documento que o usuário deve criar é bem simples de ser construído, sendo necessária apenas a especificação das mídias a serem usadas. A parte mais complicada de todo esse processo é a criação do *template*, pois na construção de um *template* geralmente se faz uso de mecanismos de linguagens procedurais, tais como, "for-each" vindo de XSLT e restrições construídas a partir de expressões XPath. O editor gráfico EDITEC, que será detalhado no próximo capítulo, foi desenvolvido justamente por esse motivo, ele tem como objetivo facilitar a construção de *templates* de composição.

Capítulo 4

EDITEC: Editor Gráfico de Templates de Composição

Este capítulo apresenta o EDITEC, um editor gráfico de *templates* de composição. O capítulo está estruturado da seguinte forma. Inicialmente é apresentado o ambiente de autoria EDITEC, onde são exibidas uma breve apresentação de sua arquitetura e as visões de edição disponíveis. Em seguida, é apresentado detalhadamente um modelo para representar estruturas de iteração, que permite, de forma fácil e interativa, a criação de diversos *templates* com as mais variadas semânticas espaço-temporais. Posteriormente, é exibida a interface gráfica desenvolvida para criar expressões XPath básicas. Além disso, o capítulo apresenta um exemplo de construção de uma aplicação que utiliza quatro *templates* construídos no EDITEC. Esses *templates* são construídos por intermédio das diversas funcionalidades disponíveis no ambiente de autoria e apresentadas no decorrer do capítulo.

4.1 Ambiente de Autoria EDITEC

O EDITEC (Damasceno et al., 2010), apresentado na Figura 15, é uma ferramenta desenvolvida em linguagem Java. A escolha de Java oferece portabilidade, permitindo o uso do editor em diferentes plataformas.

O ambiente de autoria EDITEC fornece três visões integradas (estrutural, de leiaute e textual), que facilitam uma melhor visão global do documento, dando ao autor um completo controle do documento durante a edição. As visões são desenvolvidas em módulos separados para facilitar a manutenção e desenvolvimento do sistema. O editor possui um pacote, que é usado pelas diversas visões, que reflete a estrutura da linguagem XTemplate, similar a proposta apresentada em (Silva, 2005), que propõe um mapeamento de todos os elementos de uma linguagem XML em classes Java. Assim, esse pacote contém classes Java de todos os elementos da linguagem XTemplate.

No ambiente de autoria EDITEC, o usuário é livre para organizar o espaço de trabalho de acordo com a sua conveniência. Em particular, os elementos visuais podem ser movidos, minimizados, maximizados e desacoplados.

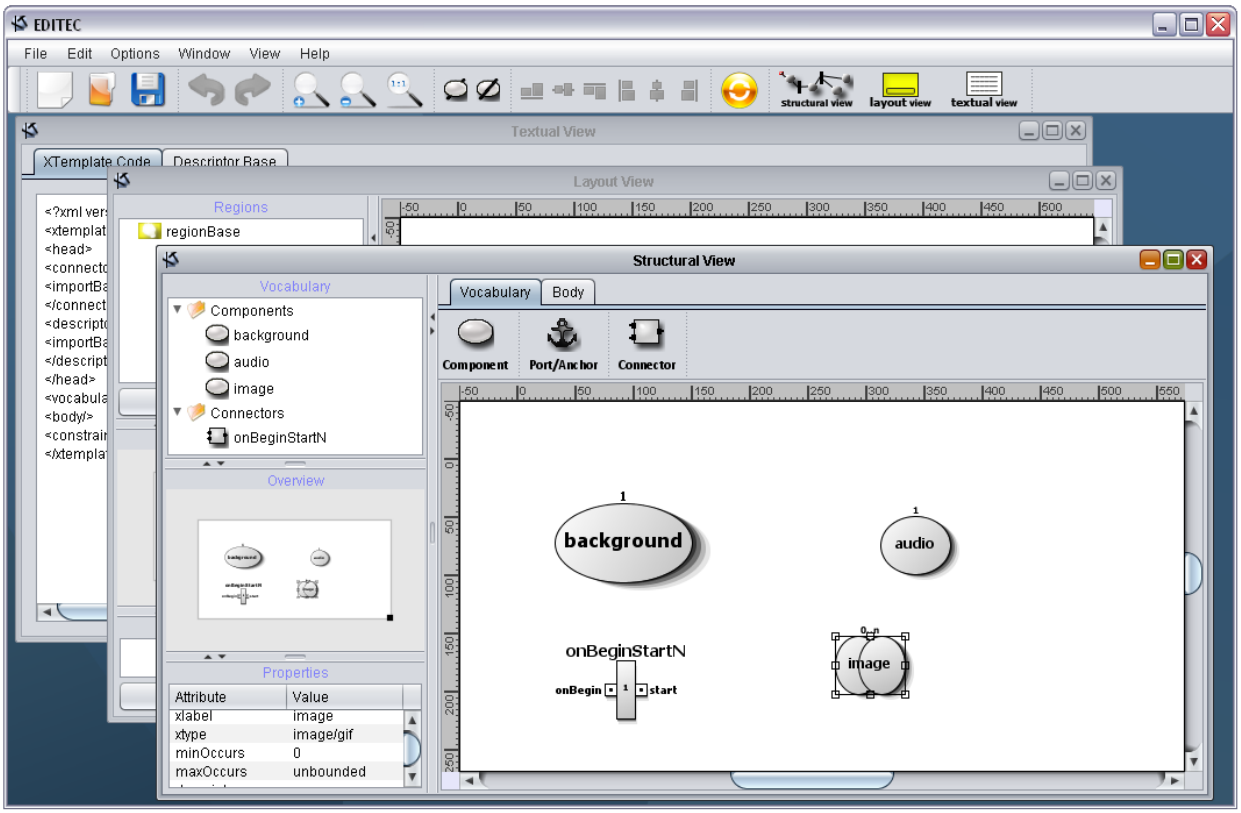


Figura 15: Ambiente de autoria EDITEC.

4.1.1 Visão Geral da Arquitetura do EDITEC

O editor está dividido em seis pacotes principais: *programs*, *layout*, *structural*, *textual*, *xtemplate e common*. O papel de cada pacote é apresentado a seguir:

- *layout*, *structural* e *textual*: são pacotes referentes às visões disponíveis no editor.
- *programs*: pacote que possui classes relativas a interface principal do ambiente de autoria EDITEC. É neste pacote que se encontra a classe "Editor", responsável pela integração de todo o ambiente de autoria. A Figura 16 apresenta o diagrama de dependências dessa classe.
- *xtemplate*: pacote que reflete a estrutura da linguagem XTemplate, onde os elementos da linguagem são representados em classes Java, este pacote é utilizado pelas diversas visões do EDITEC.

- *common*: pacote que agrega as classes de elementos comuns das principais visões.

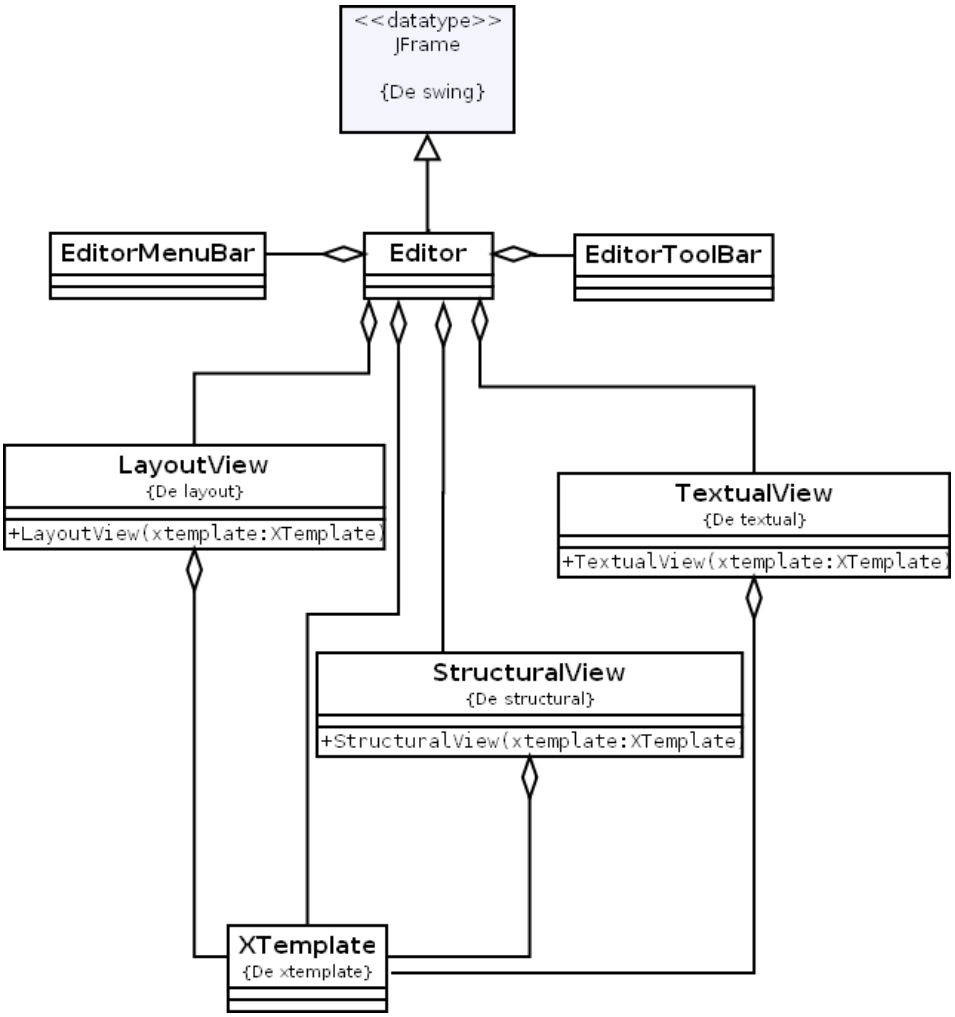


Figura 16: Diagrama de dependências da classe "Editor" do pacote *programs*.

4.1.2 Facilidades do Editor

Durante a autoria de documentos hipermídia, o autor pode ter dificuldades com a quantidade de informação para manipular e com a localização de cada pedaço da informação. Em documentos hipermídia, essa desorientação é causada principalmente pelo número elevado de nós e elos.

O EDITEC apresenta diversas funcionalidades que facilitam a visão e a busca por determinada informação. Ele apresenta em miniatura uma visão geral (*Overview*) da área de trabalho, tanto na visão estrutural quanto na visão de leiaute, onde é mostrada a área visível para o usuário e regiões em volta dessa área que não estão visíveis. Assim, elementos que não estão na região visível da área de trabalho continuam visíveis na visão geral.

Além da visão geral em miniatura da área de trabalho, o editor apresenta recursos de *zoom*, *scroll* e filtragem de elementos. Por exemplo, na visão estrutural, um duplo clique no conector, exibe ou oculta os nomes dos papéis, e um duplo clique em contextos os deixam colapsados. Na visão de leiaute podemos, por exemplo, fazer filtragem de regiões. Assim, as regiões filtradas não são exibidas na área de trabalho, melhorando a sua visualização.

4.1.3 Visão Estrutural

A visão estrutural é a principal visão do EDITEC. Ela foi desenvolvida visando a facilitar o entendimento da linguagem XTemplate 3.0. Os elementos gráficos representam as principais entidades definidas na linguagem, indicando os nós de mídia, elos, conectores, portas e contextos.

A visão estrutural é dividida em duas subvisões (duas abas na interface). Uma visão apresenta os elementos do vocabulário, chamada de *Vocabulary View*, e outra visão apresenta o corpo do *template*, chamada de *Body View*. É na visão do corpo, através da criação de elos, que os relacionamentos espaço-temporais entre componentes do *template* são definidos. As Figuras 17 e 18 apresentam a visão estrutural. A Figura 17 destaca a subvisão de vocabulário e a Figura 18, a subvisão de corpo.

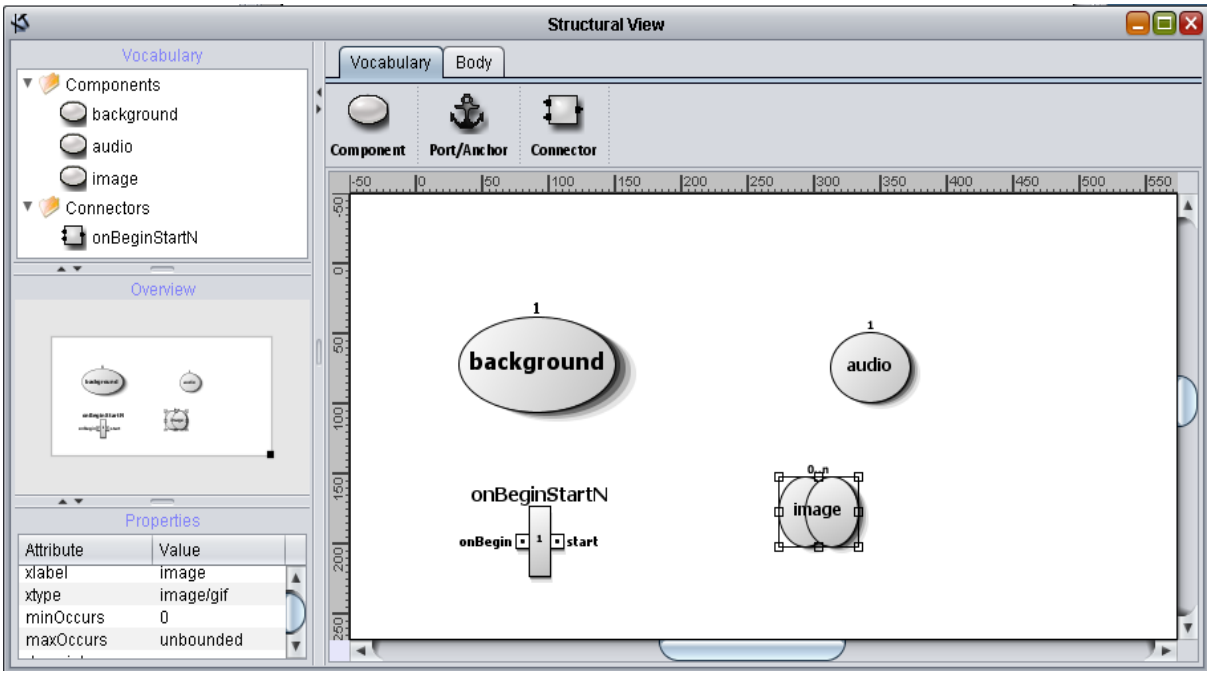


Figura 17: Subvisão de vocabulário da visão estrutural do EDITEC.

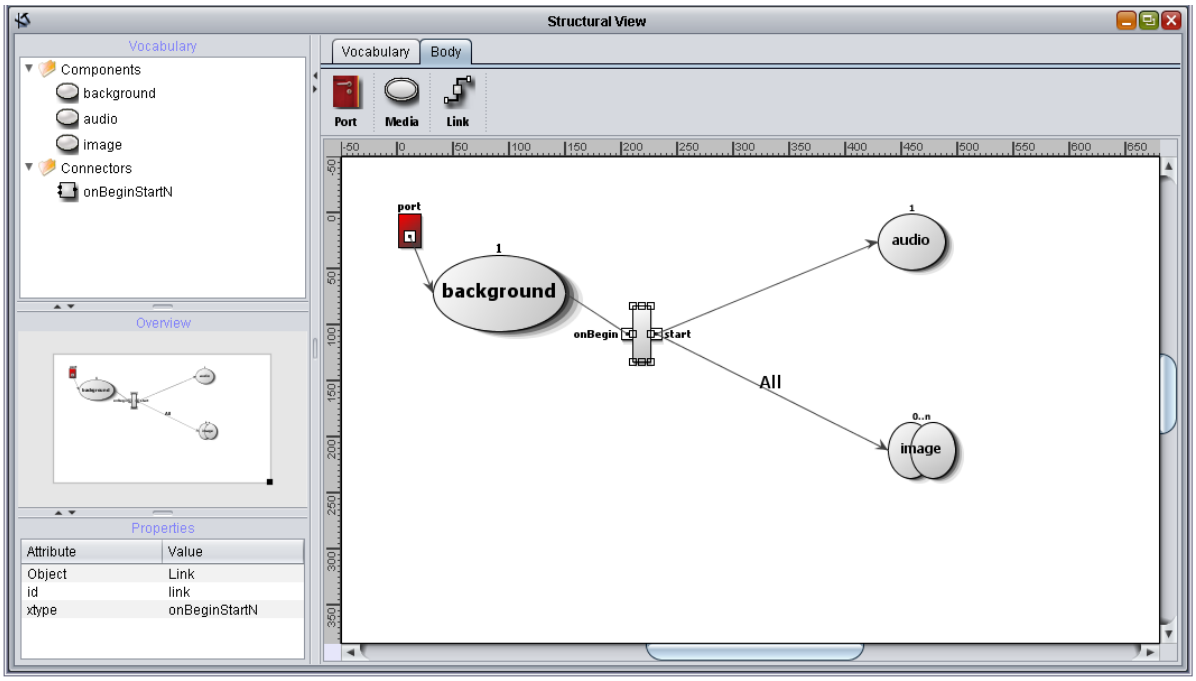


Figura 18: Subvisão de corpo da visão estrutural do EDITEC.

Para criar um novo elemento (nó componente, porta, conector, elo), o usuário seleciona o ícone que representa o elemento desejado e o arrasta para a área de trabalho na posição desejada. Quando o botão do mouse é solto, é aberta uma janela referente ao objeto que o usuário criou, onde o usuário insere informações complementares. As opções disponíveis na janela dependem do tipo de objeto criado. Depois de solto na posição desejada, o objeto pode ser movido e redimensionado, de forma a obter uma disposição espacial que favoreça a construção do *template*. Os elementos são criados de acordo com a linguagem XTemplate. Componentes, portas e conectores são criados na visão de vocabulário, já os elos e portas do *template* são criados na visão de corpo. O usuário pode tanto criar, como editar e apagar graficamente portas, componentes e elos.

Quando um elemento é selecionado, as suas propriedades são apresentadas na parte da visão estrutural referente a propriedades (canto esquerdo inferior da visão estrutural).

4.1.4 Visão de Leiaute

A visão de leiaute apresenta as regiões da tela onde as mídias poderão ser apresentadas, como ilustrado na Figura 19. Nessa visão, o autor pode criar, editar e apagar regiões graficamente. Toda vez que uma nova região é criada, é inserido um nó na árvore de regiões correspondente a nova região (apresentada no canto superior esquerdo da tela). As regiões podem ser filtradas clicando nos nós correspondentes a elas na árvore de regiões,

permitindo uma melhor visualização da área de trabalho. A Figura 19 apresenta um exemplo da visão de leiaute.

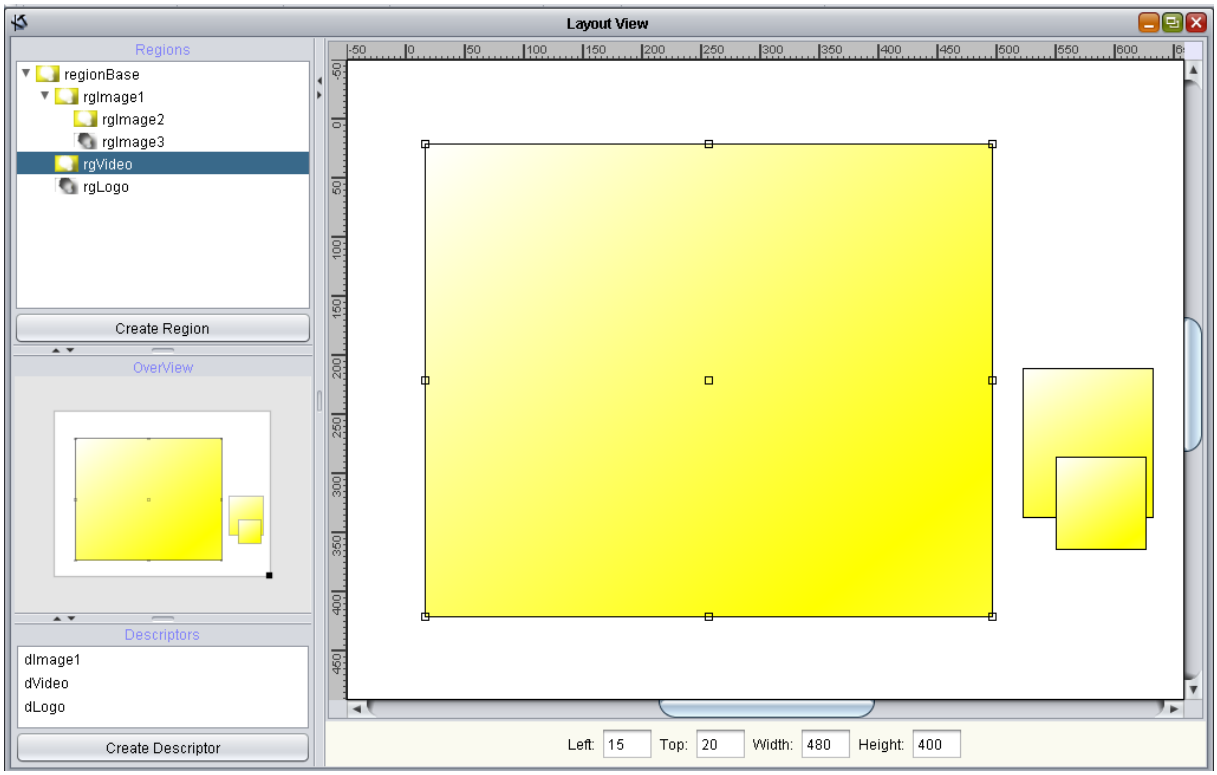


Figura 19: Visão de Leiaute do EDITEC.

O leiaute espacial das regiões pode ser definido interativamente, através de redimensionamento e movimentação. Uma região filha tem sua posição definida de acordo com a posição de sua região pai, quando uma região pai é movimentada, todas as suas sub-regiões filhas também o são, mantendo as posições relativas. Para esta visão, também existem, na barra de ferramenta do EDITEC, várias opções de alinhamento de grupo de regiões.

Na visão de leiaute também é possível a criação de descritores, com a opção de referenciar uma região criada. Os descritores criados são listados no canto inferior esquerdo da visão de leiaute. Depois que um descritor é criado ele pode ser referenciado na visão estrutural pelos componentes. Quando um componente é criado na visão estrutural, o usuário pode, através da janela que se abre para inserir informações complementares do componente, especificar um descritor dos descritores criados na visão de leiaute.

A visão de leiaute do EDITEC também serve como uma ferramenta auxiliar para usuários que desenvolvem aplicações NCL através da abordagem textual, pois eles podem utilizar apenas esta visão de leiaute para definir melhor as regiões da tela em que as mídias (imagens, vídeos, textos...) de sua apresentação serão apresentadas, pois esta tarefa usando apenas a abordagem textual é geralmente tediosa. Usando a visão de leiaute como ferramenta

complementar, os usuários têm um retorno imediato do posicionamento mais adequado das mídias na tela de exibição, facilitando e agilizando o processo de criação de documentos.

Uma observação a ser destacada é que podemos ter *templates* só de apresentação. Deste modo, a visão de leiaute se torna a principal visão de desenvolvimento de *templates*. O documento XTemplate final formado, basicamente, será composto pela base de apresentação e o vocabulário.

4.1.5 Visão Textual

A visão textual, apresentada parcialmente na Figura 20, é dividida em duas partes, uma que mostra o código do *template* e outra, relacionada à visão de leiaute, que mostra o código da base de descritores, regiões e regras do template.

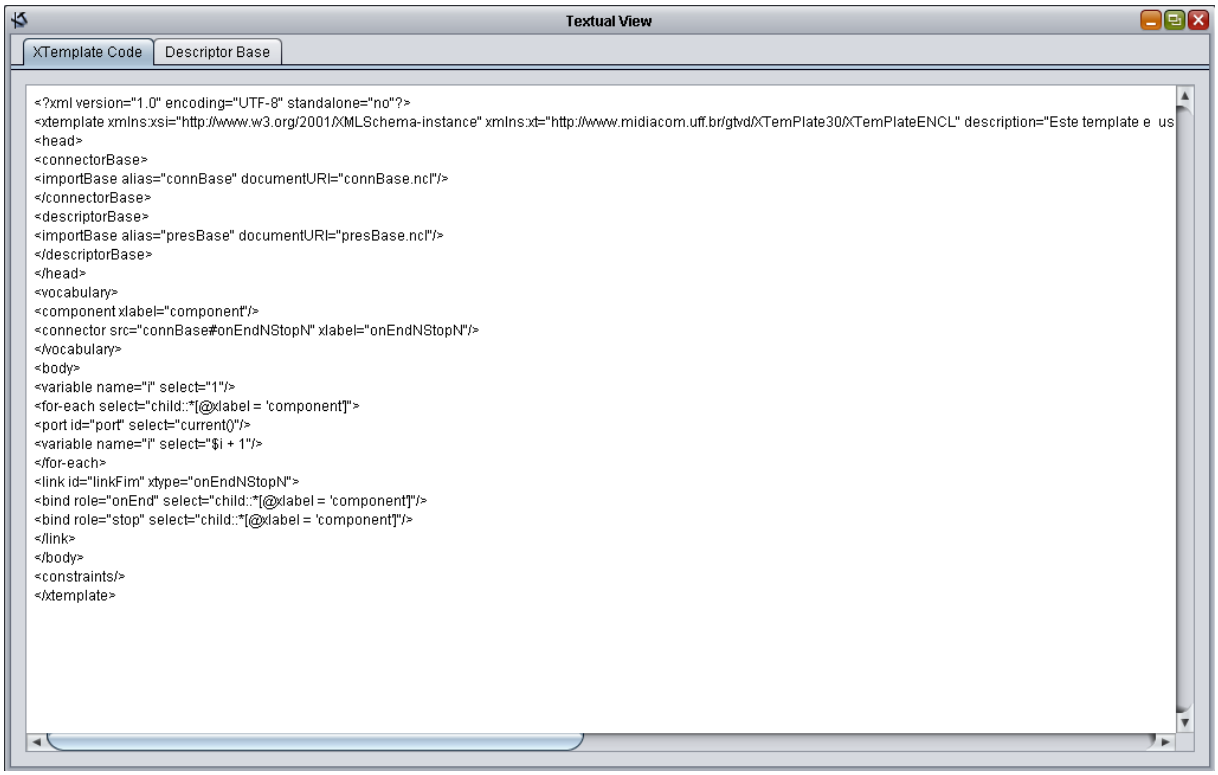


Figura 20: Visão Textual do EDITEC.

O EDITEC possui um processador de código que analisa a representação visual dos relacionamentos especificados na visão estrutural e gera automaticamente o código XTemplate desses relacionamentos. O código resultante é exibido na visão textual.

4.2 Estruturas de Iteração

Esta seção apresenta um modelo, composto de diversas opções, para representar graficamente estruturas de iteração usadas na criação de *templates*. O modelo foi desenvolvido especialmente para o EDITEC, e é usado para criar, de forma gráfica, diversos tipos de relacionamentos (elos) entre componentes de um *template*. O EDITEC analisa a estrutura gráfica do *template* com as opções escolhidas, processa-a e gera o código XTemplate correspondente em XML. As opções de iteração são especificadas em *binds* e *port mappings*, quando esses se ligam a componentes genéricos que representam um conjunto de mídias. A autoria gráfica facilita o trabalho do usuário, pois a edição gráfica do *template* é muito mais fácil e intuitiva do que a edição textual.

As estruturas de iteração são utilizadas, por exemplo, na definição do conjunto de *binds* de um elo, que relacionam papéis de um conector a componentes, ou para criação de portas, uma para cada mídia de um conjunto de mídias que um componente pode representar.

A Tabela 2 mostra as diferentes opções de estruturas de iteração e as próximas subseções as apresentam mais detalhadamente.

Tabela 2: Opções de estruturas de iteração.

Opções	Definição
i	Associa, a cada iteração, um papel de conector ao elemento da posição i de um conjunto de elementos. Onde i é incrementado a cada iteração até referenciar todas as posições possíveis do conjunto.
All	Associa um papel de conector a cada elemento de um conjunto de elementos ou cria uma nova porta para cada elemento de um conjunto de elementos.
All-i	Associa um papel de conector a cada elemento de um conjunto de elementos, mas exclui o nó gerador do evento de sofrer a ação imposta ao conjunto, caso esse nó faça parte do conjunto.
Next	Associa, a cada iteração, um papel de conector ao elemento seguinte ao da posição atual.
Prev	Associa, a cada iteração, um papel de conector ao elemento anterior ao da posição atual.

4.2.1 Opção “i”

A opção “i” associa, a cada iteração, um papel de conector ao elemento da posição i de um conjunto de elementos, onde i é incrementado a cada iteração até referenciar todas as posições possíveis do conjunto. A Figura 21 apresenta um exemplo de elo XTemplate criado

na visão estrutural do EDITEC usado para ilustrar o uso da opção “i” na criação de um relacionamento espaço-temporal em um *template*. Esse elo XTemplate possui dois componentes que representam dois conjuntos distintos de mídias, a opção “i” é usada nos *binds* ligados aos componentes “x” e “y”. Como os componentes “x” e “y” representam conjuntos de nós, esse elo XTemplate depois de processado gerará um conjunto de elos no documento NCL final.

Figura 21: Exemplo “i para i”.

Para que os nós do documento que referencia o *template* recebam a mesma lógica definida para um determinado componente interligado pelo elo, basta atribuir a eles o mesmo *xlabel* desse componente. Por exemplo, se um nó de um documento, que referencia um *template* que tenha o elo da Figura 21, receber o *xlabel* igual a “x”, quando esse documento for processado, esse nó receberá um *bind* que o ligará ao papel *onBegin* de um dos elos gerados.

Na Listagem 4, é apresentado o trecho de código do *template* relacionado ao elo da Figura 21. O código XML é gerado automaticamente pelo EDITEC e exibido na visão textual. Pode-se notar que é mais fácil e intuitivo para o usuário a criação de elos através da abordagem visual do que através do paradigma textual, principalmente quando o documento apresenta muitos elos e mídias.

Listagem 4: Trecho de código XTemplate, exemplo “i para i”.

```
<variable name = "i" select = "1"/>
<for-each select = "child::*[@xlabel = 'x']">
        <link id = "link" xtype = "onBeginStart">
                <bind role ="onBegin" select = "current()”/>
                <bind role ="start" select = "child::*[@xlabel = 'y'][position() = $i]"/>
        </link>
<variable name="i" select="$i+1"/>
</for-each>
```

Um contexto NCL que referencia um *template* que contenha esse elo, ao ser processado, se transforma em um contexto com um conjunto de elos. O processador seleciona do documento que referencia o *template* todos os nós (componentes do contexto) que têm

xlabel igual a “x”, esses nós receberão a mesma lógica definida para o componente genérico “x” do *template*. Depois de selecionados, é definida uma estrutura de repetição que varia desde o primeiro nó até o último, criando um elo a cada iteração. O primeiro elo criado contém como origem o primeiro nó com *xlabel* igual a “x” do contexto, um *bind* associa esse nó ao papel de condição *onBegin*, e como destino o primeiro nó com *xlabel* igual a “y”, um outro *bind* é usado para associar esse outro nó de destino ao papel de ação *start*. O segundo elo criado representa um relacionamento entre o segundo nó com *xlabel* igual a “x” e o segundo nó com *xlabel* igual a “y”, de forma similar ao feito no primeiro elo. Os demais elos seguem essa mesma lógica. A Figura 22 apresenta graficamente como seriam os elos resultantes no documento NCL final depois de processado.

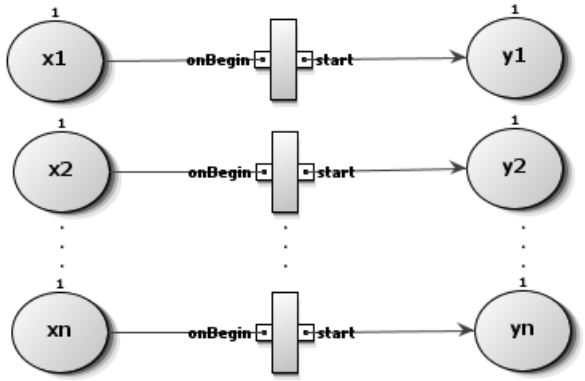


Figura 22: Representação gráfica em um documento NCL final, exemplo “i para i”.

Em algumas situações, restrições contendo expressões XPath são inseridas automaticamente pelo EDITEC no código do *template* para facilitar o seu uso. Restrições mais elaboradas devem ser editadas diretamente no documento XTemplate gerado. Um trabalho futuro é a criação de uma interface para a construção de restrições pelo usuário de forma interativa. No exemplo mostrado anteriormente, o editor insere automaticamente uma restrição, pois para que o documento possa ser processado corretamente o número de nós com *xlabel* igual a “x” deve ser igual ao número de nós com *xlabel* igual a “y”. A criação de restrições automáticas seria para este caso semelhante à exibida na Listagem 5.

Listagem 5: Restrição inserida automaticamente.

```
<constraints>
        <constraint select="count(child::*[@xlabel = 'x']) = count(child::*[@xlabel = 'y'])"
                        description = "The number of x's  must be equal to the number of y's."/>
</constraints>
```

4.2.1.1 Exemplo de uso da opção “i” no EDITEC

Para ilustrar um *template* que usa a opção de iteração “i”, usamos o exemplo “vídeo com legendas” apresentado no Capítulo 3. Nesse exemplo, o *template* de composição especifica a sincronização entre trechos de um objeto de vídeo (*tracks*), uma imagem (logo) e diversos objetos de texto (*subtitle*). Esse *template* pode ser usado em um documento onde é necessário sincronizar trechos de um vídeo com as legendas correspondentes.

Esse *template* é criado no EDITEC da seguinte forma: na *Vocabulary View*, mostrada na Figura 23, são definidos os componentes e conectores que poderão participar do *template*, sendo definidos o número mínimo e o máximo de ocorrências de cada um. A indicação de mínimo e máximo é mostrada em cima do símbolo do componente, por exemplo, em vídeo e logo o mínimo e o máximo são um e em *track* e *subtile* o mínimo é zero e o máximo é ilimitado (ilimitado é representado por *n* no EDITEC). Quando o máximo é maior que um, o componente é representado por dois círculos sobrepostos (como o componente *subtitle* no exemplo), indicando que esse componente representa um conjunto de elementos. A criação de todos os elementos é feita de forma interativa bastando apenas pressionar o mouse no ícone correspondente ao elemento desejado e o arrastar para a posição desejada da área de trabalho.

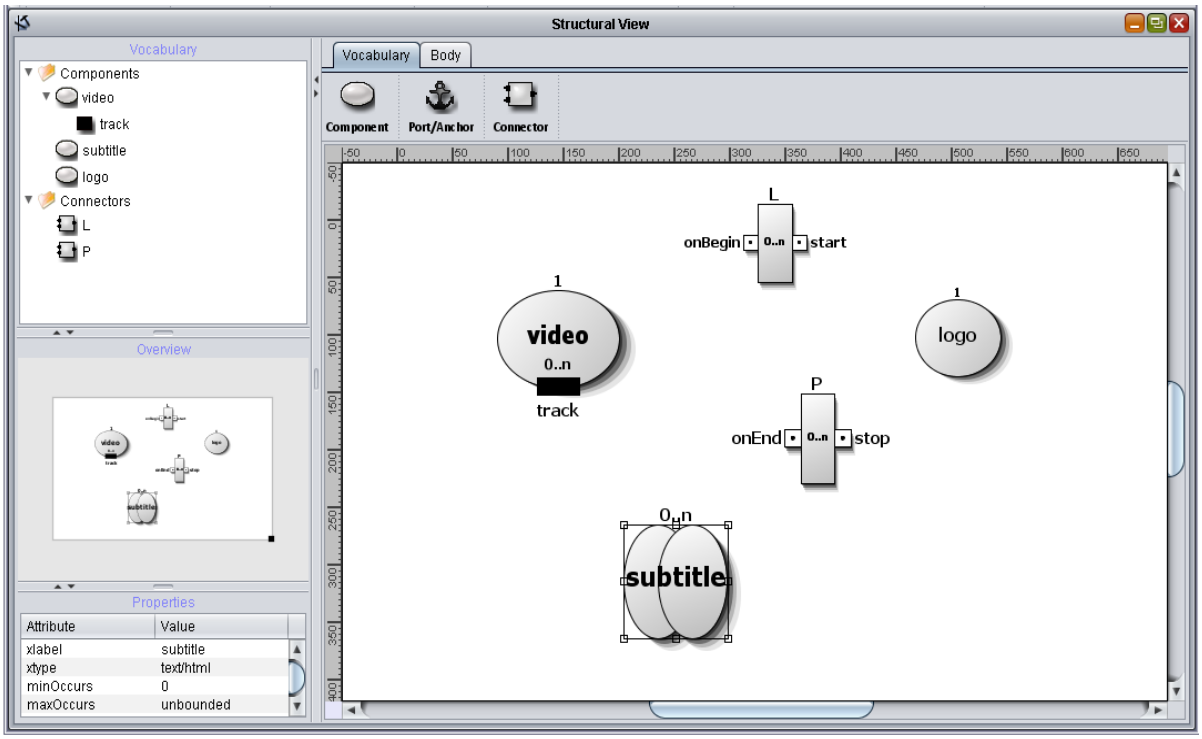


Figura 23: Subvisão de vocabulário da visão estrutural do EDITEC, exemplo: *template* “vídeo com legendas”.

Na *Body View*, exibida na Figura 24, são definidos os elos responsáveis pelos relacionamentos entre os nós, as portas que especificam os elementos que serão iniciados

quando a composição for iniciada e nós de mídia inseridos diretamente pelo *template*. Quando um *template* define nós específicos, não é necessário que o usuário os especifique no documento que usa o *template*. Os nós de mídia específicos são representados por duas circunferências concêntricas (como o nó logo no exemplo). A definição de um nó de mídia específico é feita no EDITEC pressionando o mouse na imagem "media" da barra de ferramentas da aba corpo e soltando a "media" em cima do componente desejado (neste caso, o componente logo), então é aberta uma janela onde o usuário indica a URI (*Uniform Resource Identifier*) em que o conteúdo da mídia se encontra.

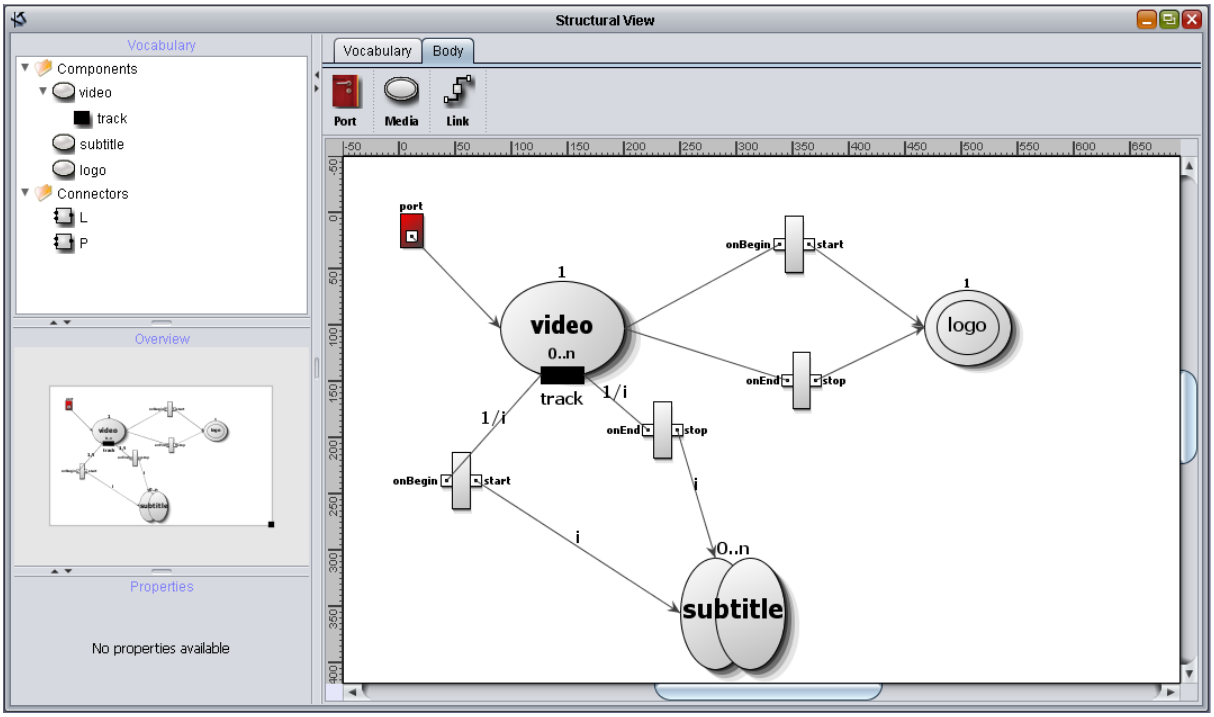


Figura 24: Subvisão de corpo da visão estrutural do EDITEC, exemplo: *template* "vídeo com legendas".

Na construção de um elo (*link*) é escolhido um dos conectores criados no vocabulário. A associação entre os papéis do conector escolhido e os nós componentes é feita através de *binds*. Para isso, basta clicar no papel desejado e soltar a ponta do mouse em cima do componente alvo, criando assim um *bind* que associa o papel do conector ao componente desejado. O mapeamento entre portas e componentes é feito de forma similar à criação de *binds*.

Nos *binds* dos elos que relacionam os trechos do vídeo (*track*) e as legendas (*subtitle*), é usada a opção de iteração "i". Para isso, clica-se duas vezes no *bind* desejado e, quando a janela for aberta, escolhe-se a opção "i". Essa janela será detalhada na Seção 4.3.

4.2.2 Opção “All”

A opção “All” associa um papel de um conector a cada elemento de um conjunto de elementos ou cria uma nova porta para cada elemento de um conjunto de elementos. Para ilustrar o uso da opção “All”, suponha um elo XTemplate interligando dois conjuntos distintos de nós, um na origem e outro no destino, onde definimos a opção “i” no *bind* referente ao componente que representa o conjunto de nós de origem e a opção “All” no *bind* referente ao componente que representa o conjunto de nós de destino. Neste caso, cada elo a ser criado deve considerar cada uma das mídias com *xlabel* “y” como associada ao papel de destino do conector. A Figura 25 apresenta um elo criado na visão estrutural do EDITEC que ilustra esse exemplo.

Figura 25: Exemplo de opções “i” e “All”.

Na Listagem 6, é apresentado o trecho de código do *template* relacionado a esse elo. O código XML é gerado automaticamente pelo EDITEC.

Listagem 6: XTemplate, exemplo de opções “i” e “All”.

```
<variable name = "i" select = "1"/>
<for-each select = "child::*[@xlabel = 'x']">
        <link id ="link" xtype="onBeginStartN">
                <bind role ="onBegin" select = "current()”/>
                <bind role ="start" select ="child::*[@xlabel = 'y']"/>
        </link>
        <variable name ="i" select ="$i+1"/>
</for-each>
```

Um contexto NCL que referencia um *template* que contenha esse elo, ao ser processado, se transforma em um contexto com um conjunto de elos. O processador seleciona do contexto NCL todos os nós que têm *xlabel* igual a “x”. Depois é definida uma estrutura de repetição que varia desde o primeiro nó até o último, criando um elo a cada iteração. O primeiro elo criado contém como origem o primeiro nó com *xlabel* igual a “x”, um *bind* associa esse primeiro nó ao papel *onBegin* e outros *binds* associam o papel *start* a todos os nós que tenham *xlabel* igual a “y”. O segundo elo criado usa um *bind* para associar o segundo nó com *xlabel* igual a “x” ao papel *onBegin* e outros *binds* para associarem todos os nós com

xlabel igual a "y" ao papel *start*, de forma similar ao feito no primeiro elo. Os demais elos seguem essa mesma lógica. A Figura 26 apresenta graficamente como seriam os elos resultantes no documento NCL final depois de processado.

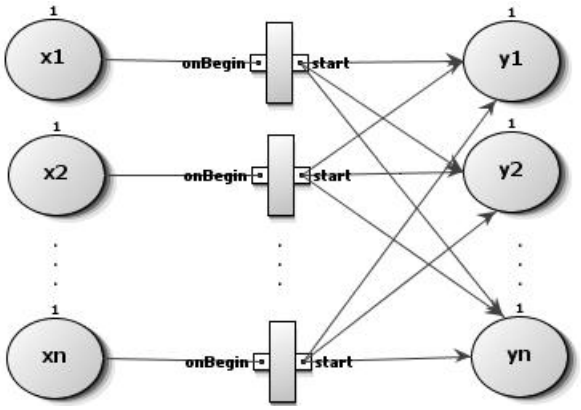


Figura 26: Representação gráfica em um documento NCL final, exemplo de opções "i" e "All".

Nesse exemplo, poderíamos ter colocado a opção "All" no *bind* relacionado ao papel *onBegin*, como exibido na Figura 27. Assim, seria criado apenas um elo com vários *binds* ligados ao papel *onBegin*, um para cada nó com *xlabel* igual a "x" e vários *binds* ligados ao papel *start*, um para cada nó com *xlabel* igual a "y". A Listagem 7 mostra o trecho de código do *template* relacionado a esse outro elo.

Figura 27: Exemplo de opção "All".

Listagem 7: Trecho de código XTemplate, exemplo "All".

```
<link id ="link" xtype="onBeginNStartN">
        <bind role ="onBegin" ="child::*[@xlabel = 'x']"/>
        <bind role ="start" select ="child::*[@xlabel = 'y']"/>
</link>
```

Na Figura 28, temos a representação gráfica de como seria o elo resultante no documento NCL final depois de processado.

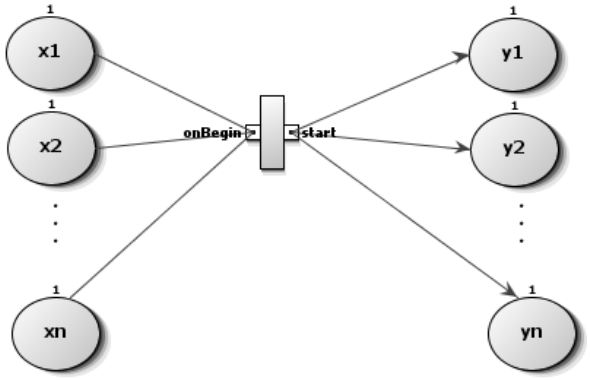


Figura 28: Representação gráfica em um documento NCL final, exemplo "All".

A opção "All" também pode ser usada na criação de portas. Neste caso, é criada uma nova porta para cada nó que tenha *xlabel* igual ao do componente alvo de *port mapping*.

4.2.2.1 Uso da opção "All" para a criação de múltiplas portas

A Figura 29 mostra como é a criação de portas no EDITEC para diversas mídias simultaneamente. Na figura, temos uma porta identificada por "port", um componente com *xlabel* igual a "x" e um mapeamento de porta (*port mapping*), com opção "All", da porta para esse componente.

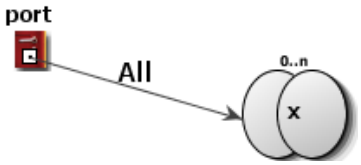


Figura 29: Exemplo de opção "All" para a criação de portas.

Na Listagem 8, é apresentado o trecho de código do *template* que define os mapeamentos das portas para os nós que têm *xlabel* igual a x. O código é gerado automaticamente pelo EDITEC. Pode-se notar que é muito mais fácil e intuitivo para o usuário a criação das portas através da abordagem visual do que através do paradigma textual.

Listagem 8: Trecho de código XTemplate, exemplo "All" para criação de portas.

```
<variable name="i" select="1"/>
<for-each select="child::*[@xlabel = 'x']">
        <port id="port" select= "current()"/>
        <variable name="i" select="$i+1"/>
</for-each>
```

Um contexto NCL que referencia um *template* que contenha essa lógica para portas, ao ser processado, se transforma em um contexto com um conjunto de portas. É criada uma

porta para cada nó do contexto que possui *xlabel* igual a “x”. O processador seleciona do contexto NCL todos os nós que têm *xlabel* igual a “x”. Depois é definida uma estrutura de repetição que varia desde o primeiro nó até o último, criando uma nova porta a cada iteração. Cada porta criada terá como atributo id o identificador definido para a porta do componente (neste caso o id “port”) mais um número, assim cada porta terá um id único. A Figura 30 apresenta graficamente como seriam as portas criadas em um documento NCL final depois de processado.

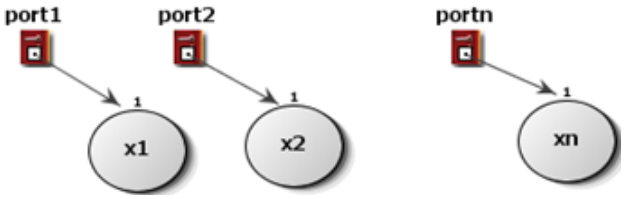


Figura 30: Exemplo 2 de opção “All” para a criação de portas.

Como mostrado na Figura 30, é criada uma porta (*port1*) para o primeiro nó, do contexto, que tenha *xlabel* igual a “x”. Depois, se houver mais nós com *xlabel* igual a “x”, cria-se uma segunda porta (*port2*) para o segundo nó do contexto, e assim sucessivamente.

4.2.2.2 Exemplo de uso da opção “All” no EDITEC

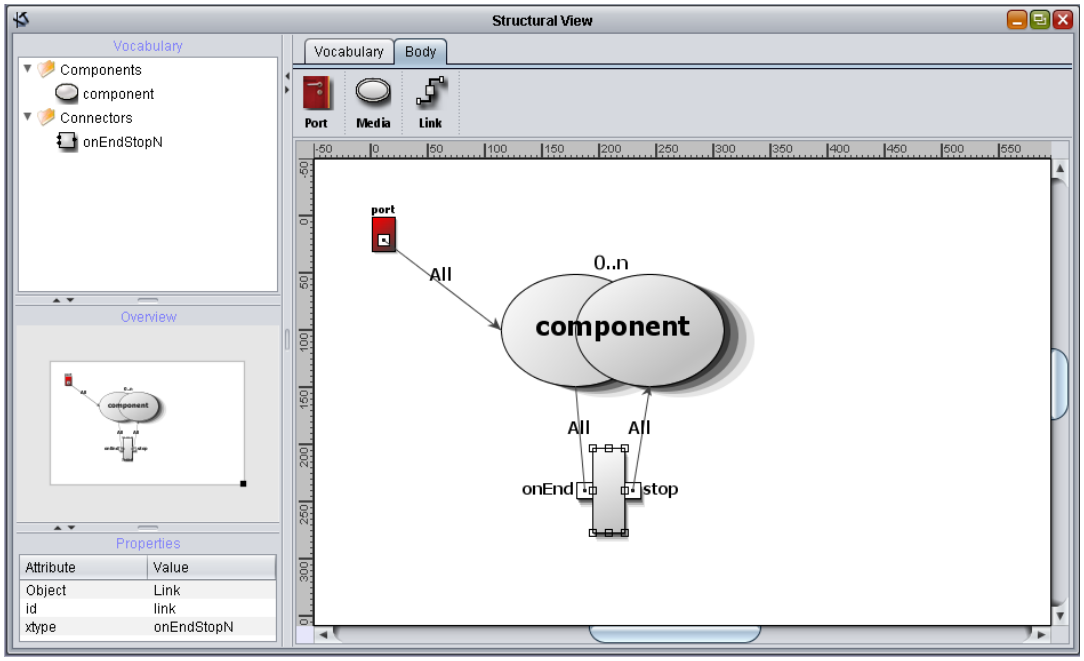


Figura 31: Visão estrutural do *template* “*parallel first*”.

A Figura 31 ilustra um exemplo de *template* de composição NCL definido com EDITEC. Este exemplo é similar à lógica *parallel first* do SMIL, onde temos um contexto, com vários objetos de mídia. Quando esse contexto é iniciado, todos os objetos de mídia, do seu interior, são iniciados ao mesmo tempo, e quando alguma mídia termina sua exibição,

todas as demais também são finalizadas. Assim, todo o conjunto de relacionamentos de sincronização é especificado no *template* e o documento que o utiliza só precisa definir os nós de mídia que participarão desse relacionamento.

Esse *template* é criado no EDITEC da seguinte forma: na *Vocabulary View*, são definidos um elemento *component*, que neste exemplo recebeu o *xlabel* igual a "component", e o conector "onEndNStopN", sendo definidos também o número mínimo e o máximo de ocorrências de cada um. A indicação de mínimo e máximo de nós de mídia que poderão receber a mesma lógica definida para "component" é mostrada em cima do símbolo do componente, que neste exemplo, pode ser de zero a infinito.

Na *Body View* são definidos o elo XTemplate responsável pelo relacionamento e a porta que especifica quais elementos serão iniciados quando a composição for iniciada. No mapeamento de porta (*port mapping*) é especificada a opção "All". Clicando-se duas vezes no mapeamento de porta, abre-se uma janela onde a opção "All" pode ser escolhida. Essa opção criará uma porta para cada nó do contexto, que referencia esse *template*, que tiver o *xlabel* igual a "component". Nos *binds* do elo XTemplate, é definida a opção "All". Esse elo dará origem a um elo NCL com vários *binds* ligados ao papel *onEnd* do conector, um para cada nó com *xlabel* igual a "component" e vários *binds* ligados ao papel *stop* do conector, também um para cada nó com *xlabel* igual a "component". O código desse *template* pode ser visualizado no Apêndice A (A.1.1), bem como um exemplo de documento (A.1.2) que referencia o *template* e o documento NCL final (A.1.3) gerado após o processamento do template.

Esse *template* poderia ser criado de outra forma, por exemplo, colocando no lugar da opção "All" do mapeamento de porta uma expressão que selecionasse apenas a primeira mídia com *xlabel* igual a "component", e criando um novo elo, além do que já existe, que definisse o relacionamento em que, quando o primeiro nó iniciar, todos os demais nós do contexto iniciem também. Para a criação desse exemplo, é necessária a definição de expressões XPath específicas. A criação dessas expressões será vista na Seção 4.3, que apresenta a interface gráfica criada para a construção de expressões XPath básicas.

4.2.3 Opção "All-i"

Esta seção e a próxima dizem respeito a casos onde um único nó é ao mesmo tempo a origem e o destino do elo. Isso leva a situações especiais, pois se pode desejar executar uma ação em um conjunto de nós quando ocorrer algum evento com algum nó deste conjunto, mas excluir o nó gerador do evento de sofrer a ação imposta ao conjunto. Para esses casos, é

utilizada a opção de iteração “All-i”, ela é uma especificidade da primitiva “All”. Por exemplo, podemos usá-la em uma aplicação onde um conjunto de N botões (ou imagens) aparece na tela. Quando algum botão deste conjunto for selecionado, todos os demais param a sua exibição, exceto o pressionado.

Na Figura 32, é apresentado um exemplo de elo XTemplate com apenas um componente representando um conjunto de nós. Esse componente tem *xlabel* igual “x”. Quando qualquer nó deste conjunto for selecionado, todos os demais somem da tela.

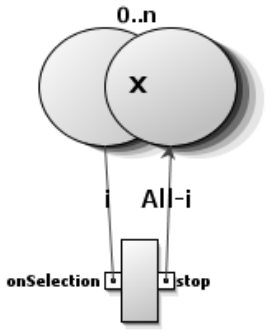


Figura 32: Exemplo de opção “All-i”.

Na Listagem 9, é apresentado o trecho de código do *template* relacionado a esse elo. O código XML é gerado automaticamente pelo EDITEC e exibido na visão textual.

Listagem 9: Trecho de código XTemplate, exemplo “All-i”.

```
<variable name="i" select="1"/>
<for-each select="child::*[@xlabel = 'x']">
        <link id="link" xtype="onSelectionStopN">
                <bind role="onSelection" select= "current()"/>
                <bind role="stop" select="child::*[@xlabel = 'x'][position() != $i]"/>
        </link>
        <variable name="i" select="$i+1"/>
</for-each>
```

Um contexto NCL que referencia um *template* que contenha esse elo, ao ser processado, se transforma em um contexto com um conjunto de elos, cada um com um conjunto de *binds*. O processador seleciona do contexto NCL todos os nós que têm *xlabel* igual a “x”. Depois é definida uma estrutura de repetição que varia desde o primeiro nó até o último, criando um elo a cada iteração. O primeiro elo criado contém como origem o primeiro nó do contexto NCL com *xlabel* igual a “x”, um *bind* associa esse primeiro nó ao papel *onSelection* e outros *binds* associam todos os nós que têm *xlabel* igual a “x”, exceto o

primeiro, ao papel *stop*. O segundo elo criado usa um *bind* para associar o segundo nó com *xlabel* igual a "x" ao papel *onSelection* e outros *binds* para associarem todos os nós com *xlabel* igual a "x", exceto o segundo, ao papel *stop*, de forma similar ao feito no primeiro elo. Os demais elos seguem essa mesma lógica. A Figura 33 apresenta graficamente como seriam os elos resultantes no documento NCL final depois de processado.

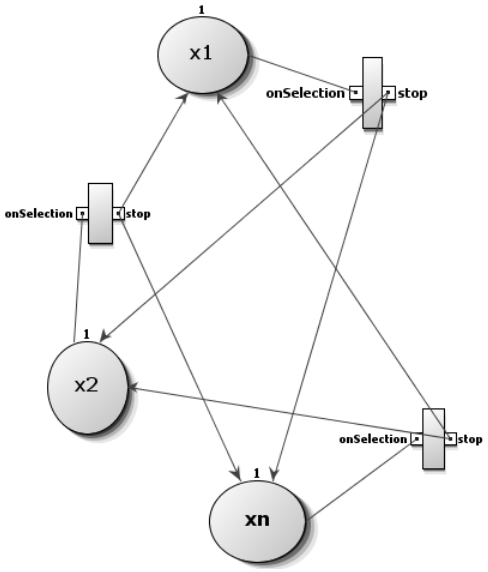


Figura 33: Representação gráfica em um documento NCL final, exemplo "All-i".

4.2.4 Opção "Next"

Em um relacionamento com a opção "Next", exemplificado na Figura 34, temos um conjunto de elos gerado a partir de um elo XTemplate que tem um mesmo componente como nó de origem e de destino. Esse componente representa um conjunto de nós, sendo que cada um deles, exceto o primeiro e o último, em um determinado elo, receberá o papel de ação e no seguinte, o papel de condição. O primeiro receberá somente o papel de condição e o último somente o papel de ação.

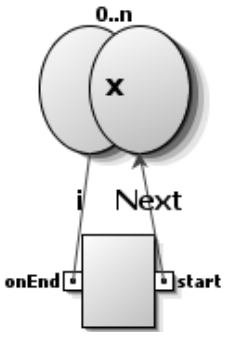


Figura 34: Exemplo de opção "Next".

Na Listagem 10, é apresentado o trecho de código do *template* relacionado a esse elo. O código XML é gerado automaticamente pelo EDITEC.

Listagem 10: Trecho de código XTemplate, exemplo “Next”.

```
<variable name = "i" select = "1"/>
<for-each select ="child::*[@xlabel = 'x'][position() ! =  last()]">
        <link id ="link" xtype ="onEndStart">
                <bind role ="onEnd" select = "current()”/>
                <bind role ="start" select ="child::*[@xlabel = 'x'] [position() = $i+1]"/>
        </link>
        <variable name ="i" select = "$i+1"/>
</for-each>
```

Um contexto NCL que referencia um *template* de composição que contenha esse elo, ao ser processado, se transforma em um contexto com um conjunto de elos, onde cada um possui dois *binds*. Como visto na Figura 35, o primeiro elo criado contém como origem o primeiro nó do contexto NCL com *xlabel* igual a “x”. Um *bind* associa esse primeiro nó ao papel *onEnd* e o outro *bind* associa o segundo nó com *xlabel* igual a “x” ao papel *start*. O segundo elo criado usa um *bind* para associar o segundo nó com *xlabel* igual a “x” ao papel *onEnd* e o outro *bind* associa o terceiro nó ao papel *start*, de forma similar ao feito no primeiro elo. Os demais elos seguem essa mesma lógica, sendo que o último nó só sofre a ação, não sendo usado no papel de condição em nenhum elo.

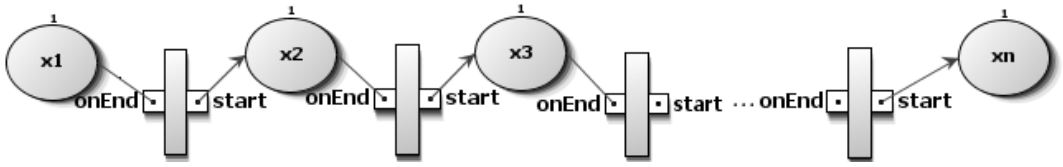


Figura 35: Representação gráfica em um documento NCL final, exemplo “Next”.

A opção “Prev” (*previous*) é semelhante à opção “Next”, só que ocorre uma inversão entre os papéis de condição e ação.

As opções “All-i”, “Prev” e “Next” são normalmente usadas quando um tipo de componente, que representa um conjunto de mídias, é a origem e o destino do elo. Em casos em que os componentes de origem e destino são diferentes, essas opções também podem ser usadas. Entretanto, as opções só têm sentido em casos específicos, por isso, é importante ter muita atenção quando se define os tipos de opções de iteração.

No processamento das opções gráficas de interação, o EDITEC leva em conta os casos especiais, onde é possível ter, por exemplo, em um único elo, as opções “All” e “Next” em diferentes *binds*.

4.3 Interface Gráfica para Declarar Expressões XPath Básicas

Uma das principais propostas no desenvolvimento de uma ferramenta de edição gráfica de *templates* de composição NCL foi a de proporcionar a construção de uma grande variedade de *templates* sem que o autor precise saber ou definir expressões XPath de forma textual. Para isso, foi desenvolvida uma interface amigável para auxiliar o usuário, substituindo a necessidade de definição textual de expressões XPath básicas por construções interativas e intuitivas da lógica das expressões XPath através de interface gráfica. Além de oferecer essa facilidade, a interface também fornece a opção de definir expressões XPath de forma textual.

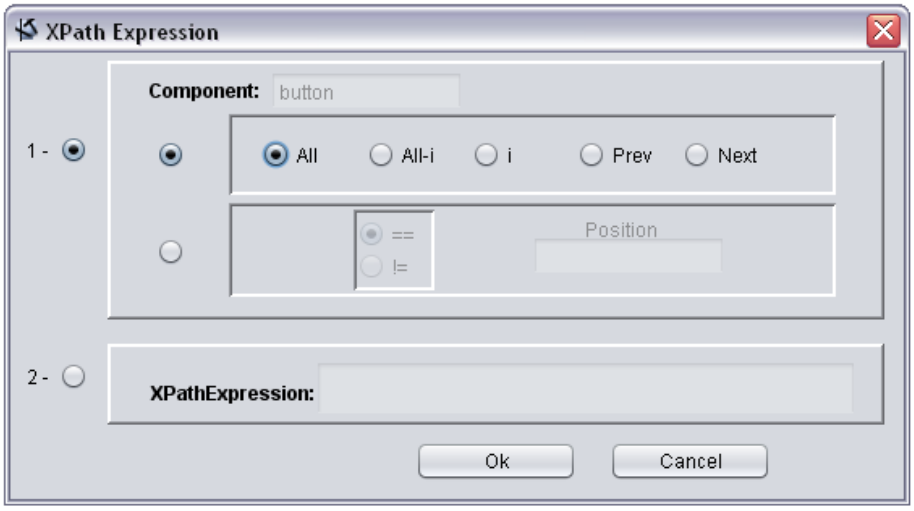


Figura 36: Exemplo de janela de definição de expressões XPath básicas.

A Figura 36 apresenta um exemplo de interface gráfica do EDITEC, com diversas opções que podem ser usadas para substituir a necessidade de definição textual de expressões XPath. A interface dá certa liberdade ao usuário para a construção de diversos tipos de expressões XPath básicas de forma interativa, como pode ser visto na segunda parte da opção 1 da Figura 36 e exemplificado mais a frente nesta seção. Além disso, a interface permite a inserção de expressões XPath textualmente, caso o usuário deseje definir alguma lógica mais específica.

A janela para a definição de expressões XPath é aberta quando se clica duas vezes em cima de um *bind* ou em um *port mapping* na visão estrutural do EDITEC. O campo "Component" da interface é preenchido automaticamente com o nome do componente alvo do *bind* (ou de um *port mapping*) responsável pela abertura da janela.

A janela é dividida em duas partes. Na primeira parte, existem duas opções. Na primeira opção, o usuário pode escolher entre definir um tipo de generalização, onde é

definida uma estrutura de repetição que serve para criar, a cada iteração, dependendo do caso, elos ou portas. Se for criação de portas só aparece, das opções de generalização, a opção “All”. Na seção anterior foi explicado detalhadamente o uso de cada uma das opções de iteração. Na outra opção da primeira parte, o usuário pode definir de forma interativa expressões XPath básicas, como:

- selecionar todos os elementos com o *xlabel* igual a "component".
- selecionar apenas o primeiro elemento (position()=1) com *xlabel* igual a "component".
- selecionar o último elemento (last()) com *xlabel* igual a "component".
- selecionar o penúltimo elemento (last()-1) com *xlabel* igual a "component".
- selecionar o elemento da posição “p”[2] com *xlabel* igual a "component".
- selecionar todos os filhos com o *xlabel* igual a "port" do n-ésimo elemento com *xlabel* igual a "context".
- selecionar o primeiro filho com *xlabel* igual a "track" do n-ésimo elemento com *xlabel* igual a "video".
- selecionar todos os elementos com *xlabel* igual a componente menos o elemento da posição “p”.

A segunda parte da janela pode ser usada para a definição textual de expressões XPath, caso o usuário deseje. Esta facilidade deve ser usada com muito cuidado para não criar inconsistências na definição do *template*, pois o usuário tem liberdade para montar a expressão com a lógica desejada.

Caso uma expressão XPath seja definida de forma interativa ou textual, ou seja, sem o uso das opções de iteração oferecidas, o *bind* (ou *port mapping*) na visão estrutural recebe a indicação “XE”, mostrando que existe uma expressão XPath específica relacionada ao componente alvo do *bind* (ou *port mapping*). As expressões específicas, identificadas pelo rótulo “XE” nos *binds* e *port mapping*, podem ser visualizadas facilmente na área referente a propriedades da visão estrutural, ou na interface XPath, no campo referente a expressões (parte 2 da interface), quando se clica duas vezes em cima do *bind* ou *port mapping*. Também podem ser vistas diretamente na visão textual do editor.

A interface exibida na Figura 36 é aberta quando o componente alvo do *bind* (ou de um *port mapping*) não é um elemento do tipo *port*. Desse modo, a parte referente a pai não é

[2] Onde “p” é o número referente à posição de um elemento dentro de um conjunto de elementos que têm o *xlabel* igual a “component”.

exibida. Quando o componente alvo do *bind* (ou de um *port mapping*) é do tipo *port*, dois tipos de janelas podem aparecer. O primeiro tipo é apresentado na Figura 37.

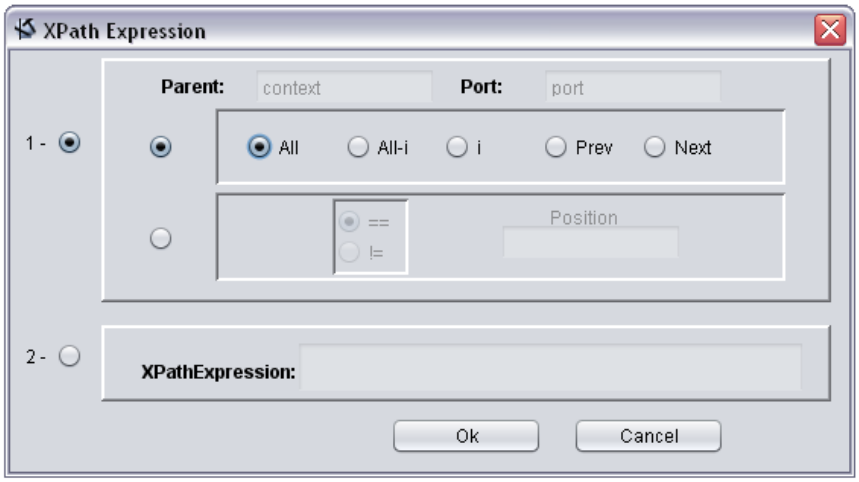


Figura 37: Exemplo 1 de Interface XPath em que o componente alvo do *bind* é do tipo *port*.

O primeiro tipo aparece em duas situações: quando o componente do tipo *port* representa apenas um elemento e o pai (componente *parent*) de *port* representa um conjunto de elementos ou quando o componente do tipo *port* representa um conjunto de elementos e o pai de *port* representa apenas um elemento. Neste caso, componente e pai são tratados como se fossem um só elemento com a notação no *bind* igual a "opção/1" ou "1/opção", como exemplificado na Figura 38, e a opção de iteração fica referente ao que representa o conjunto de elementos. Os campos "Component" e "Parent" da interface são preenchidos automaticamente. O campo "Component" é preenchido com o nome do componente alvo do *bind* (ou de um *port mapping*) responsável pela abertura da janela e o campo "Parent" é preenchido com o nome do pai desse componente.

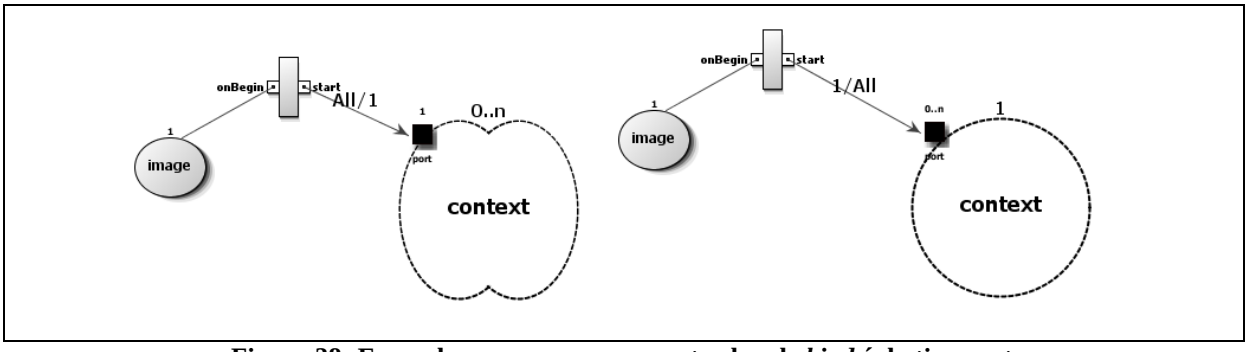


Figura 38: Exemplo em que o componente alvo de *bind* é do tipo *port*.

O segundo tipo de janela, que pode ser exibida, ocorre quando o componente do tipo *port* e o pai (componente *parent*) de *port* representam cada um, um conjunto de elementos. Essa janela é apresentada na Figura 39. Como *port* é tratado em separado de seu pai, só são exibidas as opções de iteração "All" e "i", pois as outras não fazem sentido quando o

elemento é do tipo *port*. Existem diversas combinações possíveis para uma janela que tem esses dois campos, essas combinações serão detalhadas na subseção seguinte.

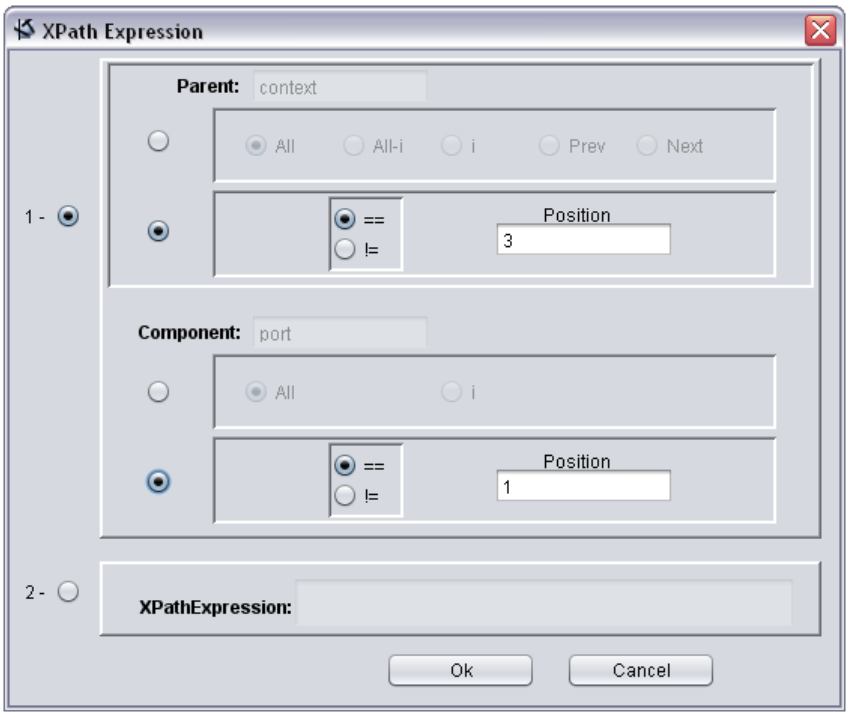


Figura 39: Exemplo 2 de Interface XPath em que o componente alvo de *bind* é do tipo *port*.

A janela XPath exibida na Figura 39 mostra um exemplo de construção de uma expressão XPath de forma interativa que seleciona o primeiro filho com o atributo *xlabel* igual a "port" do terceiro elemento com atributo *xlabel* igual a "context". Na Listagem 11, temos um exemplo de trecho de código XML de um elo que usa a lógica apresentada na interface da Figura 39, em seu segundo *bind*.

Listagem 11: Trecho de código XTemplate, exemplo de expressão XPath.

```
<link id="link" xtype="onBeginStart">
  <bind role="onBegin" select="child::*[@xlabel = 'video']"/>
  <bind role="start" select="child::*[@xlabel = 'context'][position() = 3] / child::*[@xlabel = 'port'][position() = 1]"/>
</link>
```

Quando o objeto alvo do *bind* ou *port mapping* é um componente que representa apenas uma mídia, a janela XPath não é aberta. No caso de um *bind* que associa um componente a um papel de condição, a janela XPath só habilita, das opções de iteração da parte 1, as opções “All” e “i”, como exibido na Figura 40.

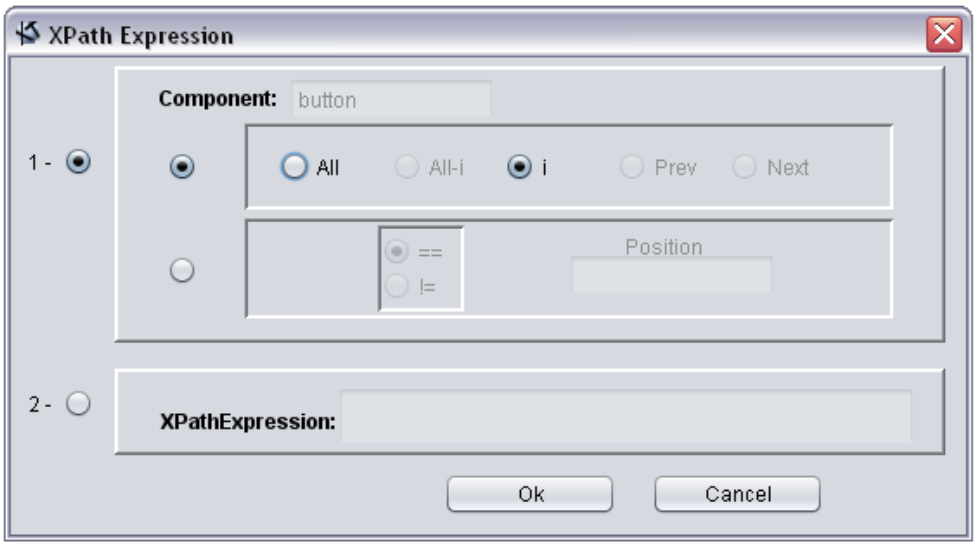


Figura 40: Parte 1 da janela XPath parcialmente desabilitada.

As janelas XPath a serem usadas na grande maioria dos casos de criação de *templates* são as janelas das Figuras 36 e 37, principalmente a 36, que é a janela a ser utilizada para todos os casos, exceto quando o componente alvo for um componente do tipo *port*. Os usuários da ferramenta não precisam se preocupar com o tipo de janela a ser aberta, pois as janelas são abertas de acordo com a situação de desenvolvimento em andamento. Assim, o tipo de janela a aparecer é transparente para o usuário.

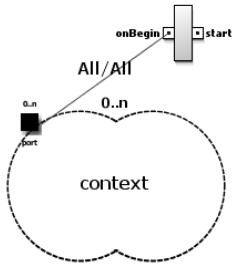
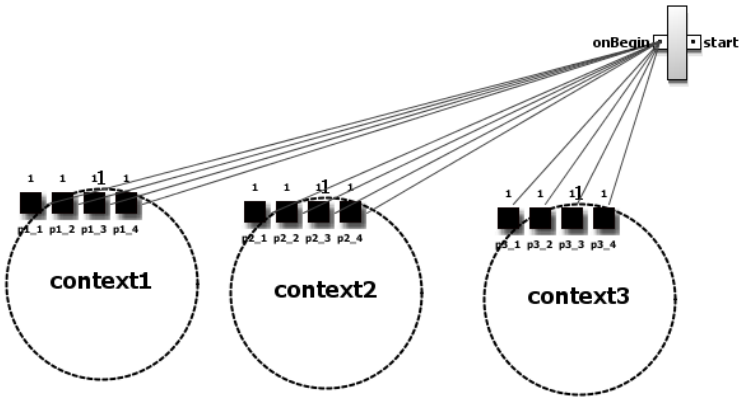
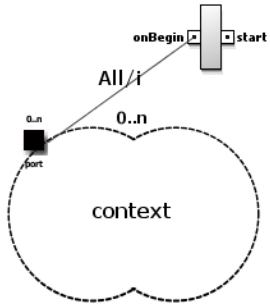
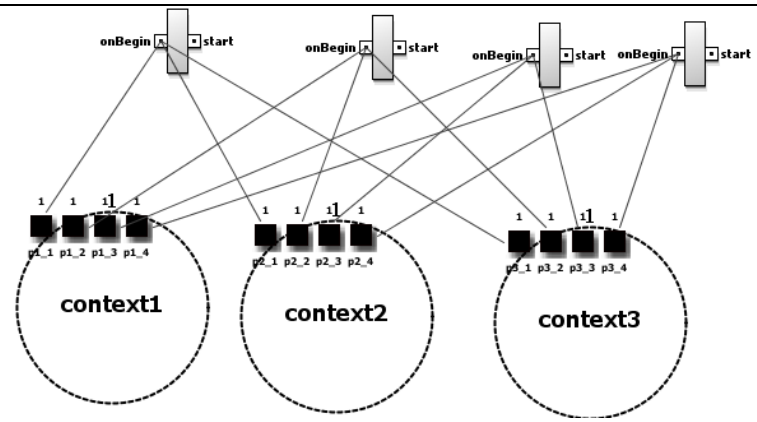
4.3.1 Análise das combinações possíveis de uma janela XPath que tenha os campos "Parent" e "Component".

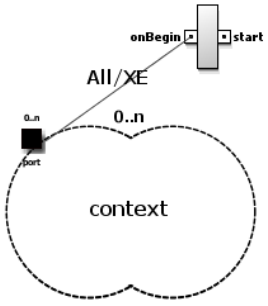
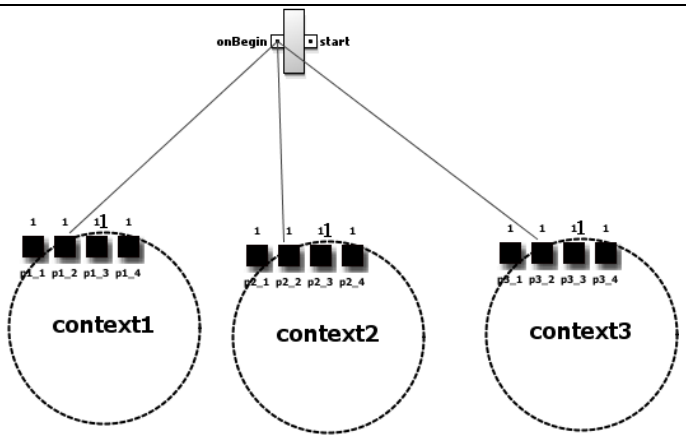
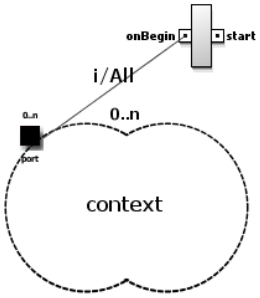
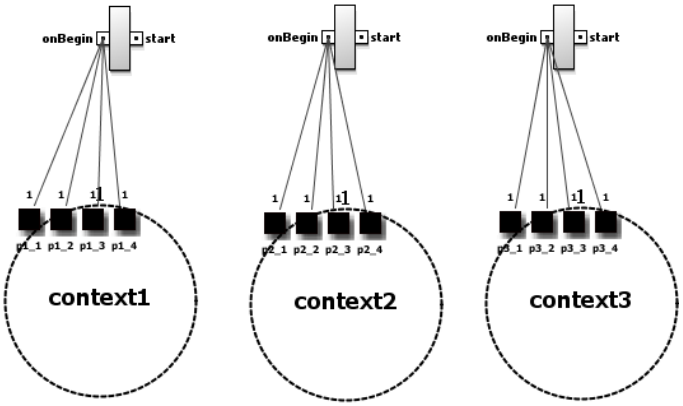
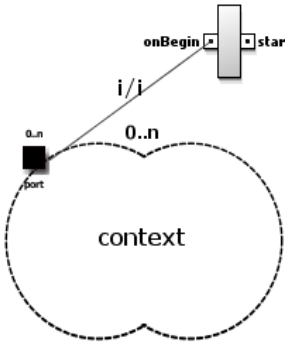
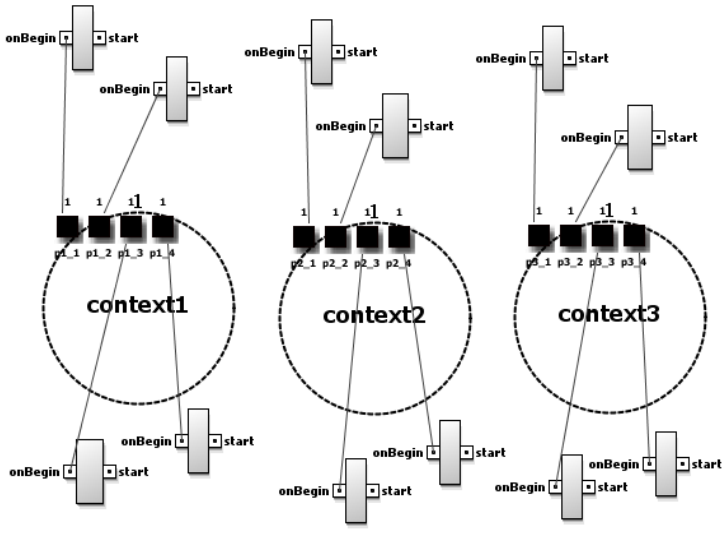
Quando a janela XPath é aberta devido a dois cliques em um *bind* associado a um componente que tem pai e, se eles, componente e pai, representarem, cada um, um conjunto de elementos, a janela XPath aberta é similar à apresentada na Figura 39. A ocorrência desse tipo de situação é rara, mas é importante que se permita, também, a criação de *templates* de forma gráfica nesses casos. A janela aberta nessa situação apresenta a possibilidade de definição de dados tanto para "Component" quanto para "Parent", possibilitando a ocorrência de várias combinações de estruturas genéricas, pois podem ser definidos operadores de iteração tanto em um quanto em outro. Esta seção apresenta as combinações possíveis para essas raras situações. O objetivo aqui é mostrar que mesmo em situações muito complexas e que raramente ocorrem, ainda é possível, na maioria das vezes, a construção dos *templates* de forma gráfica sem necessitar recorrer a edição textual.

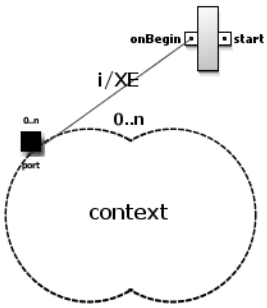
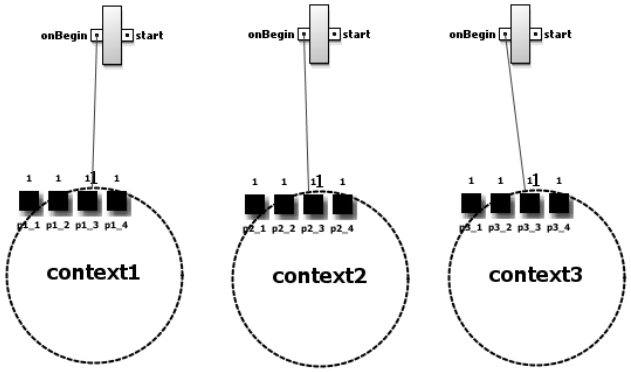
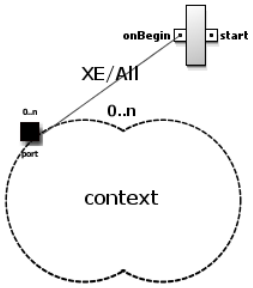
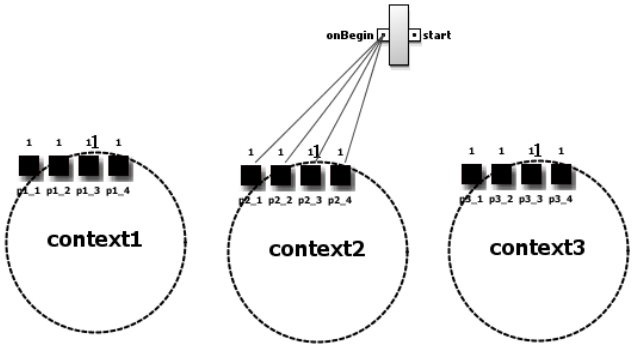
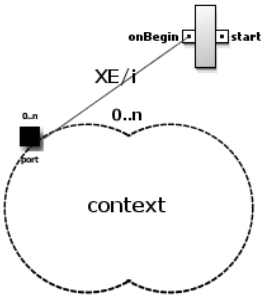
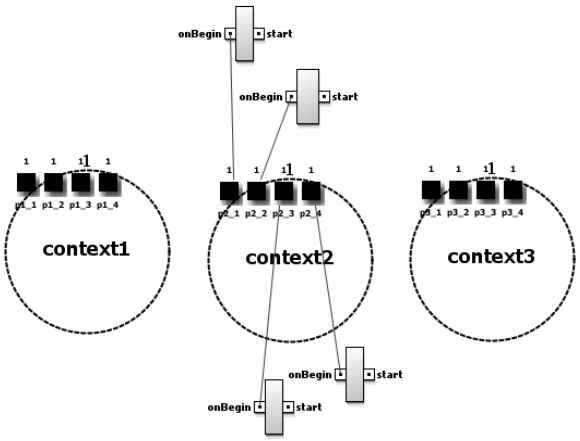
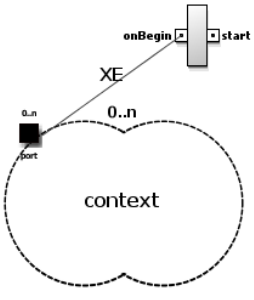
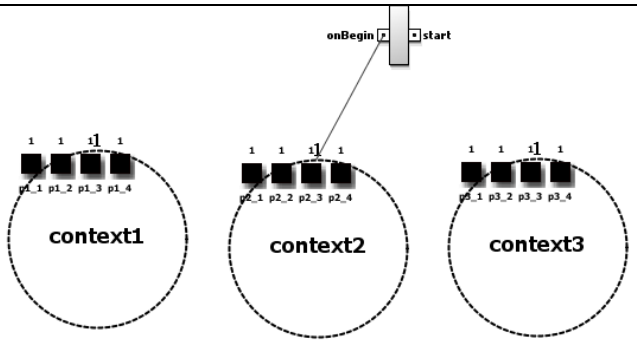
A Tabela 3 apresenta as possíveis combinações quando um *bind* é associado a um papel de condição. Neste caso, a janela não disponibiliza o uso das opções "All-i", "Next" e

"Prev" (*Previous*). Na terceira coluna da tabela, é exibido o resultado do documento final gerado se fosse feito o processamento de um documento que usasse um *template* com a lógica exibida no exemplo da segunda coluna da tabela. No exemplo, o documento que referencia o *template* possui três elementos do tipo contexto, cada um com quatro portas. Cada contexto do documento possui *xlabel* igual ao do componente "context" do *template*. Nos resultados da terceira coluna, quando é usada a opção "All", é gerado geralmente apenas um elo, com vários *binds*. Por outro lado, quando é usada a opção "i", são gerados vários elos. Uma observação importante deve ser feita quando é usada a opção "i" no *bind* de condição, a combinação final dependerá do tipo de opção de iteração colocada no(s) *bind*(s) de ação.

Tabela 3: Possíveis combinações em uma janela XPath com os campos "Component" e "Parent", onde um *bind* associa o componente a um papel de condição.

Combinação	Exemplo	Resultado esperado se existissem 3 contextos, cada um com 4 portas
All/All		
All/i		

All/XE		 XE especificando a segunda porta
i/All		
i/i		

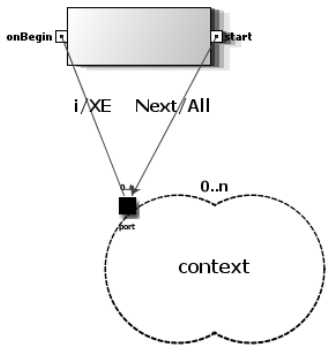
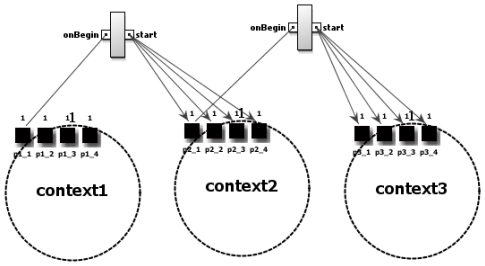
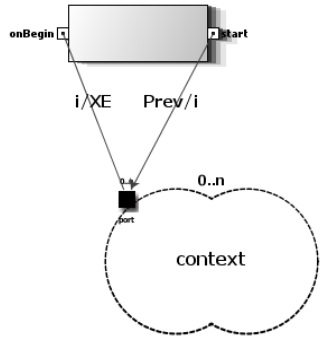
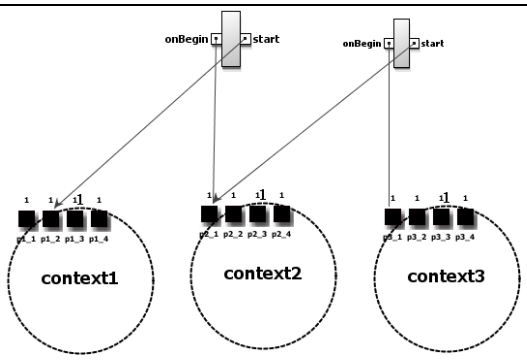
i/XE		 XE especificando a terceira porta
XE/All		 XE especificando o segundo contexto
XE/i		 XE especificando o segundo contexto
XE/XE ou XE		 XE especificando a terceira porta do segundo contexto

As possíveis combinações da janela com opções para “Component” e “Parent”, nos casos em que o *bind* estiver associado a um papel de ação, são exibidas a seguir, na seguinte notação: “operador / operador”, onde o operador antes de “/” é referente a “Parent” e o operador posterior é referente a “Component”:

- All/All; All/i e All/XE;
- All-i/All; All-i/i e All-i/XE;
- i/All; i/i e i/XE;
- Next/All; Next/i e Next/XE;
- Prev/All; Prev/i e Prev/XE;
- XE/All; XE/i e XE/XE ou (XE);

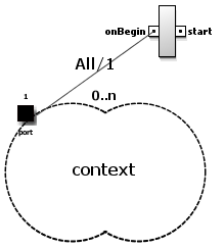
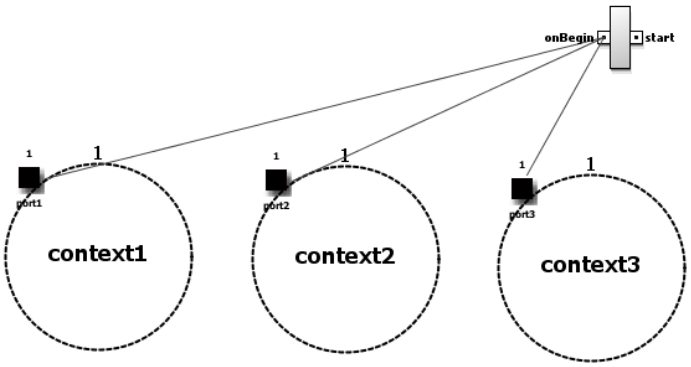
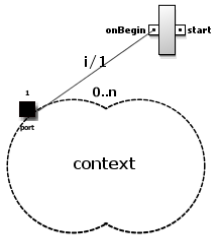
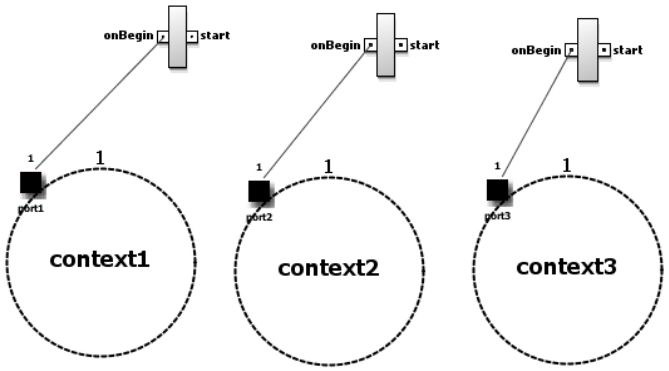
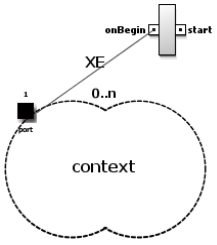
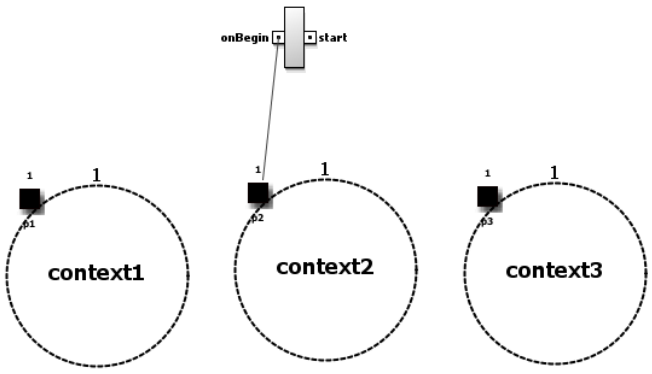
A Tabela 4 apresenta duas possíveis combinações quando um *bind* é associado a um papel de ação. Na terceira coluna da tabela, é exibido o resultado do documento final gerado se fosse feito o processamento de um documento que usasse um *template* com a lógica exibida no exemplo da segunda coluna da tabela. No exemplo, o documento que referencia o *template* possui três elementos do tipo contexto, cada um com quatro portas.

Tabela 4: Possíveis combinações em uma janela XPath com os campos “Component” e “Parent”, onde um *bind* associa o componente a um papel de ação.

Combinação	Exemplo	Resultado esperado se existissem 3 contextos, cada um com 4 portas
Next/All		 XE especificando a primeira porta
Prev/i		 XE especificando a primeira porta

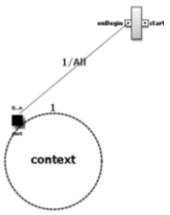
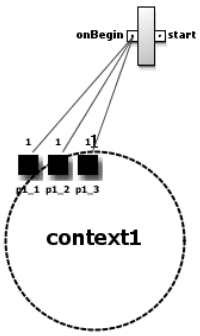
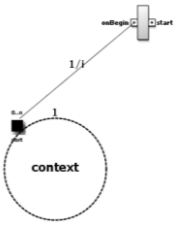
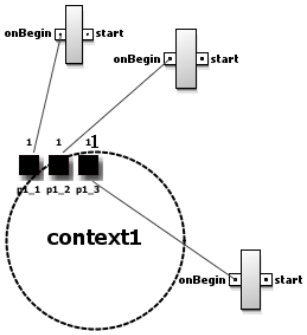
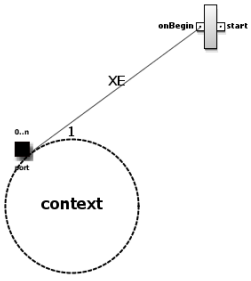
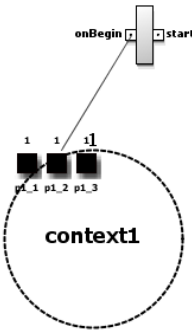
A Tabela 5 apresenta as combinações possíveis quando a janela aberta é similar à apresentada na Figura 37, em que o *bind* está associado a um papel de condição e o componente alvo do *bind* é um componente do tipo *port*, que representa apenas um elemento, e o pai (componente *parent*) de *port* é um elemento que representa um conjunto de elementos. Nos exemplos da Tabela 5, o documento que referencia o *template* possui três elementos do tipo contexto, cada um com uma porta.

Tabela 5: Possíveis combinações em uma janela XPath onde o componente ("port") representa apenas um elemento e um *bind* o associa a um papel de condição.

Combinação	Exemplo	Resultado esperado se existissem 3 contextos cada um com 1 porta
All/1		
i/1		
XE/1 ou (XE)		 XE especificando o segundo contexto

A Tabela 6 apresenta as combinações possíveis, na situação inversa da anterior, onde o componente do tipo *port* representa um conjunto de elementos e o pai de *port* representa apenas um elemento. Nos exemplos da Tabela 6, o documento que referencia o *template* possui um elemento do tipo contexto, que possui três portas.

Tabela 6: Possíveis combinações em uma janela XPath onde o pai do componente ("context") representa apenas um elemento e um *bind* o associa a um papel de condição.

Combinação	**Exemplo**	**Resultado esperado se existisse 1 contexto com 3 portas referenciando um *template* com lógica similar a coluna dois.**
1/All		
1/i		
1/XE ou XE		 XE especificando a segunda porta

As combinações possíveis nos casos em que o *bind* estiver associado a um papel de ação são:

- 1/All; 1/All-i; 1/i; 1/Prev; 1/Next e 1/XE;

- All/1; All-i/1; i/1; Prev/1; Next/1 e XE/1;

Quando a janela é aberta devido a dois cliques em um *port mapping* e se ele estiver associado a um elemento do tipo *port*, a interface que aparece é similar à apresentada na Figura 41 e as combinações possíveis são:

- All/All; All/i; All/XE;
- XE/All e XE/XE;

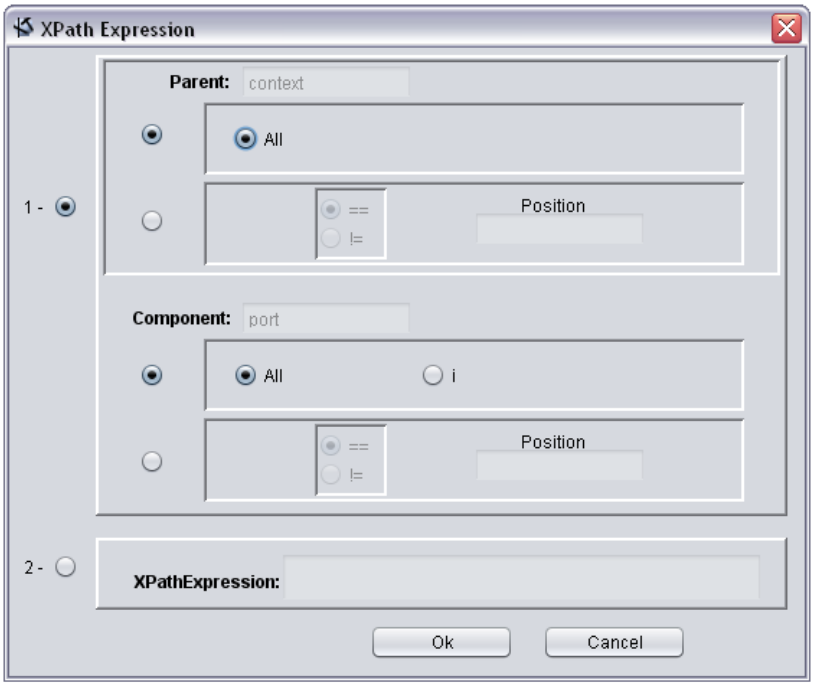
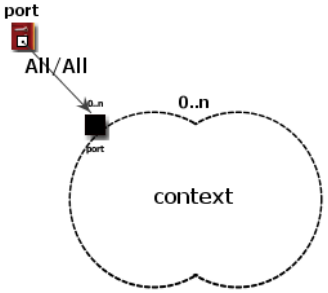
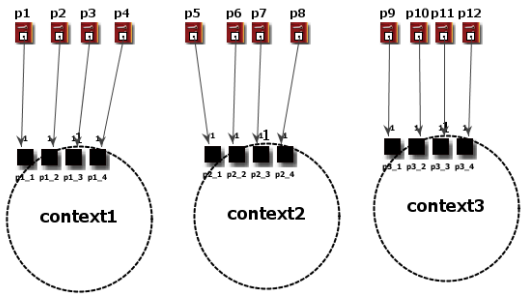
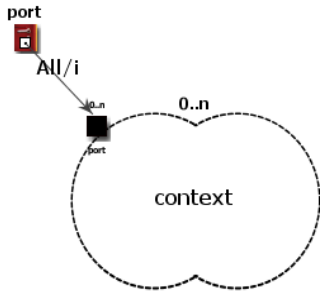
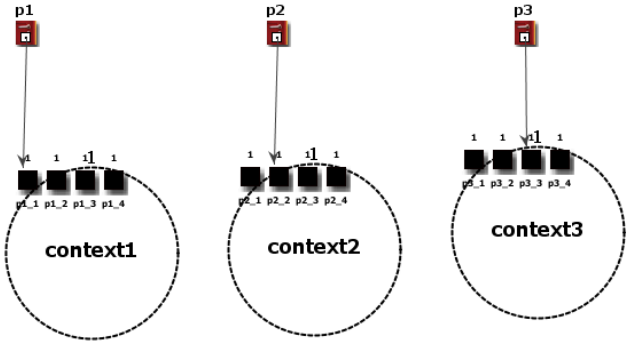
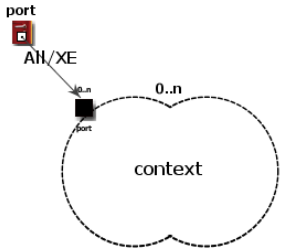
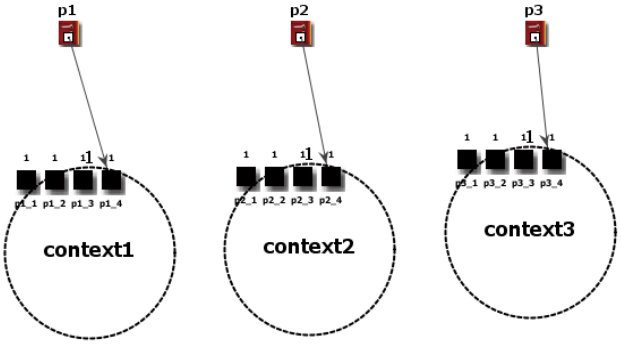
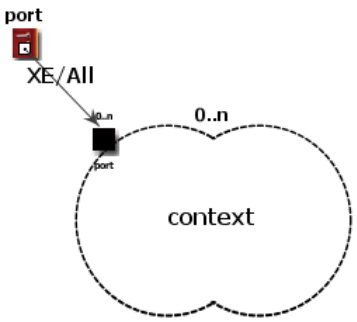
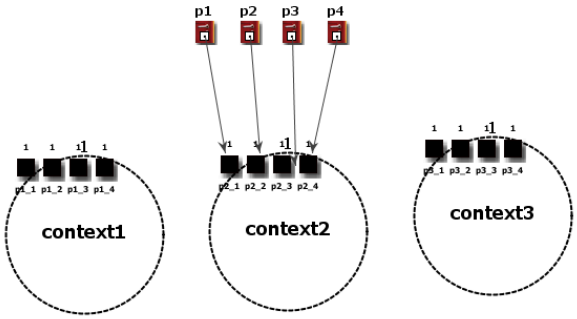
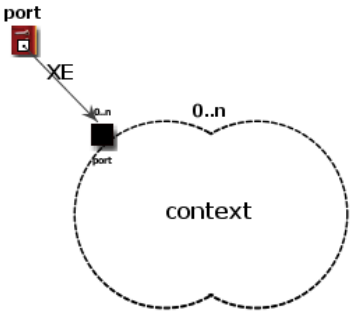
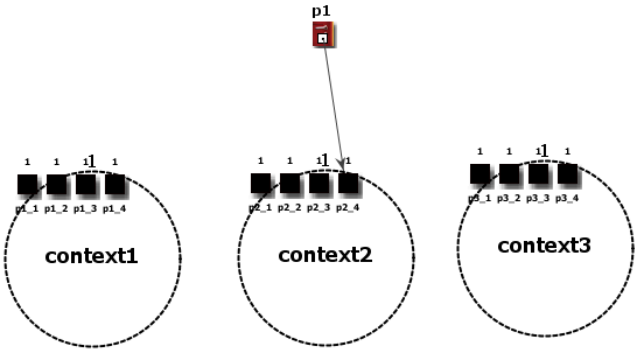


Figura 41: Interface XPath para port mapping.

A Tabela 7 detalha melhor as combinações possíveis da interface da Figura 41, considerando que o documento que referencia o *template* possui três elementos do tipo contexto, cada um com quatro portas.

Tabela 7: Possíveis combinações em uma janela XPath com os campos "Component" e Parent", onde o componente do tipo *port* está associado a um *port mapping*.

Combinação	Exemplo	Resultado esperado se existissem 3 contextos cada um com 4 portas
All/All		

All/i		
All/XE		 XE especificando a última porta.
XE/All		 XE especificando o segundo contexto
XE/XE ou XE		 XE especificando a última porta do segundo contexto

Analisando a interface gráfica para declarar expressões XPath básicas, as opções de generalização e suas possíveis combinações, nota-se que os usuários do ambiente gráfico têm muitas opções disponíveis para, de forma gráfica, construir diversos tipos de *templates*, inclusive, opções para a construção de *templates* mais complexos. Algumas combinações ainda estão em estudo para verificar a sua real importância de uso.

4.4 Exemplo de *template* construído com o uso das opções "Next" e "Prev" e com auxílio de expressões XPath declaradas de forma interativa

Nesta seção, são apresentados quatro *templates* usados para ilustrar como as opções de iteração e a interface XPath agilizam a construção de *templates*. Esses *templates* representam uma apresentação de slides. Neste exemplo, além de uma imagem de fundo comum a todas as telas (*slides*), cada tela contém uma mídia texto para o título, uma mídia texto para o texto do corpo e mídias imagem para os botões anterior e próximo slide. Além de apresentação de slides, os *templates* deste exemplo também podem ser usados como *templates* para manuais de instruções, livros infantis, etc. A Figura 42 apresenta a visão estrutural criada no EDITEC do *template* "Apresentação de slides" (tApreSlides).

No *template* dessa figura, existem três contextos, um para a primeira tela de apresentação (ctx1), outro para a última (ctxn) e um terceiro que representa um conjunto de telas intermediárias (ctxMeio, do segundo ao penúltimo contexto), localizados entre a primeira tela de apresentação e a última. A primeira e a última tela têm características próprias, já as intermediárias têm características semelhantes, permitindo assim o reúso da estrutura de código de um contexto nas diversas telas intermediárias que existirem. Antes de usar esse *template*, devem ser especificadas as mídias e quais os tipos de relacionamentos que vão existir em cada um dos três contextos internos (ctx1, ctxMeio e ctxn) ao contexto principal. Para isso, são especificados mais três *templates*, um para cada contexto.

Ainda na Figura 42, a porta "pInicio" de entrada do programa, aponta para o nó componente "background" que será o primeiro componente a ser exibido. Um elo que usa o conector *onBeginStart* interliga esse componente ao primeiro contexto (ctx1) em sua interface "pCtx1". Desse modo, assim que o programa iniciar, a imagem de fundo e a primeira tela de apresentação (ctx1) serão iniciadas.

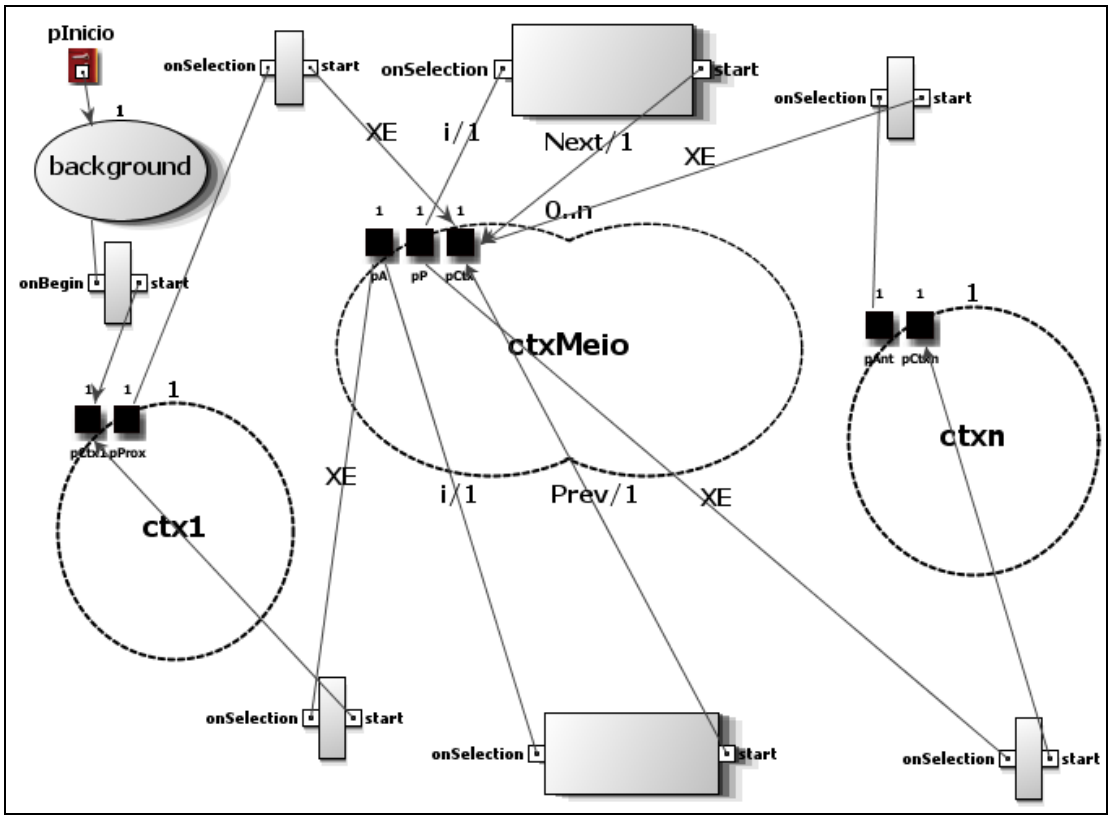


Figura 42: Visão estrutural do *template* "tApresSlides" do exemplo "Apresentação de Slides".

Um segundo elo (canto superior esquerdo da Figura 42) com conector *onSelectionStart* interliga o primeiro contexto (ctx1), através da interface "pProx", ao primeiro contexto do conjunto de contextos representados por "ctxMeio' na interface "pCtx". Para selecionar essa interface específica, é utilizada a expressão XPath "child::*[@xlabel = 'ctxMeio'][position() = 1]/child::*[@xlabel = 'pCtx']" construída através da janela XPath, como exibido na Figura 43. Essa janela é aberta clicando-se duas vezes no *bind* que interliga a interface "pCtx" ao papel *start* do conector mencionado acima. Esse *bind* recebe a notação "XE", que significa que existe uma expressão específica relacionada ele. A expressão pode ser visualizada na parte da visão estrutural referente a propriedades ou diretamente na visão textual no código XTemplate gerado. Um outro elo (canto inferior esquerdo da Figura 42) também com conector *onSelectionStart* cria uma relação semelhante a anterior só que de forma inversa, ele interliga a interface "pA"[3] do primeiro contexto do conjunto de contextos representados por "ctxMeio" ao contexto (ctx1) na interface pCtx. Nesse caso, é utilizada a expressão XPath "child::*[@xlabel = 'ctxMeio'][position() = 1]/child::*[@xlabel = 'pA']" para interligar a interface "pA" do primeiro contexto do conjunto ao papel *onSelection* do conector do elo. Essa expressão, ligeiramente diferente da apresentada anteriormente, também é

[3] "pA"de porta anterior

construída através da interface XPath. Relacionamentos similares são criados entre o último contexto do conjunto de contextos representados por “ctxMeio” e o contexto “ctxn”, última tela de apresentação.

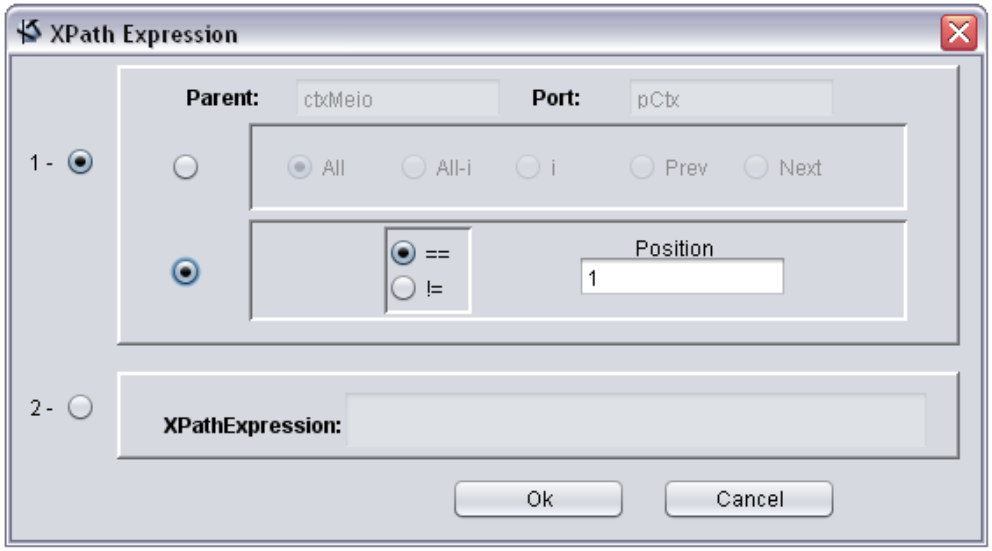


Figura 43: Janela XPath, exemplo “Apresentação de Slides’.

Para os contextos intermediários, representados por “ctxMeio”, são criados dois elos: um que usa o operador de iteração “Next” e outro que usa o operador “Prev” (*Previous*). Também é usada, no elo, a opção de iteração “i”. A interface da janela aberta para definir esses operadores é similar à apresentada na Figura 43. O primeiro elo (parte central superior da Figura 42) interliga a interface “pP”[4] do contexto “ctxMeio” à interface “pCtx” também do contexto “ctxMeio”. Nesse elo, no *bind* ligado a interface “pP”, é definido o operador de iteração “i”, e no *bind* ligado a interface “pCtx”, o operador “Next”. Assim, é criado um conjunto de elos que interligam de forma sequencial crescente os contextos intermediários representados por “ctxMeio”. O primeiro elo interliga a porta “pP”, do primeiro contexto do conjunto, à porta “pCtx”, do segundo contexto do conjunto; o segundo elo interliga a porta “pP”, do segundo contexto do conjunto, à porta “pCtx”, do terceiro contexto do conjunto; e assim sucessivamente. O objetivo dessa lógica é para que, quando o botão de próximo for selecionado, a tela de apresentação seja alterada da atual para a próxima. A Listagem 12 mostra, para este caso, o trecho de código gerado com o uso do operador “Next”.

[4] “pP” de porta posterior ou próxima porta

Listagem 12: Trecho de código XTemplate gerado com o uso do operador "Next".

```
<variable name="i" select="1"/>
<for-each select="child::*[@xlabel = 'ctxMeio'][position ()!= last()]/child::*[@xlabel = 'pP']">
    <link id="link2an-1" xtype="onSelectionStart">
        <bind role="onSelection" select="current()"/>
        <bind role="start" select="child::*[@xlabel='ctxMeio'][position() = $i+1]/child::*[@xlabel = 'pCtx']"/>
    </link>
    <variable name="i" select="$i + 1"/>
</for-each>
```

O segundo elo (parte central inferior da Figura 42) interliga a interface "pA" do contexto "ctxMeio" à interface "pCtx" também do contexto "ctxMeio". Neste segundo elo, no *bind* ligado a interface "pA", é definido o operador de iteração "i", e no *bind* ligado à interface "pCtx", o operador "Prev". Assim, é criado um conjunto de elos que interligam de forma sequencial decrescente os contextos intermediários representados por "ctxMeio". O primeiro elo interliga a porta "pA", do último contexto do conjunto, à porta "pCtx", do penúltimo contexto do conjunto. O segundo elo interliga a porta "pA", do penúltimo contexto do conjunto, à porta "pCtx", do antepenúltimo contexto do conjunto, e assim sucessivamente. A Listagem 13 mostra, para este caso, o trecho de código gerado com o uso do operador "Prev" e o Apêndice A.2.1 apresenta o código completo desse *template* (tApreSlides) "Apresentação de Slides".

Listagem 13: Trecho de código XTemplate gerado com o uso do operador "Prev".

```
<variable name="j" select="1"/>
<for-each select="child::*[@xlabel = 'ctxMeio'][position()!=last()]/child::*[@xlabel = 'pA']">
    <link id="linkn-1a2" xtype="onSelectionStart">
        <bind role="start" select="current()"/>
        <bind role="onSelection" select="child::*[@xlabel='ctxMeio'][position()=$j+1]/child::*[@xlabel='pCtx']"/>
    </link>
    <variable name="j" select="$j + 1"/>
</for-each>
```

O *template* "tApreSlides" possui os contextos "ctx1", "ctxMeio" e "ctxn" como componentes, ele especifica os relacionamentos entre esses componentes, mas não os relacionamentos que são internos a eles. Deste modo, é necessário definir mais três *templates*, um para cada contexto, que especificam esses relacionamentos internos.

Para o primeiro contexto (ctx1) é especificado o *template* "tCtx1", exibido na Figura 44. Neste *template*, existe uma porta principal "pCtx1", definida para ser acionada quando o contexto "ctx1" for inicializado. Um *port mapping* faz o mapeamento dessa porta para o componente "buttonProx". Na autoria de um *template*, na criação de portas do *template*, o usuário, além de atribuir um identificador para uma porta, também pode especificar um *xlabel* para ela. Assim, depois que o contexto é processado e a porta é criada, ela já recebe um *xlabel*, inserido pelo próprio *template.* Usando esse *xlabel*, essa porta pode ser referenciada por um contexto externo. Isso é muito interessante em um processamento de contextos aninhados. De outra forma, essa especificação só seria possível depois que o contexto fosse processado. O usuário teria que processar o contexto interno, alterar manualmente o código criado e inserir um *xlabel* na porta criada para que a porta pudesse ser referenciada por um contexto externo, pois as portas do contexto NCL só são criadas após o processamento do contexto. Somente depois disso, ele poderia processar o contexto externo.

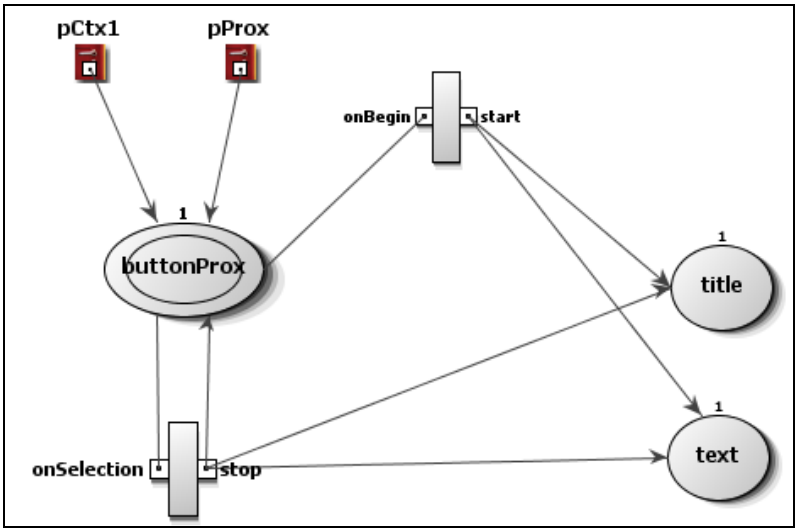


Figura 44: Visão Estrutural do *template* "tCtx1" usado no primeiro contexto (ctx1) do *template* "tApresSlides" do exemplo "Apresentação de Slides".

Na Figura 44, o símbolo de duas circunferências concêntricas em "buttonProx" significa que é um objeto de mídia criado pelo próprio *template*. Deste modo, não é preciso definir uma mídia para o componente "buttonProx" no documento que referencia o *template.* Isso é feito no EDITEC pressionando o mouse na imagem "media" da barra de ferramentas da aba de corpo da visão estrutural e soltando a "media" em cima do componente "buttonProx". Feito isso, é aberta uma janela onde o usuário indica a URI onde a mídia se encontra. Quando "buttonProx" é iniciado, simultaneamente, também são iniciados o título e o texto de corpo do slide. Quando o "buttonProx" é selecionado, o título, o texto de corpo e o próprio "buttonProx" são finalizados. Deste modo, o contexto "ctx1" é finalizado. O evento de

seleção em “buttonProx” pode disparar elos, externos ao contexto “ctx1”, interligando as portas (pCtx1 e pProx) que são mapeadas para “buttonProx”. Assim, elementos externos a esse contexto podem saber o que se passa com “buttonProx”. Um contexto que usa o *template* “ctx1” será usado como o componente “ctx1” do *template* “tApresSlides”. O Apêndice A.2.2 apresenta o código do *template* “tCtx1”.

Para o último contexto “ctxn”, é especificado o *template* “tCtxn”, apresentado na Figura 45. Os relacionamentos internos desse *template* são similares ao do *template* “tCtx1”. Um contexto que usa o *template* “tCtxn” será usado como o componente “ctxn” do *template* “tApresSlides”. O Apêndice A.2.3 apresenta o código do *template* “tCtxn”.

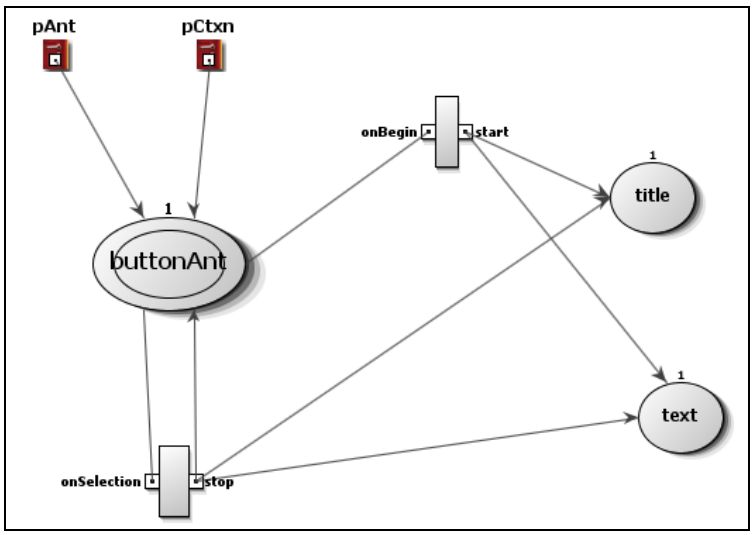


Figura 45: Visão Estrutural do *template* “tCtxn” usado no último contexto (ctxn) do exemplo “Apresentação de Slides”.

Na Figura 46, é apresentado o *template* para o contexto “ctxMeio”. Nesse *template*, existe uma porta principal “pCtx”, definida para ser acionada quando o contexto intermediário atual for iniciado. Um *port mapping* faz o mapeamento dessa porta para o componente “buttonAnt”. Quando “buttonAnt” é iniciado, simultaneamente, também são iniciados o componente “buttonProx”, o título e o texto de corpo do slide (elo do centro da Figura 46). Quando o “buttonAnt” ou “buttonProx” são selecionados, o título, texto de corpo, “buttonAnt” e “buttonProx” são finalizados. Deste modo, o contexto intermediário atual é finalizado. Nesse *template*, as mídias “buttonAnt” e “buttonProx” não são inseridos pelo uso do *template*, sendo que o usuário deve especificar essas mídias no contexto NCL que referencia esse *template*, caso contrário teríamos várias mídias inseridas com o mesmo id. Uma atualização no processador de templates deve ser feita para contornar esse problema. O Apêndice A.2.4 apresenta o código do *template* “tCtx2aN_1”.

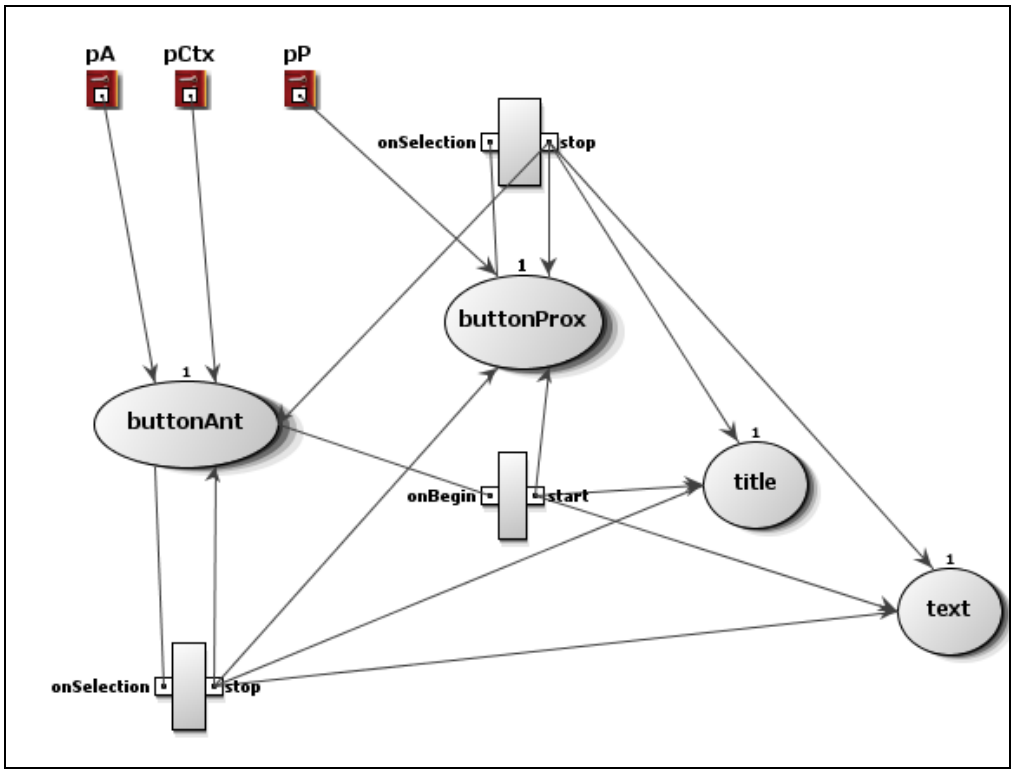


Figura 46: Visão Estrutural do *template* "tCtx2aN_1" usado nos contextos intermediários (ctx 2 a n-1) do exemplo "Apresentação de Slides".

A Listagem 4 apresenta um exemplo de documento que referencia os *templates* "tCtx1", "tCtx2aN_1", "tCtxn" e "tApreSlides" detalhados nesta seção. O Apêndice A.2.5 apresenta o código do documento final depois de processado.

Listagem 14: Representação de um documento que usa os *templates* "tCtx1", "tCtx2aN_1", "tCtxn" e "tApreSlides" do exemplo "Apresentação de Slides".

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <ncl id="apreSlides" xmlns="http://www.midiacom.uff.br/gtvd/XTemplate30/EXTProfile">
3.    <head>
4.      <templateBase>
5.              <!--importacao dos templates-->
6.              <importBase documentURI="tApreSlides.xml" alias="tApreSlides"/>
7.              <importBase documentURI="tCtx1.xml" alias="tCtx1"/>
8.              <importBase documentURI="tCtx2aN_1.xml" alias="tCtx2aN_1"/>
9.              <importBase documentURI="tCtxn.xml" alias="tCtxn"/>
10.     </templateBase>
11.   </head>
12.
13.    <!--composicao referenciando o apelido 'tApreSlides' do template atraves do 'atributo xtemplate' -->
14.   <body xtemplate="tApreSlides">
15.     <media id="background" src="media/background.gif" type="image/gif"  xlabel="background"/>

16.     <!--composicao referenciando o apelido 'tCtx1' do template atraves do 'atributo xtemplate' -->
17.     <context id="ctx1" xtemplate="tCtx1" xlabel="ctx1" >
18.             <media id="title1" src="media/title1.html" type="text/html"  xlabel="title"/>
19.             <media id="text1" src="media/text1.html" type="text/html"  xlabel="text"/>
```

```
20.       </context>
21.
22.       <!--composicao referenciando o apelido 'tCtx2aN' do template atraves do 'atributo xtemplate' -->
23.       <context id="ctx2aN_1_1" xtemplate="tCtx2aN_1" xlabel="ctxMeio" >
24.                 <media id="title2" src="media/title2.html" type="text/html"  xlabel="title"/>
25.                 <media id="text2" src="media/text2.html" type="text/html"  xlabel="text"/>
26.                 <media id="buttonAnt2" src="media/buttonAnt.gif" type="image/gif" xlabel="buttonAnt"/>
27.                 <media id="buttonProx2" src="media/buttonProx.gif" type="image/gif"
                              xlabel="buttonProx"/>
28.       </context>

29.       <!--composicao referenciando o apelido 'tCtx2aN' do template atraves do 'atributo xtemplate' -->
30.       <context id="ctx2aN_1_2" xtemplate="tCtx2aN_1" xlabel="ctxMeio" >
31.                 <media id="title3" src="media/title3.html" type="text/html"  xlabel="title"/>
32.                 <media id="text3" src="media/text3.html" type="text/html"  xlabel="text"/>
33.                 <media id="buttonAnt3" src="media/buttonAnt.gif" type="image/gif" xlabel="buttonAnt"/>
34.                 <media id="buttonProx3" src="media/buttonProx.gif" type="image/gif"
                              xlabel="buttonProx"/>
35.       </context>

36.       <!--composicao referenciando o apelido 'tCtx2aN_1' do template atraves do 'atributo xtemplate' -->
37.       <context id="ctx2aN_1_3" xtemplate="tCtx2aN_1" xlabel="ctxMeio" >
38.                 <media id="title4" src="media/title4.html" type="text/html"  xlabel="title"/>
39.                 <media id="text4" src="media/text4.html" type="text/html"  xlabel="text"/>
40.                 <media id="buttonAnt4" src="media/buttonAnt.gif" type="image/gif" xlabel="buttonAnt"/>
41.                 <media id="buttonProx4" src="media/buttonProx.gif" type="image/gif"
                              xlabel="buttonProx"/>
42.       </context>
43.
44.       <!--composicao referenciando o apelido 'tCtxn' do template atraves do 'atributo xtemplate' -->
45.       <context id="ctxn" xtemplate="tCtxn" xlabel="ctxn" >
46.                 <media id="titleN" src="media/titleN.html" type="text/html"  xlabel="title"/>
47.                 <media id="textN" src="media/textN.html" type="text/html"  xlabel="text"/>
48.       </context>
49.    </body>
50. </ncl>
```

Nas linhas 6 a 9 ocorre a importação dos *templates*, com isso, as composições podem referenciá-los através de seus apelidos (atributo *alias*) e identificar seus componentes usando seus rótulos (atributo *xlabel*). Na linha 14 da Listagem 4, o contexto *body* faz uso do *template* "tApreSlides", já na linha 17, o contexto "ctx1" faz uso do *template* "tCtx1", nas linhas 23, 30 e 37 são definidos os contextos intermediários que fazem referência ao *template* "ctxMeio" e por último na linha 45, o contexto "ctxn" faz uso do *template* "tCtxn". Note que, em NCL, o corpo e um contexto (elementos *body* e *context*) definem composições.

No primeiro contexto "ctx1", os *xlabels* "title" e "text", definidos para as mídias "title1" e "text1" respectivamente, são usados no processamento do contexto. Após o

processamento desse contexto, são criados os relacionamentos e outros componentes internos do contexto. São criadas, entre outros elementos, as portas (portas com ids “pCtx1” e “pProx”) para a mídia “buttonProx” (mídia inserida pelo próprio *template*). Essas portas recebem, depois de criadas, os *xlabels* “pCtx1” e “pProx” que são utilizados no processamento do contexto externo *body*, pois na criação do template “tCtx1” foi especificado que essas portas receberiam esses *xlabels* (A.2.2, linhas 26 e 28). Assim, o usuário não precisa se preocupar em especificar esses *xlabels*. Esses *xlabels* não podem ser especificados inicialmente, como feitos em “title” e “text” (linhas 18 e 19 da Listagem 4), pois as portas do contexto “ctx1” só vão existir depois que o contexto for processado. Durante a criação dos contextos, já era previsto o uso dessas portas no processamento do contexto externo *body* que usa o *template* “tApreSlides”. Uma determinada porta com um *xlabel*, no processamento do contexto *body*, recebe a mesma lógica especificada para o elemento do *template* “tApreSlides”, que tem esse *xlabel*.

A especificação das mídias dos demais contextos e dos *xlabels* dessas mídias são similares ao explicado para o primeiro contexto “ctx1”. A única diferença é que nos contextos intermediários os botões “buttonAnt” e “buttonProx” devem ser especificados pelos usuários.

Neste exemplo, o conteúdo das bases foi inserido em um só arquivo, tendo assim, apenas um arquivo de base de conectores e um só de base de apresentação.

Capítulo 5

Implementação

Este capítulo apresenta uma visão geral da implementação das principais partes do ambiente EDITEC, que é desenvolvido em linguagem Java (Deitel, 2008; Sun, 2010). A escolha de Java oferece portabilidade, permitindo o uso do editor em diferentes plataformas.

O capítulo também comenta algumas funcionalidades ainda não disponíveis na implementação atual que devem ser acrescentadas futuramente.

Além da visão geral da implementação, este capítulo também apresenta um estudo de usabilidade da ferramenta EDITEC, realizado com um grupo de treze usuários com nível de conhecimento variado em NCL e XTemplate, incluindo usuários avançados e iniciantes. O objetivo do estudo foi traçar, através de um questionário, indicadores de facilidade de uso do editor e verificar se ele facilita o entendimento dos conceitos da linguagem XTemplate.

5.1 Arquitetura

Como apresentado inicialmente no Capítulo 4, o editor está dividido em seis pacotes principais: *programs*, *layout*, *structural*, *textual*, *xtemplate e common*. O papel de cada pacote é apresentado a seguir:

- *layout*, *structural* e *textual*: são pacotes referentes às visões disponíveis no editor. Cada pacote agrega os principais elementos da visão que ele representa. Também existem dentro de alguns desses pacotes outros pacotes. Por exemplo, *layout* e *structural* possuem, cada um, um pacote chamado *windows*, que reúne as janelas relativas a cada visão.
- *programs*: pacote que possui classes relativas a interface principal do ambiente de autoria EDITEC. É neste pacote que se encontra a classe “Editor”, responsável pela integração de todo o ambiente de autoria. A Figura 47 apresenta o diagrama de dependências dessa classe.

- *xtemplate*: pacote que reflete a estrutura da linguagem XTemplate, onde os elementos da linguagem são representados em classes Java, este pacote é utilizado pelas diversas visões do EDITEC.
- *common*: pacote que agrega as classes de elementos comuns das principais visões, como por exemplo, a classe "Overview" responsável pela visão em miniatura da área de trabalho das visões de leiaute e estrutural.

Na Figura 47, a classe "Editor" estende "JFrame" do pacote *swing*, que é uma API Java para interfaces gráficas (Sun, 2008a).

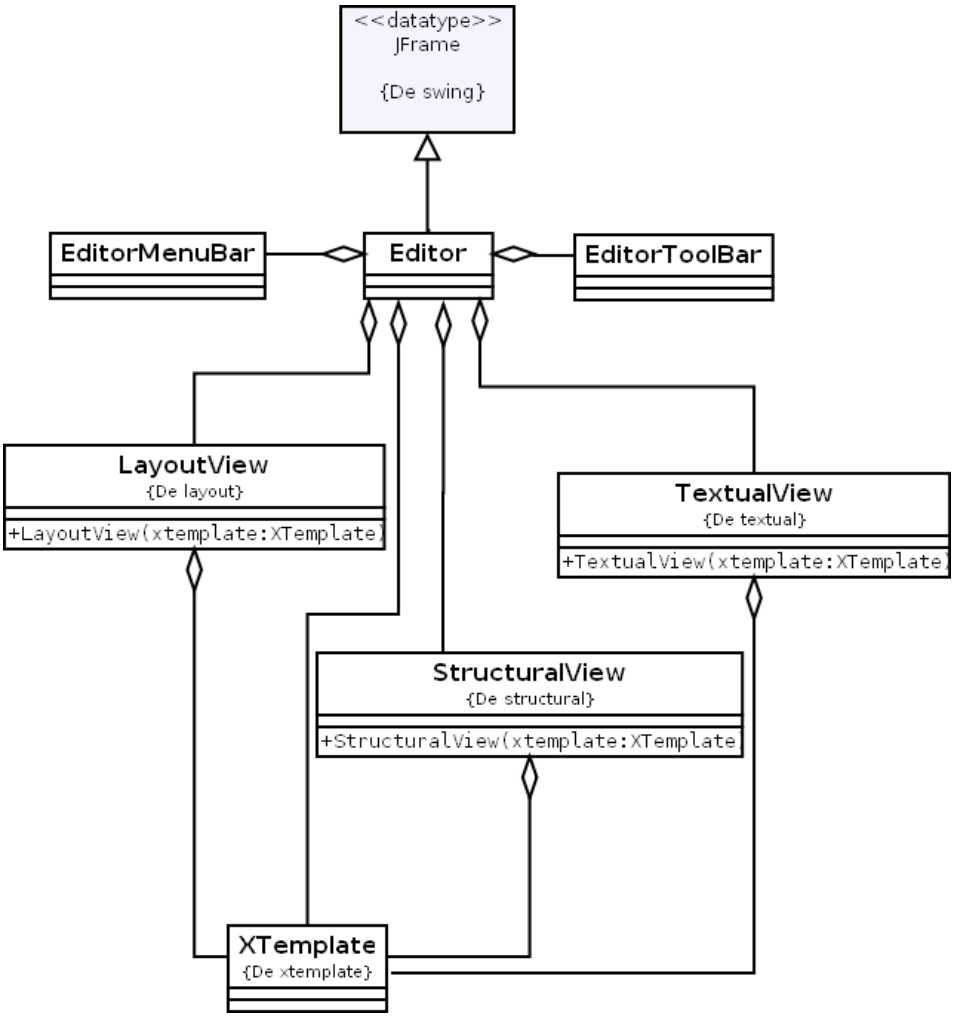


Figura 47: Diagrama de dependências da classe "Editor" do pacote *programs*.

A aparência e o comportamento de uma GUI *swing* podem ser uniformes em todas as plataformas em que um programa Java executa ou a GUI pode ser personalizada utilizando aparência e comportamento plugáveis (PLAF – *pluggable look-and-feel*) do *swing*. As imagens do ambiente EDITEC exibidas nesta dissertação são uniformes em todas as plataformas. O EDITEC permite através do menu Windows, escolher outras aparências para

interface. As opções disponíveis são listadas de forma automática de acordo com a plataforma em que ele executa. Algumas são comuns a todas as plataformas e outras são específicas a cada plataforma.

Ainda na Figura 47, um objeto “Editor” instancia um objeto “XTemplate”, que representa o documento XTemplate, e o repassa para as diversas visões presentes no editor. A Figura 48 apresenta o diagrama de classes do pacote *xtemplate*. A classe XTemplate é a principal classe desse pacote. Existe um outro pacote dentro do pacote *xtemplate* que é o pacote *presentation*. Neste pacote, encontram-se os elementos relativos às bases de descritores, regiões e regras.

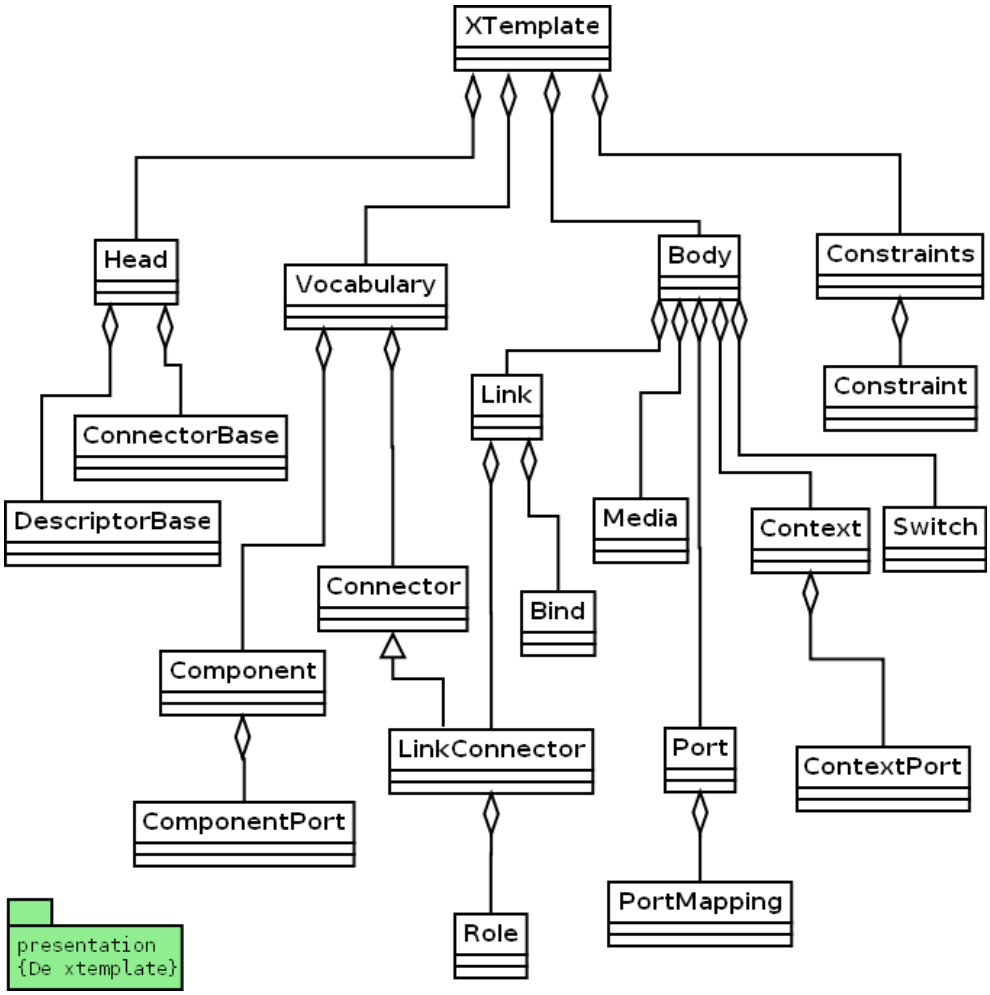


Figura 48: Diagrama de classes do pacote *xtemplate.*

5.1.1 Visão Estrutural

A Figura 49 apresenta o diagrama de classes da visão estrutural. As principais classes usadas nessa visão são:

- *StructuralView*: responsável pela integração dos elementos da visão estrutural.
- *VocabularyTree*: os elementos que são criados na subvisão de vocabulário são inseridos numa árvore de elementos do vocabulário. Esta classe é a responsável pela exibição gráfica desses elementos.
- *Overview*: classe do pacote *common*, responsável pela exibição de uma visão em miniatura da área de trabalho.
- *DrawArea_StructuralView*: área onde os elementos da visão estrutural são desenhados.
- *StrcturalToolBar*: esta classe engloba as barras de ferramentas das duas subvisões (de vocabulário e de corpo) da visão estrutural. As barras de ferramentas são exibidas de acordo com a aba selecionada pelo usuário, uma alteração de aba altera também os elementos que são exibidos na área de desenho (*DrawArea_StructuralView*). Quando um determinado botão da barra de ferramentas é selecionado e solto na área de trabalho, uma janela, do pacote *windows* (*structural.windows*), relativa ao elemento selecionado é aberta.
- *PropertiesInterface*: responsável pela exibição das propriedades do elemento da visão estrutural, quando este é selecionado.

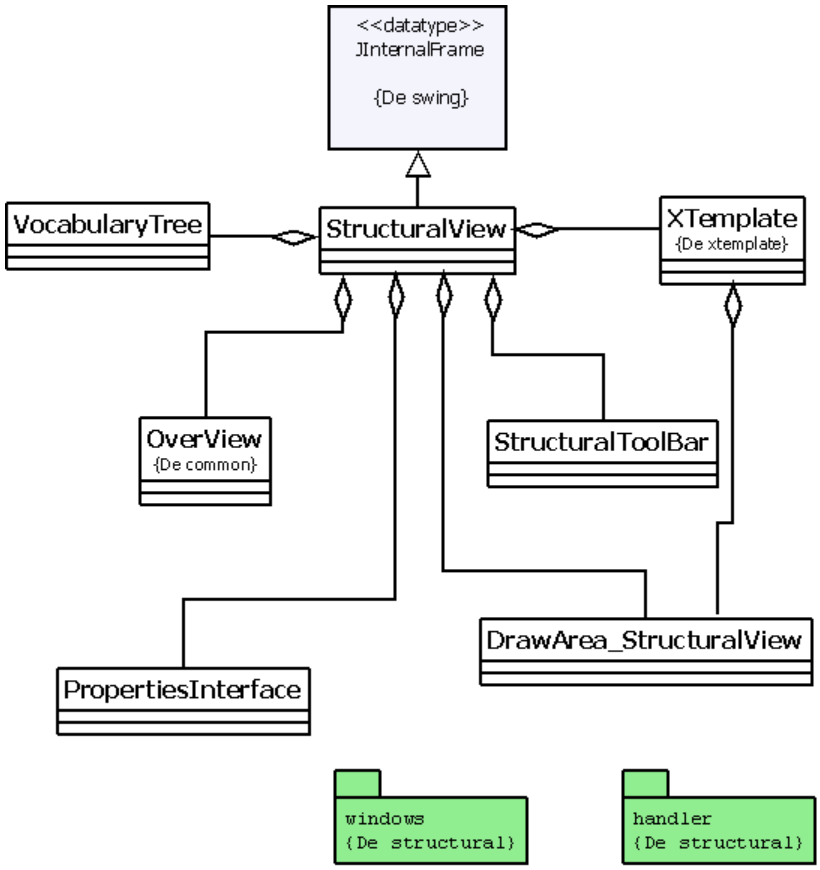


Figura 49: Diagrama de classes da visão estrutural.

Quando um objeto é criado, ele é desenhado na tela, e quando ele é movido ou redimensionado, o desenho é atualizado. Cada objeto possui um método de desenho que é chamado quando o objeto precisa ser desenhado na tela. Para desenhar os elementos na tela são usadas as capacidades gráficas de desenho do Java. O Java contém várias capacidades gráficas como parte da API do Java 2D (Sun, 2008b). Podem ser citadas, capacidades gráficas para desenhar formas bidimensionais, controlar cores e fontes, desenhar imagens gráficas em componentes. Além disso, existem ainda capacidades mais poderosas, como controlar o estilo das linhas utilizado para desenhar formas, e a maneira como formas são preenchidas com cores e padrões, por exemplo, usando um gradiente de cor.

Para tratar eventos desta visão, existe no pacote *structural*, além do pacote interno *windows* já mencionado, um outro pacote que é o pacote *handler* (*structural.handler*). Ele é responsável pelo tratamento dos eventos que ocorrem com os elementos da visão estrutural, quando esses elementos são selecionados, apagados, movidos ou redimensionados. Os componentes GUI podem gerar uma variedade de eventos em reposta às interações de usuário. Quando um evento ocorre, ele é despachado apenas para os ouvintes de evento do tipo apropriado. Cada componente GUI dá suporte a vários tipos de evento, como eventos de mouse, eventos de teclado, entre outros.

5.1.2 Visão de Layout

A Figura 50 mostra o diagrama de classes da visão de leiaute.

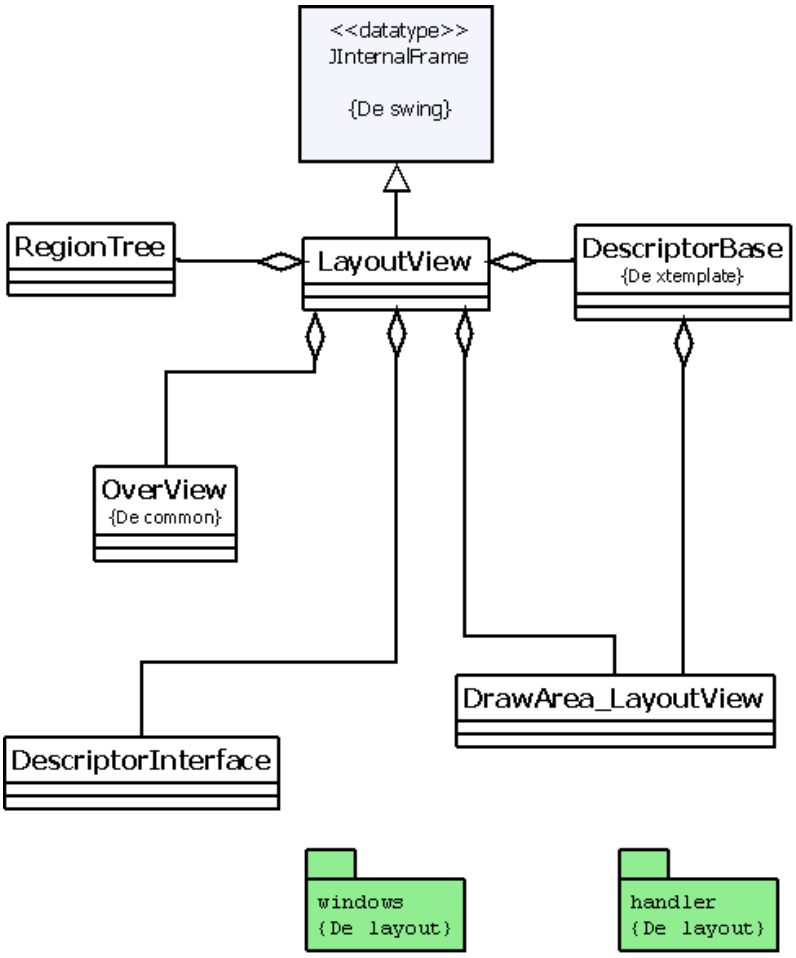


Figura 50: Diagrama de classes da visão de leiaute.

As principais classes usadas nessa visão são:

- *LayoutView*: responsável pela integração dos elementos da visão de leiaute.
- *DrawArea_LayoutView*: área onde os elementos da visão de leiaute são desenhados.
- *Overview*: classe do pacote *common*, responsável pela exibição de uma visão em miniatura da área de trabalho.
- *RegionTree*: as regiões criadas na área de trabalho são inseridas em uma árvore de regiões. Esta classe é a responsável pela exibição gráfica dessa árvore.
- *DescriptorInterface*: responsável pela criação e exibição dos descritores listados no canto inferior esquerdo da visão de leiaute. Na criação de descritores é possível referenciar uma região já criada.

Similar ao que ocorre na visão estrutural, para tratar eventos desta visão, existe no pacote *layout*, além do pacote interno *windows*, um outro pacote que é o pacote *handler*

(*layout.handler*) responsável pelo tratamento dos eventos que ocorrem com os elementos desta visão, quando estes são selecionados, apagados, movidos ou redimensionados.

5.1.3 Visão Textual

Na visão textual, a classe mais importante é a classe *CodeProcessor*. Ela recebe um objeto *XTemplate* e o percorre gerando o código do *template*. Essa classe utiliza o pacote JAXP (*Java API for XML Processing*) (Sun, 2002), que fornece uma biblioteca com várias funções para processamento de documentos baseados em XML. No processamento do código, é utilizado o modelo de documento DOM (*Document Object Model*) (DOM, 2000), padronizado pelo W3C, para manipular a árvore XML contendo os elementos do documento.

No processamento do código quando chega o momento de processar os elos (elemento *link*), faz-se uso de uma outra classe chamada *LinkProcessor*. Isso é feito devido ao fato do processamento de elos ser uma tarefa um pouco mais complexa, pois ocorre uma conversão da representação visual dos elementos, com diversas possibilidades de combinações de opções de iteração, em código textual. Assim, fica a cargo dessa classe analisar as diversas combinações de opções e gerar o código XTemplate correspondente.

5.2 Limitações da Implementação Atual

Um ponto do editor que ainda precisa ser tratado é o suporte à criação de nós de *switch* através da interface gráfica. Outro ponto é a possibilidade de abrir um arquivo de texto XTemplate no EDITEC. Atualmente o editor salva o *template* criado como um arquivo de projeto EDITEC em uma pasta juntamente com o arquivo XML do *template* criado, um arquivo contendo a base de conectores, com os conectores usados no *template* e um quarto arquivo com as bases de apresentação que pode conter as bases de descritores, de regiões e regras. O usuário só pode abrir, na versão atual do editor, o arquivo de projeto criado e salvo no EDITEC e o arquivo com as bases de apresentação, não sendo possível a abertura de um arquivo XTemplate escrito de forma textual. Um arquivo textual XTemplate não contém informações suficientes para a correta exibição do *template* no EDITEC, como, por exemplo, o posicionamento espacial dos elementos na visão estrutural. Isso poderia ser tratado criando de forma aleatória (ou usando algum algoritmo) o posicionamento dos elementos na visão

estrutural. Também deve ser analisada, quando se importa um arquivo textual XTemplate, as opções de importação de base de conectores e descritores, pois o arquivo textual XTemplate pode conter URIs (*Uniform Resource Identifier*) para outros arquivos onde essas bases se encontram.

Um outra funcionalidade ainda não implementada é a importação para o ambiente de edição de uma base de conectores. O EDITEC possui uma base de conectores interna, com uma grande variedade de conectores, mas o usuário pode querer criar uma base de conectores com um conector mais específico e então importar essa base criada para ser usada no ambiente EDITEC. Isso pode ser útil em situações em que a lógica requerida pelo usuário não é atendida usando apenas um conector dos conectores disponíveis na base de conectores do editor. Assim, o usuário evita, nessas situações, ter que usar mais de um elo/conector para montar a sua lógica desejada.

5.3 Estudo inicial de usabilidade do EDITEC

Esta seção apresenta um estudo de usabilidade da ferramenta EDITEC. O estudo foi realizado com um grupo de treze usuários com nível de conhecimento variado em NCL e XTemplate, incluindo usuários avançados e iniciantes. O editor foi usado em uma aula prática sobre XTemplate pela turma de mestrado da disciplina Fundamentos de Sistemas Multimídia e por um aluno de iniciação científica em Engenharia de Telecomunicações. A turma de mestrado era composta por alunos de mestrado em Computação e Engenharia de Telecomunicações da UFF. O objetivo do estudo era traçar, através de um questionário, indicadores de facilidade de uso do editor e se ele facilitava o entendimento dos conceitos da linguagem XTemplate. Adicionalmente, o estudo coletou sugestões de possibilidades de melhoria da interface do editor.

Em (Soares Neto, 2008), é apresentada uma metodologia similar à abordada nesta seção. Um questionário foi passado para alunos com diferentes níveis de conhecimento em programação e TV digital interativa (TVDI), com o objetivo de avaliar pontos em que o perfil de NCL poderia e deveria ser aprimorado para facilitar as atividades de criadores de conteúdos para TVDI, que podem ter uma formação bastante heterogênea.

No estudo de usabilidade do EDITEC, inicialmente, foi feita uma breve apresentação da linguagem XTemplate, explicando qual era o propósito da linguagem, a estruturação de um documento XTemplate e a geração de um documento NCL completo a partir de um

documento mais simples que referencia um *template*. Em seguida foi apresentado o EDITEC, para que os alunos pudessem interagir com a ferramenta e criassem alguns *templates*. Foram apresentados três exemplos de *templates* como propostas de tarefas a serem feitas. Dois *templates* similares às composições sequencial e paralela do SMIL, que foram feitos de forma guiada e um terceiro similar ao *template* vídeo com legendas apresentado no Capítulo 3. Este último foi apresentado como um desafio para que os alunos fizessem sozinhos, para avaliar o grau de conhecimento adquirido com o uso da ferramenta. No final da aula, foi passado para os alunos um pequeno questionário sobre a usabilidade do EDITEC com algumas perguntas que eles preencheram de forma anônima. O resultado desse questionário é apresentado nas Tabelas 8 e 9. O objetivo do questionário foi extrair a opinião dos alunos sobre o uso da ferramenta. O questionário também apresentava no final um campo para Sugestões e Críticas, que forneceram uma base adicional de informações sobre alguns pontos que poderiam ser melhorados no editor.

Tabela 8: Avaliação de usabilidade do EDITEC, parte1.

Perguntas	Alunos						
	1	2	3	4	5	6	7
1- Nível de conhecimento sobre NCL	Iniciante	Iniciante	Iniciante	Iniciante	Iniciante	Avançado	Iniciante
2- Nível de conhecimento sobre XTemplate	Iniciante	Iniciante	Iniciante	Iniciante	Iniciante	Iniciante	Iniciante
3- Considera a interface do EDITEC simples e fácil de usar	Sim	Sim	Sim	Sim	Não	Sim	Sim
4- Considera que o uso do EDITEC facilita a criação de *templates*, comparada a edição textual	Sim	Sim	Sim	Sim	Sim	Sim	Sim
5- Considera que o EDITEC ajuda a entender os conceitos da linguagem XTemplate	Sim	Sim	Sim	Sim	Não	Sim	Sim
6- Conseguiu concluir a criação dos *templates* usando a ferramenta	Sim	Sim	Sim	Sim	Sim	Sim	Sim

Tabela 9: Avaliação de usabilidade do EDITEC, parte2.

Perguntas	Alunos					
	8	9	10	11	12	13
1- Nível de conhecimento sobre NCL	Iniciante	Iniciante	Iniciante	Intermediário	Avançado	Iniciante
2- Nível de conhecimento sobre XTemplate	Iniciante	Iniciante	Iniciante	Intermediário	Avançado	Iniciante
3- Considera a interface do EDITEC simples e fácil de usar	Sim	Sim	Sim	Sim	Sim	Sim
4- Considera que o uso do EDITEC facilita a criação de *templates*, comparada a edição textual	Sim	Sim	Sim	Sim	Sim	Sim
5- Considera que o EDITEC ajuda a entender os conceitos da linguagem XTemplate	Sim	Sim	Sim	Não	Sim	Sim
6- Conseguiu concluir a criação dos *templates* usando a ferramenta	Sim	Sim	Sim	Sim	Sim	Sim

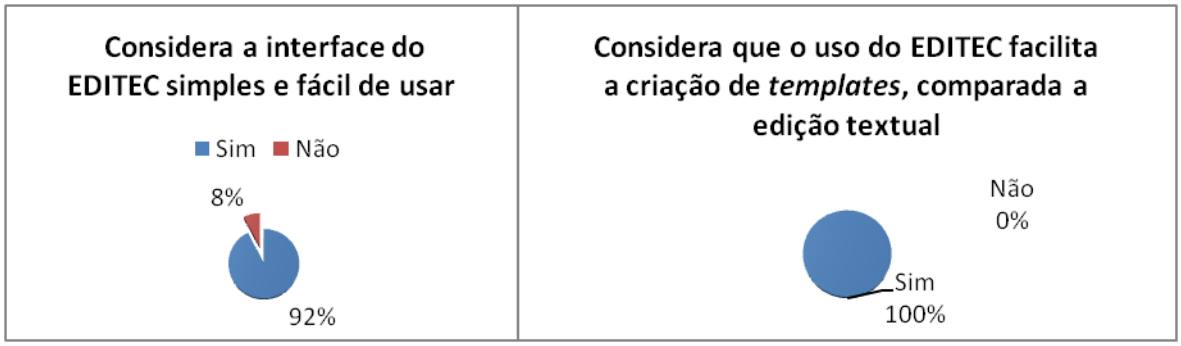


Figura 51: Gráficos relacionados as perguntas 3 e 4 das Tabelas 8 e 9.

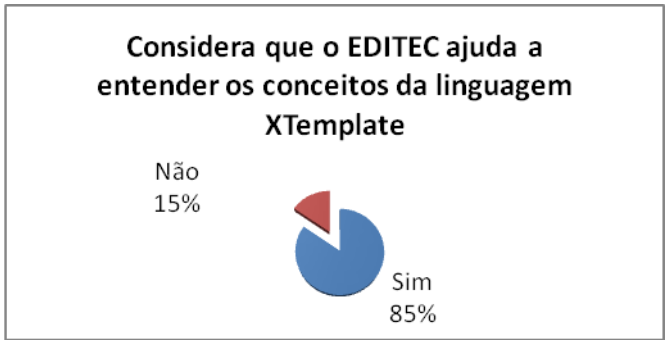


Figura 52: Gráfico relacionado a pergunta 4 das Tabelas 8 e 9.

De acordo com as Tabelas 8 e 9 e as Figuras 51 e 52, a maioria dos usuários achou que o editor facilitava a criação de *templates* comparada à edição textual e consideraram a interface do EDITEC simples e fácil de usar e que ela ajudava a entender os conceitos da

linguagem XTemplate. Os usuários, que não acharam a interface fácil de usar, sinalizaram nos comentários que ainda eram necessários bons conhecimentos técnicos em NCL e XTemplate para utilizar a ferramenta. Outros ainda comentaram que depois de entender a lógica da linguagem e de uso das opções de iteração, a interface se tornou mais simples e fácil de usar.

Alguns fizeram observações interessantes. Um usuário relatou que a separação da visão estrutural em duas subvisões (*Vocabulary View e Body View*) dificultava uma ambientação mais rápida. Entretanto, essas subvisões são apresentadas em separado para facilitar o entendimento da linguagem XTemplate e também porque podem existir *templates* só com elementos do vocabulário, por exemplo, quando se trata de um *template* só de apresentação. Neste caso, a *Body View* não é usada. Por isso, foi mantida a separação em duas subvisões no editor.

Outros usuários sugeriram um tutorial de uso, em texto e em vídeo. Um tutorial com diversos exemplos ajudará a entender melhor o uso da ferramenta, o que será providenciado futuramente.

Uma observação importante é que com o EDITEC os usuários também se familiarizam melhor com NCL, pois XTemplate apresenta vários elementos semelhantes a NCL. Assim, o editor ajuda a acelerar a tarefa de geração de aplicações declarativas NCL para TVDI.

Capítulo 6

Conclusões e Trabalhos Futuros

Esta dissertação apresentou o EDITEC, um editor gráfico de *template*s de composição baseado na linguagem XTemplate 3.0, e o seu modelo para representar estruturas de iteração que permite, de forma interativa, a criação de diversos *template*s com as mais variadas semânticas espaço-temporais. Um ponto chave na manipulação dos objetos gráficos do editor é a flexibilidade como a edição pode ser realizada. O editor está inserido em um conjunto de ferramentas que têm como objetivo facilitar a criação de documentos NCL.

O EDITEC foi desenvolvido visando a atender aos diversos requisitos desejados para um editor de *templates* de composição para a linguagem NCL. O editor está em fase de aprimoramento. À medida que ele for sendo usado, novas facilidades poderão ser identificadas e acrescentadas ao editor, até mesmo, novas opções para representar estruturas de iteração.

Podem existir situações em que a ferramenta de edição gráfica não seja capaz de fornecer todas as funcionalidades para que a lógica desejada pelo autor seja implementada, sendo necessário, nessas situações, o emprego da edição textual.

A próxima seção faz uma análise comparativa do EDITEC com as ferramentas apresentadas no Capítulo 2. Em seguida, são apresentadas as contribuições desta dissertação e os trabalhos futuros, visando a aperfeiçoar o ambiente de autoria EDITEC.

6.1 Análise Comparativa

Uma característica comum das ferramentas citadas no Capítulo 2 e a ferramenta EDITEC é que elas são desenvolvidas em linguagem Java, visando à portabilidade em diferentes plataformas.

Em relação ao LimSee2, o EDITEC também possui um conjunto de visões que facilitam um melhor entendimento do programa durante o processo de autoria. Sendo que o EDITEC não possui a visão temporal, e a visão textual é usada somente para a exibição do código, diferente do LimSee2, em que a visão textual pode ser editada. Por outro lado, o

LimSee2 não possui uma visão estrutural, sendo esta a principal visão de autoria do ambiente EDITEC. LimSee2 apresenta, além da visão textual, uma visão em XML do código do programa, que facilita a busca e identificação de informações. O EDITEC apresenta estruturas semelhantes, como a parte da visão estrutural que exibe os elementos do vocabulário, e a parte, também da visão estrutural, que exibe propriedades mais detalhadas dos elementos criados, semelhante à visão de atributos do LimSee2. LimSee2 não oferece o uso de *templates* para a autoria.

O modelo LimSee3, semelhante ao EDITEC, usa elementos do padrão XSLT. No EDITEC, esses elementos são utilizados para construir as estruturas de repetição. Ambos também utilizam expressões XPath para referenciar objetos. No caso do LimSee3, a definição temporal e espacial é feita diretamente no objeto, isso dificulta o reúso. LimSee3 permite também a manipulação do documento através de diversas visões e tem como principal foco o uso de *templates*.

O Composer, assim como o EDITEC, também possui múltiplas visões. O Composer, que tem como foco a construção direta de aplicações NCL, possui, além das visões estrutural, textual e de leiaute, uma visão temporal que permite a visualização no tempo dos elementos do documento. Na visão estrutural, o EDITEC permite a construção interativa dos elos associando papéis de conector a componentes. Um ponto importante do EDITEC em relação a essa visão é a exibição em miniatura da área de trabalho. As visões de leiaute de ambas as ferramentas são bastante similares, só que a visão do EDITEC possui mais funcionalidades, como a visão geral da área de trabalho e a possibilidade de definir, nesta visão, os descritores que serão usados no documento. Um ponto interessante do Composer é que ele utiliza o Ginga-NCL Emulator, uma ferramenta disponibilizada para testes de aplicações NCL. Apesar do Ginga-NCL Emulator apresentar problemas de fidelidade na apresentação de documentos NCL, a integração do Composer com essa ferramenta facilita a depuração da aplicação, aumentando o poder da ferramenta em desenvolver aplicações com um maior nível de interatividade. O Composer não oferece o uso de *templates* para autoria, mas o seu projeto de desenvolvimento pretende implementar essa opção futuramente.

A respeito do LuaComp, esta ferramenta, semelhante ao EDITEC, possui as visões estrutural, de leiaute e textual. A visão estrutural é bem mais simples que a fornecida pelo EDITEC. Um recurso interessante do LuaComp é a inserção de componentes pré-configurados, isso permite a inserção de códigos e ao mesmo tempo abstrai do autor, toda ou pelo menos parcialmente, a complexidade de se programar em NCL e Lua.

Similar ao Lamp que utiliza operadores para a sincronização de objetos de mídia, o EDITEC também utiliza operadores especiais, só que para a criação de generalizações, onde são geradas estruturas de repetição.

Em relação ao Contextual Ginga, o seu ambiente de autoria é bastante similar à visão de leiaute do EDITEC. Ambas as ferramentas possuem uma visão de propriedades com propósitos bastante similares e têm como foco o desenvolvimento de aplicações para TV digital. O EDITEC também salva o resultado da autoria de um *template* em vários arquivos. Ele salva o resultado em um arquivo de projeto, além desse arquivo também é criado um arquivo com o código textual XTemplate, um outro com a base de conectores onde se encontram os conectores usados no projeto, e um último arquivo referente às bases de apresentação (regiões e descritores).

O SCO Creator Tool é uma ferramenta que usa *templates* para criar conteúdo de ensino adaptativo via TV digital. Ele possui uma interface para criar regras de adaptação que possui algumas características similares a interface XPath do EDITEC. Nela o usuário cria expressões lógicas, de forma interativa, que são convertidas em código XML. Esse editor também permite a exibição de forma hierárquica dos objetos do *template* similar à exibição em árvore dos elementos do vocabulário na visão estrutural do EDITEC e das regiões na visão de leiaute. Além de possuir uma visão de propriedades, similar a visão de propriedades da visão estrutural do EDITEC.

A Tabela 10 complementa a Tabela 1, exibida no capítulo 2, apresentando as principais características do EDITEC.

Tabela 10: Análise comparativa das ferramentas de autoria, parte referente ao EDITEC.

Ferramenta	**Contexto Principal**	**Principais Características**	**Linguagem Base**	**Foco**	**Permite o Uso ou Criação de *Templates***
EDITEC	TV Digital	- Sincronismo de mídias; - Visões de leiaute, estrutural e textual; - Interface para construção de expressões XPath básicas; - Definição de estruturas de iteração de forma gráfica; - Visão em miniatura da área de trabalho.	XTemplate 3.0 e NCL	*Templates*	Sim

6.2 Contribuições

Pode-se destacar como principais contribuições desta dissertação:

- estudo de editores gráficos de documentos hipermídia e programas para TV digital;
- especificação e desenvolvimento da ferramenta EDITEC para criação e edição de *templates* de composição de acordo com a linguagem XTemplate 3.0;
- especificação e implementação de um modelo visual para criação de opções de iteração para criar relacionamentos e portas que se referem a componentes genéricos, que representam conjuntos de nós em um documento NCL que usa *templates*;
- proposta de interface para a criação de expressões XPath básicas no EDITEC;
- teste inicial de usabilidade do EDITEC considerando usuários com diferentes níveis de conhecimento em NCL e XTemplate.

6.3 Trabalhos Futuros

Como trabalho futuro, pretende-se implementar a visão temporal do editor. A visão temporal é a melhor maneira de mostrar quando os objetos de mídia são exibidos. No entanto, devido às restrições da representação no tempo, geralmente, a visão temporal é aplicável somente na autoria de documentos não interativos e não adaptativos, por restrições do paradigma utilizado para exibição da sincronização (Guimarães, 2007). São necessárias alternativas para contornar problemas de representação de eventos imprevisíveis, de forma a tornar a autoria através da visão temporal, mais atrativa para desenvolvedores de programas para TV digital.

Um documento XTemplate, como já foi dito, é dividido em quatro partes principais, cabeçalho, vocabulário, corpo e restrições. A parte de restrições (*constraints*) permite a definição de restrições adicionais sobre os componentes (nós e elos) utilizando a linguagem XPath. Essas restrições são usadas para garantir a consistência do documento final gerado, evitando erros no programa criado. Em algumas situações, restrições contendo expressões XPath são inseridas automaticamente pelo EDITEC no código do *template* para facilitar o seu uso. Restrições mais elaboradas, devem ser editadas diretamente no documento XTemplate

gerado. Um trabalho futuro é o desenvolvimento de uma interface gráfica para a construção dessas restrições pelos usuários de forma interativa. A criação desses tipos de restrições exige a construção de expressões XPath mais elaboradas, sendo necessário um estudo detalhado para a criação de um design de interface que seja de fácil uso.

Uma interface ou visão para a construção de conectores de forma gráfica e interativa também seria um trabalho futuro interessante. A criação dessa interface pode ser feita observando as definições e representações de relações definidas em (Guimarães et al., 2008), onde é apresentada uma abordagem visual de alto nível para especificar relações espaço-temporais. Esse trabalho apresenta um vocabulário visual, onde os símbolos visuais foram definidos para serem intuitivamente significantes para os autores. Esses símbolos (ícones) são usados para representar, por exemplo, condições simples e compostas, papéis de ação e condição de um conector.

Outro trabalho futuro é o desenvolvimento de um ambiente de testes, com um emulador ou simulador que permita que o autor execute e valide seu trabalho. O objetivo é permitir que o usuário teste o código do *template* gerado e também valide uma simulação do documento NCL final processado sem precisar indicar as mídias do documento.

Um objetivo futuro é estender o editor para ser, além de editor de *templates*, um editor de aplicativos NCL. O modo de criar elementos no EDITEC é semelhante à criação de elementos da linguagem NCL. A principal diferença se encontra no fato que o EDITEC permite que um componente represente um conjunto de mídias, mas se os componentes criados representarem apenas uma mídia — como nos *templates* "tctx1', "tctx2aN_1" e "tctxn' do Capítulo 4 — a autoria de aplicações XTemplate no EDITEC se assemelha à autoria de aplicações NCL. Neste caso, o modelo de opções de iteração e a interface XPath não são usados. O usuário constrói a sua aplicação de forma estruturada, criando cada contexto individualmente e depois define, em um documento intermediário, as mídias que participarão de cada contexto e cada contexto referencia um *template* específico. Assim, retirando as opções de iteração, a interface XPath e definindo que cada componente representa apenas um elemento, o EDITEC se torna semelhante a um editor de aplicações NCL, mas ainda é necessário o processamento dos *templates*, que nesse caso são semelhantes a contextos NCL finais. O objetivo é salvar o resultado em um arquivo NCL sem precisar de processamento de documento.

Outro trabalho futuro é o desenvolvimento de um editor gráfico de documentos NCL que usem *template*s. Seria um editor que oferecesse uma variedade de *templates* e ainda algumas opções extras para editar os *templates*. O usuário poderia definir em partes do

programa contextos que usassem *templates* específicos e em outras partes ele poderia criar e editar os relacionamentos usando elos diretamente.

Referências Bibliográficas

(ABNT, 2007) Norma ABNT 15606-2:2007. **Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações**, 2007.

(Alencar, 2007) Alencar M. S. **Televisão Digital**. Editora Érica. 1ª. Edição. 2007.

(Carvalho e Ferraz, 2010) Carvalho, A. P. B. A. and Ferraz. C. A. G. **Contextual Ginga: Uma Ferramenta de Autoria de Aplicações Interativas Sensíveis ao Contexto de TV digital para Ginga-NCL.** XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Gramado, Brasil, Maio de 2010.

(Celentano e Gaggi, 2003) Celentano, A. and Gaggi, O. **Template-Based Generation of Multimedia Presentations**. International Journal of Software Engineering and Knowledge Engineering. World Scientific Publishing Company. Vol. 13, No. 4, p.419-445, 2003.

(Contextual Ginga, 2010) **Contextual Ginga.** Disponível em: <http://www.cin.ufpe.br/~apba/index.html>. Acesso em setembro de 2010.

(Damasceno et al., 2010) Damasceno, J. R., Santos, J. A. F. and Muchaluat-Saade, D. C. **EDITEC: Editor Gráfico de Templates de Composição para Facilitar a Autoria de Programas para TV Digital Interativa**. Aceito em: XVI Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia2010. Belo Horizonte, Brasil, Outubro de 2010.

(Deitel, 2008) Deitel, H. M. and Deitel, P. J. **Java Como Programar 6ª Edição**. Ed. Pearson. São Paulo, 2008.

(Deltour e Roisin, 2006) Deltour, R. and Roisin, C. **The Limsee3 Multimedia Authoring Model**. In: Proceedings of the 2006 ACM Symposium on Document Engineering - DocEng'06, 173–175. Amsterdã, Holanda, Outubro de 2006.

(Deltour et al., 2005) Deltour, R., Layaida, N. and Weck, D. **LimSee2: A Cross-Platform SMIL Authoring Tool**. The European Research Consortium for Informatics and Mathematics – ERCIM News. n. 62, July 2005.

(DOM, 2000) **Document Object Model (DOM) Level 2 Core Specification Version 1.0**. W3C - World-Wide Web Consortium. W3C Recommendation, 13 de novembro de 2000. Disponível em: <http://www.w3.org/TR/DOM-Level-2-Core>. Acesso em setembro de 2010.

(Filho et al., 2007) Filho, G. S., Leite, L. and Batista, C. **Ginga-J: The Procedural Middleware for the Brazilian Digital TV System**. Journal of the Brazilian Computer Society, Vol. 12, No. 4, p. 47-56. March 2007.

(Gaggi e Celentano, 2006) Gaggi, O. and **Celentano**, A. **A Laboratory for Prototyping and Testing Multimedia Presentations**. International Journal of Software Engineering and Knowledge Engineering, Vol. 16, No 4, p. 615-642. August 2006.

(Ginga-NCL, 2010) **Ginga-NCL Emulator**. Disponível em: <www.ginga.org.br>. Acesso em agosto de 2010.

(Guimarães, 2007) Guimarães, R. L. **Composer: um ambiente de autoria de documentos NCL para TV digital interativa**. Dissertação de mestrado, Departamento de Ciência da Computação, PUCRio, Rio de Janeiro, Brasil, Junho 2007.

(Guimarães et al., 2008) Guimarães, R. L., Soares Neto, C. S R. and Soares, L. F. G. **A Visual Approach for Modeling Spatiotemporal Relations**. In: Proceedings of the 2008 ACM Symposium on Document Engineering - DocEng'08, p. 285-288. São Paulo, Brasil, Setembro de 2008.

(INRIA, 2010) **INRIA - Institut national de recherche en informatique et en automatique**. Disponível em: < http://www.inria.fr/index.en.html>. Acesso em setembro de 2010.

(ISDB-T, 1998) Terrestrial Integrated Services Digital Broadcasting **"Specification of Channel Coding, Framing Structure and Modulation"**. Setembro de 1998.

(ISO/IEC, 2005) ISO/IEC International Organization for Standardization. **Coding of audio-visual objects - Part 11: Scene description and application engine**. ISO/IEC 14496-11:2005.

(ITU, 2009) ITU - International Telecommunication Union. **Nested Context Language (NCL) and Ginga-NCL for IPTV services**. ITU-T Recommendation H.761. 2009. " Geneva, April 2009.

(LimSee2, 2010) **LimSee2**. Disponível em: <http://limsee2.gforge.inria.fr/> Acesso em setembro de 2010.

(LimSee3, 2010) **LimSee3**. Disponível em: < http://limsee3.gforge.inria.fr/public-site/> Acesso em setembro de 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, Vol. 40, No. 3, p. 409–451. 2008.

(Mendes, 2007) Mendes, L. L. SBTVD - **Uma visão sobre a TV Digital no Brasil**. TC Amazônia, Ano V, No. 12. Outubro de 2007.

(Morse et al., 2000) Morse, D. R., Armstrong, S., and Dey, A. K. **The What, Who, Where, When, and How of Context-Awareness.** In Proceedings of the CHI 2000 Workshop. Georgia Institute of Technology. Disponível em: <http://smartech.gatech.edu/handle/1853/3464>. Acesso em setembro de 2010.

(Muchaluat-Saade, 2003) Muchaluat-Saade, D.C. **Relações em Linguagens de Autoria Hipermídia: Aumentando Reuso e Expressividade**. Tese de Doutorado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, Março de 2003.

(Muchaluat-Saade e Soares, 2002) Muchaluat-Saade, D. C. and Soares, L. F. G. **XConnector & XTemplate: Improving the Expressiveness and Reuse in Web Authoring Languages**. The New Review of Hypermedia and Multimedia Journal, Vol. 8, p. 139 - 169. 2002.

(Neto e Ferraz, 2006) Neto, F. C. A. and Ferraz, C. A. G**. Uma Arquitetura para Suporte ao Desenvolvimento de Aplicações Sensíveis ao Contexto em Cenário de Convergência**. In: Anais do II Workshop de TV Digital (II WTVD), p. 39-49. Curitiba, Brasil, junho de 2006.

(Prota, 2010) Prota, T. M. **MOONDO: Um Framework para Desenvolvimento de Aplicações Declarativas no SBTVD.** Monografia, Centro de Informática, UFPE, Brasil, Janeiro de 2010.

(Santos, 2009) Santos J. A. F. **XTemplate 3.0: Adicionando Semântica a Composições e Fornecendo Reúso de Estruturas de Documentos Hipermídia**. Trabalho de Conclusão de Curso, Departamento de Engenharia de Telecomunicações, UFF/RJ, Niterói, Brasil, Dezembro de 2009.

(Santos e Muchaluat-Saade, 2009) Santos, J. A. F. and Muchaluat-Saade, D. C. **Linguagem XTemplate 3.0: Facilitando a Autoria de Programas NCL para TV Digital Interativa**. XV Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia2009. Fortaleza, Brasil, outubro de 2009.

(SBTVD, 2010) **SBTVD - Sistema Brasileiro de TV Digital.** Disponível em: <http://sbtvd.cpqd.com.br/> Acesso em agosto de 2010.

(Silva, 2005) Silva, H. V. O. **X-SMIL: Aumentando Reuso e Expressividade em Linguagens de Autoria Hipermídia**. Dissertação de mestrado, Departamento de Ciência da Computação, PUCRio, Rio de Janeiro, Brasil, Abril 2005.

(SMIL, 2005) **Synchronized Multimedia Integration Language – SMIL 2.1** Specification, W3C World-Wide Web Consortium. W3C Recommendation, 13 de dezembro de 2005. Disponível em: < http://www.w3.org/TR/SMIL2/>. Acesso em setembro de 2010.

(Soares e Barbosa, 2009) Soares, L. F. G. and Barbosa, S. D. J. **Programando em NCL: desenvolvimento de aplicações para Middleware Ginga, TV digital e Web**. Ed. Elsevier. Rio de Janeiro, 2009.

(Soares e Castro, 2008) Soares, L. F. G. and Castro P. H. **Middleware Ginga**. Técnica SBTVD-T - parte 5. 2008.

(Soares e Rodrigues, 2005) Soares, L. F. S e Rodrigues, R. F. **Nested Context Model 3.0 Part 1 – NCM Core**. Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, No. 18/05. ISSN 0103-9741. Rio de Janeiro. Maio de 2005.

(Soares et al., 2007) Soares, L. F. G., Rodrigues, R. F. and Moreno, M. F. **Ginga-NCL: The Declarative Environment of the Brazilian Digital TV System**. Journal of the Brazilian Computer Society, Vol. 12, No. 4. March, 2007.

(Souza Júnior, 2009) Souza Júnior, P. J. L. G. **LuaComp: Ferramenta de Autoria de Aplicações para TV Digital**. Dissertação de Mestrado, Departamento de Engenharia Elétrica, Universidade de Brasília, 2009.

(Sun, 2002) Sun Microsystems. **Java API for XML Processing (JAXP)**. 23 de Agosto de 2002. Disponível em: <http://java.sun.com/xml/jaxp/>. Acesso em setembro de 2010.

(Sun, 2008a) Sun Microsystems. **Graphical User Interfaces.** 14 de fevereiro de 2008. Disponível em: <http://java.sun.com/docs/books/tutorial/ui/index.html>. Acesso em agosto de 2010.

(Sun, 2008b) Sun Microsystems. **Java 2D API.** 14 de fevereiro de 2008. Disponível em: <http://java.sun.com/docs/books/tutorial/2d/index.html>. Acesso em agosto de 2010.

(Sun, 2010) Sun Microsystems. **Java 2 Platform Standard Edition 5.0 API Specification**. Disponível em <http://java.sun.com/j2se/1.5.0/docs/api/index.html>. Acesso em agosto de 2010.

(XML, 2008) **Extensible Markup Language (XML) 1.0 (Fifth Edition).** W3C - World-Wide Web Consortium. W3C Recommendation, 26 de novembro de 2008. Disponível em: http://www.w3.org/TR/REC-xml/. Acesso em setembro de 2010.

(XPath, 1999) **XML Path Language (XPath) Version 1.0**. W3C - World-Wide Web Consortium. W3C Recommendation, 16 de novembro de 1999. Disponível em: <http://www.w3.org/TR/xpath>. Acesso em setembro de 2010.

(XSLT, 1999) **XSL Transformations (XSLT) Version 1.0**. W3C - World-Wide Web Consortium. W3C Recommendation, 16 de novembro de 1999. Disponível em: <http://www.w3.org/TR/xslt>. Acesso em setembro de 2010.

Apêndice A

A.1 Exemplo “Parallel First”

A.1.1 *Template* “Parallel First”

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xtemplate  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xt="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL" description="Este
    template e  usado para..." id="tParallelFirst"
    xsi:schemaLocation="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL
    http://www.midiacom.uff.br/gtvd/XTemPlate30/profiles/XTPENCL.xsd">
3.    <head>
4.      <!--importacao da base de conectores-->
5.      <connectorBase>
6.              <importBase alias="connBase" documentURI="connBase.ncl"/>
7.      </connectorBase>
8.    </head>
9.
10.   <vocabulary>
11.     <!-- definicao de componentes-->
12.     <component xlabel="component"/>
13.
14.     <!-- definicao dos tipos de relacoes que podem existir-->
15.     <connector src="connBase#onEndNStopN" xlabel="onEndNStopN"/>
16.   </vocabulary>
17.
18.   <body>
19.     <!--definicao das portas do documento -->
20.     <variable name="i" select="1"/>
21.     <for-each select="child::*[@xlabel = 'component']">
22.             <port id="port" select="current()"/>
23.             <variable name="i" select="$i + 1"/>
24.     </for-each>
25.
26.     <!--definicao dos relacionamentos do documento -->
27.     <link id="linkOnEndMediaNAbortMediaN" xtype="onEndNStopN">
28.             <bind role="onEnd" select="child::*[@xlabel = 'component']"/>
29.             <bind role="stop" select="child::*[@xlabel = 'component']"/>
30.     </link>
31.   </body>
32.   <constraints/>
33. </xtemplate>
```

A.1.2 Exemplo de documento que referencia ao *template* "Parallel First"

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <ncl id="parFirst" xmlns="http://www.midiacom.uff.br/gtvd/XTemPlate/EXTProfile">
3.    <head>
4.
5.      <!--importacao do template-->
6.      <templateBase>
7.              <importBase documentURI="tParallelFirst.xml" alias="tParallelFirst" />
        </templateBase>

8.      <!--definicao das bases de regioes e descritores-->
9.      <regionBase>
10.             <region id="rgTV" width="640" height="480">
11.                     <region height="200" id="rg1" left="50" top="33" width="200" zIndex="0"/>
12.                     <region height="200" id="rg2" left="390" top="34" width="200" zIndex="1"/>
13.                     <region height="200" id="rg3" left="219" top="255" width="200" zIndex="1"/>
14.             </region>
15.     </regionBase>
16.
17.     <descriptorBase>
18.             <descriptor id="dVideo1" region="rg1"/>
19.             <descriptor id="dVideo2" region="rg2"/>
20.             <descriptor id="dVideo3" region="rg3"/>
21.     </descriptorBase>
22.
23.   </head>
24.
25.   <!--composicao referenciando o apelido do template atraves do atributo 'xtemplate' -->
26.   <body xtemplate="tParallelFirst">
27.
28.             <!-- um elemento especifica um xlabel igual a de um componente generico do template,
                assim, ele  recebe a mesma logica definida, no template, para esse componente generico-->
29.             <media id="video1" src="media/salto12s.mpeg" type="video/mpeg" descriptor="dVideo1"
                        xlabel="component"/>
30.
31.             <media id="video2" src="media/gol38s.mpeg" type="video/mpeg" descriptor="dVideo2"
                        xlabel="component"/>
32.
33.             <media id="video3" src="media/goleiro30s.mpeg" type="video/mpeg" descriptor="dVideo3"
                xlabel="component"/>
34.   </body>
35. </ncl>
```

A.1.3 Exemplo de documento NCL depois de processado que referenciava ao *template* "tParallelFirst".

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!-- Powered by XTemplate. -->
3.  <ncl id="parFirst-processed" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
4.    <head>
```

```
5.      <!--definicao das bases de regioes e descritores-->
6.      <regionBase>
7.              <region height="480" id="rgTV" width="640">
8.                      <region height="200" id="rg1" left="50" top="33" width="200" zIndex="0"/>
9.                      <region height="200" id="rg2" left="390" top="34" width="200" zIndex="1"/>
10.                     <region height="200" id="rg3" left="219" top="255" width="200" zIndex="1"/>
11.             </region>
12.     </regionBase>
13.
14.     <descriptorBase>
15.             <descriptor id="dVideo1" region="rg1"/>
16.             <descriptor id="dVideo2" region="rg2"/>
17.             <descriptor id="dVideo3" region="rg3"/>
        </descriptorBase>

18.     <!--importacao da base de conectores-->
19.     <connectorBase>
20.             <importBase alias="connBase" documentURI="connBase.ncl"/>
21.     </connectorBase>
22.
23.  </head>
24.
25.  <body>
26.     <media descriptor="dVideo1" id="video1" src="media/salto12s.mpeg" type="video/mpeg"/>
27.     <media descriptor="dVideo2" id="video2" src="media/gol38s.mpeg" type="video/mpeg"/>
28.     <media descriptor="dVideo3" id="video3" src="media/goleiro30s.mpeg" type="video/mpeg"/>
29.
30.     <!--definicao dos relacionamentos do documento -->
31.     <link id="link1" xconnector="connBase#onEndNStopN">
32.             <bind component="video1" role="onEnd"/>
33.             <bind component="video2" role="onEnd"/>
34.             <bind component="video3" role="onEnd"/>
35.             <bind component="video1" role="stop"/>
36.             <bind component="video2" role="stop"/>
37.             <bind component="video3" role="stop"/>
        </link>

38.     <!—portas de inicio do documento-->
39.     <port component="video1" id="port1"/>
40.     <port component="video2" id="port2"/>
41.     <port component="video3" id="port3"/>
42.  </body>
43. </ncl>
```

A.2 Exemplo “Apresentação de Slides”

A.2.1 *Template* “tApreSlides”

```
1.  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
2.  <xtemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xt="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL" description="Este
    template e  usado para..." id="tApreSlides"
    xsi:schemaLocation="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL
    http://www.midiacom.uff.br/gtvd/XTemPlate30/profiles/XTPENCL.xsd">
3.   <head>
4.      <!--importacao das bases de conectores e descritores -->
5.      <connectorBase>
6.              <importBase alias="connBase" documentURI="connBase.ncl"/>
7.      </connectorBase>
8.
9.      <descriptorBase>
10.             <importBase documentURI="presBase.ncl" alias="presBase"/>
11.     </descriptorBase>
12.  </head>
13.
14.  <vocabulary>
15.     <!-- definicao dos componentes e de seus pontos de interface-->
16.     <component descriptor="presBase#dBackground"  maxOccurs="1" minOccurs="1"
        xlabel="background"       xtype="image/gif"/>
17.
18.     <component maxOccurs="1" minOccurs="1" xlabel="ctx1" xtype="context">
19.             <port maxOccurs="1" minOccurs="1" xlabel="pCtx1"/>
20.             <port maxOccurs="1" minOccurs="1" xlabel="pProx"/>
21.     </component>
22.
23.     <component xlabel="ctxMeio" xtype="context">
24.             <port maxOccurs="1" minOccurs="1" xlabel="pA"/>
25.             <port maxOccurs="1" minOccurs="1" xlabel="pP"/>
26.             <port maxOccurs="1" minOccurs="1" xlabel="pCtx"/>
27.     </component>
28.
29.     <component maxOccurs="1" minOccurs="1" xlabel="ctxn" xtype="context">
30.             <port maxOccurs="1" minOccurs="1" xlabel="pAnt"/>
31.             <port maxOccurs="1" minOccurs="1" xlabel="pCtxn"/>
        </component>

32.     <!-- definicao dos tipos de relacoes que podem existir-->
33.     <connector src="connBase#onSelectionStart" xlabel="onSelectionStart"/>
34.     <connector src="connBase#onBeginStart" xlabel="onBeginStart"/>
35.  </vocabulary>
36.
37.  <body>
38.     <!--ponto de interface da composicao -->
        <port id="pInicio" select="child::*[@xlabel = 'background']"/>

39.     <!--definicao dos relacionamentos do template -->
40.     <link id="linkInicio" xtype="onBeginStart">
41.             <bind role="onBegin" select="child::*[@xlabel = 'background']"/>
42.             <bind role="start" select="child::*[@xlabel = 'ctx1']/child::*[@xlabel = 'pCtx1']"/>
43.     </link>
44.
```

```
45.     <link id="link1e2" xtype="onSelectionStart">
46.             <bind role="onSelection" select="child::*[@xlabel = 'ctx1']/child::*[@xlabel = 'pProx']"/>
47.             <bind role="start" select="child::*[@xlabel = 'ctxMeio'][position() = 1]/child::*[@xlabel =
                        'pCtx']"/>
48.     </link>
49.
50.     <link id="link2e1" xtype="onSelectionStart">
51.             <bind role="onSelection" select="child::*[@xlabel = 'ctxMeio'][position() =
    1]/child::*[@xlabel =           'pA']"/>
52.             <bind role="start" select="child::*[@xlabel = 'ctx1']/child::*[@xlabel = 'pCtx1']"/>
53.     </link>
54.
55.     <variable name="i" select="1"/>
56.     <for-each select="child::*[@xlabel = 'ctxMeio'][position ()!= last()]/child::*[@xlabel = 'pP']">
57.             <link id="link2an-1" xtype="onSelectionStart">
58.                     <bind role="onSelection" select="current()"/>
59.                     <bind role="start" select="child::*[@xlabel = 'ctxMeio'][position() =
                        $i+1]/child::*[@xlabel = 'pCtx']"/>
60.             </link>
61.             <variable name="i" select="$i + 1"/>
62.     </for-each>
63.
64.     <variable name="j" select="1"/>
65.     <for-each select="child::*[@xlabel = 'ctxMeio'][position()!=last()]/child::*[@xlabel = 'pA']">
66.             <link id="linkn-1a2" xtype="onSelectionStart">
67.                     <bind role="start" select="current()"/>
68.                     <bind role="onSelection" select="child::*[@xlabel='ctxMeio'][position()=$j+1]/
69.                     child::*[@xlabel='pCtx']"/>
70.             </link>
71.             <variable name="j" select="$j + 1"/>
72.     </for-each>
73.
74.     <link id="linkn-1en" xtype="onSelectionStart">
75.             <bind role="onSelection" select="child::*[@xlabel =
    'ctxMeio'][position()=last()]/child::*[@xlabel =           'pP']"/>
76.             <bind role="start" select="child::*[@xlabel = 'ctxn']/child::*[@xlabel = 'pCtxn']"/>
77.     </link>
78.
79.     <link id="linknen-1" xtype="onSelectionStart">
80.             <bind role="onSelection" select="child::*[@xlabel = 'ctxn']/child::*[@xlabel = 'pAnt']"/>
81.             <bind role="start" select="child::*[@xlabel = 'ctxMeio'][position() = last()]/child::*[@xlabel =
                'pCtx']"/>
82.     </link>
83.   </body>
84.   <constraints/>
85. </xtemplate>
```

A.2.2 *Template* "tCtx1"

```
1.  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
2.  <xtemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xt="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL" description="Este
    template e  usado para..." id="tCtx1"
    xsi:schemaLocation="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL
    http://www.midiacom.uff.br/gtvd/XTemPlate30/profiles/XTPENCL.xsd">
3.   <head>
4.      <!--importacao das bases de conectores e descritores -->
5.      <connectorBase>
6.              <importBase alias="connBase" documentURI="connBase.ncl"/>
7.      </connectorBase>
8.
9.      <descriptorBase>
10.             <importBase alias="presBase" documentURI="presBase.ncl"/>
11.     </descriptorBase>
12.  </head>
13.
14.  <vocabulary>
15.     <!-- definicao dos componentes-->
16.     <component descriptor="presBase#dTitle"  maxOccurs="1" minOccurs="1" xlabel="title"
        xtype="text/html"/>
17.
18.     <component descriptor="presBase#dText"  maxOccurs="1" minOccurs="1" xlabel="text"
        xtype="text/html"/>
19.
        <component descriptor="presBase#dButtonProxCorner"  maxOccurs="1" minOccurs="1"
        xlabel="buttonProx"       xtype="image/gif"/>

20.     <!-- definicao dos tipos de relacoes que podem existir-->
21.     <connector maxOccurs="1" minOccurs="1" src="connBase#onBeginStartN"
        xlabel="onBeginStartN"/>
22.
23.     <connector maxOccurs="1" minOccurs="1" src="connBase#onSelectionStopN"
        xlabel="onSelectionStoN"/>
24.  </vocabulary>
25.
26.  <body>
27.     <!--pontos de interface da composicao -->
28.     <port id="pCtx1" select="child::*[@xlabel = 'buttonProx']" xlabel="pCtx1"/>
29.
30.     <port id="pProx" select="child::*[@xlabel = 'buttonProx']" xlabel="pProx"/>

        <!--no' de midia especificado no proprio template -->
31.     <media id="buttonProx1" src="media/buttonProx.gif" type="image/gif" xlabel="buttonProx"/>

        <!--definicao dos relacionamentos do template -->
32.     <link id="linkInicio" xtype="onBeginStartN">
33.             <bind role="onBegin" select="child::*[@xlabel = 'buttonProx']"/>
34.             <bind role="start" select="child::*[@xlabel = 'text']"/>
35.             <bind role="start" select="child::*[@xlabel = 'title']"/>
36.     </link>
37.
38.     <link id="linKFim" xtype="onSelectionStoN">
39.             <bind role="onSelection" select="child::*[@xlabel = 'buttonProx']"/>
```

```
40.                 <bind role="stop" select="child::*[@xlabel = 'title']"/>
41.                 <bind role="stop" select="child::*[@xlabel = 'text']"/>
42.                 <bind role="stop" select="child::*[@xlabel = 'buttonProx']"/>
43.         </link>
44.   </body>
45.   <constraints/>
46.   </xtemplate>
```

A.2.4 *Template* "tCtxn"

```
1.   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2.   <xtemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xt="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL" description="Este
     template e  usado para..." id="tCtxn"
     xsi:schemaLocation="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL
     http://www.midiacom.uff.br/gtvd/XTemPlate30/profiles/XTPENCL.xsd">
3.    <head>
4.        <!--importacao das bases de conectores e descritores -->
5.        <connectorBase>
6.                 <importBase alias="connBase" documentURI="connBase.ncl"/>
7.        </connectorBase>
8.
9.        <descriptorBase>
10.                <importBase alias="presBase" documentURI="presBase.ncl"/>
11.       </descriptorBase>
12.   </head>
13.
14.   <vocabulary>
15.       <!-- definicao dos componentes-->
16.       <component descriptor="presBase#dTitle"  maxOccurs="1" minOccurs="1" xlabel="title"
          xtype="text/html"/>
17.
18.       <component descriptor="presBase#dText"  maxOccurs="1" minOccurs="1" xlabel="text"
          xtype="text/html"/>
19.
          <component descriptor="presBase#dButtonAntCorner"  maxOccurs="1" minOccurs="1"
          xlabel="buttonAnt"          xtype="image/gif"/>

20.       <!-- definicao dos tipos de relacoes que podem existir-->
21.       <connector maxOccurs="1" minOccurs="1" src="connBase#onBeginStartN"
          xlabel="onBeginStartN"/>
22.
23.       <connector maxOccurs="1" minOccurs="1" src="connBase#onSelectionStopN"
          xlabel="onSelectionStoN"/>
24.   </vocabulary>



25.   <body>
26.       <!--pontos de interface da composicao -->
27.       <port id="pAnt" select="child::*[@xlabel = 'buttonAnt']" xlabel="pAnt"/>
28.
```

```
        <port id="pCtxn" select="child::*[@xlabel = 'buttonAnt']" xlabel="pCtxn"/>

29.     <!--no' de midia especificado no proprio template -->
30.     <media id="buttonAntN" src="media/buttonAnt.gif" type="image/gif" xlabel="buttonAnt"/>

31.     <!--definicao dos relacionamentos do template -->
32.     <link id="linkInicio" xtype="onBeginStartN">
33.             <bind role="onBegin" select="child::*[@xlabel = 'buttonAnt']"/>
34.             <bind role="start" select="child::*[@xlabel = 'text']"/>
35.             <bind role="start" select="child::*[@xlabel = 'title']"/>
36.     </link>
37.
38.     <link id="linKFim" xtype="onSelectionStoN">
39.             <bind role="onSelection" select="child::*[@xlabel = 'buttonAnt']"/>
40.             <bind role="stop" select="child::*[@xlabel = 'title']"/>
41.             <bind role="stop" select="child::*[@xlabel = 'text']"/>
42.             <bind role="stop" select="child::*[@xlabel = 'buttonAnt']"/>
43.     </link>
44.  </body>
45.  <constraints/>
46. </xtemplate>
```

A.2.4 *Template* "tCtx2aN_1"

```
1.  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2.  <xtemplate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xt="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL" description="Este
    template e  usado para..." id="tCtx2aN_1"
    xsi:schemaLocation="http://www.midiacom.uff.br/gtvd/XTemPlate30/XTemPlateENCL
    http://www.midiacom.uff.br/gtvd/XTemPlate30/profiles/XTPENCL.xsd">
3.   <head>
4.      <!--importacao das bases de conectores e descritores -->
5.      <connectorBase>
6.              <importBase alias="connBase" documentURI="connBase.ncl"/>
7.      </connectorBase>
8.
9.      <descriptorBase>
10.             <importBase alias="presBase" documentURI="presBase.ncl"/>
11.     </descriptorBase>
12.  </head>
13.
14.  <vocabulary>
15.     <!-- definicao dos componentes-->
16.     <component descriptor="presBase#dTitle"  maxOccurs="1" minOccurs="1" xlabel="title"
        xtype="text/html"/>
17.
18.     <component descriptor="presBase#dText" maxOccurs="1" minOccurs="1" xlabel="text"
        xtype="text/html"/>
19.
20.     <component descriptor="presBase#dButtonAnt"  maxOccurs="1" minOccurs="1" xlabel="buttonAnt"
        xtype="image/gif"/>
21.
```

```
        <component descriptor="presBase#dButtonProx"  maxOccurs="1" minOccurs="1"
        xlabel="buttonProx"        xtype="image/gif"/>



22.     <!-- definicao dos tipos de relacoes que podem existir-->
23.     <connector maxOccurs="1" minOccurs="1" src="connBase#onBeginStartN"
        xlabel="onBeginStartN"/>
24.
25.     <connector maxOccurs="2" minOccurs="2" src="connBase#onSelectionStopN"
        xlabel="onSelectionStoN"/>
26.   </vocabulary>
27.
28.   <body>
29.     <!--pontos de interface da composicao -->
30.     <port id="pCtx" select="child::*[@xlabel = 'buttonAnt']" xlabel="pCtx"/>
31.     <port id="pA" select="child::*[@xlabel = 'buttonAnt']" xlabel="pA"/>
32.     <port id="pP" select="child::*[@xlabel = 'buttonProx']" xlabel="pP"/>
33.
34.     <!--definicao dos relacionamentos do template -->
35.     <link id="linkInicio" xtype="onBeginStartN">
36.             <bind role="onBegin" select="child::*[@xlabel = 'buttonAnt']"/>
37.             <bind role="start" select="child::*[@xlabel = 'text']"/>
38.             <bind role="start" select="child::*[@xlabel = 'title']"/>
39.             <bind role="start" select="child::*[@xlabel = 'buttonProx']"/>
40.     </link>
41.
42.     <link id="linKFim" xtype="onSelectionStoN">
43.             <bind role="onSelection" select="child::*[@xlabel = 'buttonAnt']"/>
44.             <bind role="stop" select="child::*[@xlabel = 'title']"/>
45.             <bind role="stop" select="child::*[@xlabel = 'text']"/>
46.             <bind role="stop" select="child::*[@xlabel = 'buttonAnt']"/>
47.             <bind role="stop" select="child::*[@xlabel = 'buttonProx']"/>
48.     </link>
49.
50.     <link id="linkFim2" xtype="onSelectionStoN">
51.             <bind role="onSelection" select="child::*[@xlabel = 'buttonProx']"/>
52.             <bind role="stop" select="child::*[@xlabel = 'title']"/>
53.             <bind role="stop" select="child::*[@xlabel = 'text']"/>
54.             <bind role="stop" select="child::*[@xlabel = 'buttonAnt']"/>
55.             <bind role="stop" select="child::*[@xlabel = 'buttonProx']"/>
56.     </link>
57.   </body>
58. <constraints/>
59. </xtemplate>
```

A.2.5 Exemplo de Documento NCL depois de processado que referenciava aos *templates* "tApreSlides", "tCtx1", "tCtx2aN_1" e "tCtxn".

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!-- Powered by XTemplate. -->
3.  <ncl id="apreSlides-processed" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
```

```
4.  <head>
5.      <!--importacao das bases de conectores e descritores -->
6.      <connectorBase>
7.              <importBase alias="connBase" documentURI="connBase.ncl"/>
8.      </connectorBase>
9.
10.     <descriptorBase>
11.             <importBase alias="presBase" documentURI="presBase.ncl"/>
12.     </descriptorBase>
13. </head>
14.
15. <body>
16.
17.     <!--midia do documento -->
        <media descriptor="presBase#dBackground" id="background" src="media/background.gif"
        type="image/gif"/>

18.     <!—nos de contexto do documento -->
19.     <context id="ctx1">
20.
21.             <!--midias de ctx1 -->
22.             <media descriptor="presBase#dTitle" id="title1" src="media/title1.html" type="text/html"/>
23.             <media descriptor="presBase#dText" id="text1" src="media/text1.html" type="text/html"/>
                <media descriptor="presBase#dButtonProxCorner" id="buttonProx1"
                                        src="media/buttonProx.gif" type="image/gif"/>

24.             <!--portas de ctx1 -->
25.             <port component="buttonProx1" id="pCtx11"/>
26.             <port component="buttonProx1" id="pProx2"/>
27.
28.                     <link id="link1" xconnector="connBase#onBeginStartN">
29.                             <bind component="buttonProx1" role="onBegin"/>
30.                             <bind component="text1" role="start"/>
31.                             <bind component="title1" role="start"/>
32.                     </link>
33.
34.                     <link id="link2" xconnector="connBase#onSelectionStopN">
35.                             <bind component="buttonProx1" role="onSelection"/>
36.                             <bind component="title1" role="stop"/>
37.                             <bind component="text1" role="stop"/>
38.                             <bind component="buttonProx1" role="stop"/>
39.                     </link>
40.     </context>
41.
42.     <context id="ctx2aN_1_1">
43.             <!--midias de ctx2aN_1_1-->
44.             <media descriptor="presBase#dTitle" id="title2" src="media/title2.html" type="text/html"/>
45.             <media descriptor="presBase#dText" id="text2" src="media/text2.html" type="text/html"/>
46.             <media descriptor="presBase#dButtonAnt" id="buttonAnt2" src="media/buttonAnt.gif"
                        type="image/gif"/>
47.             <media descriptor="presBase#dButtonProx" id="buttonProx2" src="media/buttonProx.gif"
                type="image/gif"/>
48.
```

```
49.             <!--portas de ctx2aN_1_1-->
50.             <port component="buttonAnt2" id="pCtx3"/>
51.             <port component="buttonAnt2" id="pA4"/>
52.             <port component="buttonProx2" id="pP5"/>
53.
54.             <link id="link3" xconnector="connBase#onBeginStartN">
55.                     <bind component="buttonAnt2" role="onBegin"/>
56.                     <bind component="text2" role="start"/>
57.                     <bind component="title2" role="start"/>
58.                     <bind component="buttonProx2" role="start"/>
59.             </link>
60.
61.             <link id="link4" xconnector="connBase#onSelectionStopN">
62.                     <bind component="buttonAnt2" role="onSelection"/>
63.                     <bind component="title2" role="stop"/>
64.                     <bind component="text2" role="stop"/>
65.                     <bind component="buttonAnt2" role="stop"/>
66.                     <bind component="buttonProx2" role="stop"/>
67.             </link>
68.
69.             <link id="link5" xconnector="connBase#onSelectionStopN">
70.                     <bind component="buttonProx2" role="onSelection"/>
71.                     <bind component="title2" role="stop"/>
72.                     <bind component="text2" role="stop"/>
73.                     <bind component="buttonAnt2" role="stop"/>
74.                     <bind component="buttonProx2" role="stop"/>
75.             </link>
76.     </context>
77.
78.     <context id="ctx2aN_1_2">
79.             <!--midias de ctx2aN_1_2-->
80.             <media descriptor="presBase#dTitle" id="title3" src="media/title3.html" type="text/html"/>
81.             <media descriptor="presBase#dText" id="text3" src="media/text3.html" type="text/html"/>
82.             <media descriptor="presBase#dButtonAnt" id="buttonAnt3" src="media/buttonAnt.gif"
                        type="image/gif"/>
83.             <media descriptor="presBase#dButtonProx" id="buttonProx3" src="media/buttonProx.gif"
                type="image/gif"/>
84.
85.             <!--portas de ctx2aN_1_2-->
86.             <port component="buttonAnt3" id="pCtx6"/>
87.             <port component="buttonAnt3" id="pA7"/>
88.             <port component="buttonProx3" id="pP8"/>
89.
90.             <link id="link6" xconnector="connBase#onBeginStartN">
91.                     <bind component="buttonAnt3" role="onBegin"/>
92.                     <bind component="text3" role="start"/>
93.                     <bind component="title3" role="start"/>
94.                     <bind component="buttonProx3" role="start"/>
95.             </link>
96.
97.             <link id="link7" xconnector="connBase#onSelectionStopN">
98.                     <bind component="buttonAnt3" role="onSelection"/>
99.                     <bind component="title3" role="stop"/>
```

```
100.                    <bind component="text3" role="stop"/>
101.                    <bind component="buttonAnt3" role="stop"/>
102.                    <bind component="buttonProx3" role="stop"/>
103.            </link>
104.
105.            <link id="link8" xconnector="connBase#onSelectionStopN">
106.                    <bind component="buttonProx3" role="onSelection"/>
107.                    <bind component="title3" role="stop"/>
108.                    <bind component="text3" role="stop"/>
109.                    <bind component="buttonAnt3" role="stop"/>
110.                    <bind component="buttonProx3" role="stop"/>
111.            </link>
112.    </context>
113.
114.    <context id="ctx2aN_1_3">
115.            <!--midias de ctx2aN_1_3-->
116.            <media descriptor="presBase#dTitle" id="title4" src="media/title4.html" type="text/html"/>
117.            <media descriptor="presBase#dText" id="text4" src="media/text4.html" type="text/html"/>
118.            <media descriptor="presBase#dButtonAnt" id="buttonAnt4" src="media/buttonAnt.gif"
                        type="image/gif"/>
                <media descriptor="presBase#dButtonProx" id="buttonProx4" src="media/buttonProx.gif"
                type="image/gif"/>

119.            <!--portas de ctx2aN_1_3-->
120.            <port component="buttonAnt4" id="pCtx9"/>
121.            <port component="buttonAnt4" id="pA10"/>
122.            <port component="buttonProx4" id="pP11"/>
123.
124.            <link id="link9" xconnector="connBase#onBeginStartN">
125.                    <bind component="buttonAnt4" role="onBegin"/>
126.                    <bind component="text4" role="start"/>
127.                    <bind component="title4" role="start"/>
128.                    <bind component="buttonProx4" role="start"/>
129.            </link>
130.
131.            <link id="link10" xconnector="connBase#onSelectionStopN">
132.                    <bind component="buttonAnt4" role="onSelection"/>
133.                    <bind component="title4" role="stop"/>
134.                    <bind component="text4" role="stop"/>
135.                    <bind component="buttonAnt4" role="stop"/>
136.                    <bind component="buttonProx4" role="stop"/>
137.            </link>
138.
139.            <link id="link11" xconnector="connBase#onSelectionStopN">
140.                    <bind component="buttonProx4" role="onSelection"/>
141.                    <bind component="title4" role="stop"/>
142.                    <bind component="text4" role="stop"/>
143.                    <bind component="buttonAnt4" role="stop"/>
144.                    <bind component="buttonProx4" role="stop"/>
145.            </link>
146.    </context>
147.
148.    <context id="ctxn">
```

```
149.            <!--midias de ctxn -->
150.            <media descriptor="presBase#dTitle" id="titleN" src="media/titleN.html" type="text/html"/>
151.            <media descriptor="presBase#dText" id="textN" src="media/textN.html" type="text/html"/>
152.            <media descriptor="presBase#dButtonAntCorner" id="buttonAntN"
                        src="media/buttonAnt.gif"  type="image/gif"/>
153.
154.            <port component="buttonAntN" id="pAnt12"/>
155.            <port component="buttonAntN" id="pCtxn13"/>
156.
157.            <link id="link12" xconnector="connBase#onBeginStartN">
158.                    <bind component="buttonAntN" role="onBegin"/>
159.                    <bind component="textN" role="start"/>
160.                    <bind component="titleN" role="start"/>
161.            </link>
162.
163.            <link id="link13" xconnector="connBase#onSelectionStopN">
164.                    <bind component="buttonAntN" role="onSelection"/>
165.                    <bind component="titleN" role="stop"/>
166.                    <bind component="textN" role="stop"/>
167.                    <bind component="buttonAntN" role="stop"/>
168.            </link>
169.
170.    </context>
171.
172.    <!—porta de inicio do documento-->
173.    <port component="background" id="port14"/>
174.
175.    <!--definicao dos relacionamentos do documento -->
176.    <link id="link14" xconnector="connBase#onBeginStart">
177.            <bind component="background" role="onBegin"/>
178.            <bind component="ctx1" interface="pCtx11" role="start"/>
179.    </link>
180.
181.    <link id="link15" xconnector="connBase#onSelectionStart">
182.            <bind component="ctx1" interface="pProx2" role="onSelection"/>
183.            <bind component="ctx2aN_1_1" interface="pCtx3" role="start"/>
184.    </link>
185.
186.    <link id="link16" xconnector="connBase#onSelectionStart">
187.            <bind component="ctx2aN_1_1" interface="pA4" role="onSelection"/>
188.            <bind component="ctx1" interface="pCtx11" role="start"/>
189.    </link>
190.
191.    <link id="link17" xconnector="connBase#onSelectionStart">
192.            <bind component="ctx2aN_1_3" interface="pP11" role="onSelection"/>
193.            <bind component="ctxn" interface="pCtxn13" role="start"/>
194.    </link>
195.
196.    <link id="link18" xconnector="connBase#onSelectionStart">
197.            <bind component="ctxn" interface="pAnt12" role="onSelection"/>
198.            <bind component="ctx2aN_1_3" interface="pCtx9" role="start"/>
199.    </link>
200.
```

```
201.     <link id="link19" xconnector="connBase#onSelectionStart">
202.             <bind component="ctx2aN_1_1" interface="pP5" role="onSelection"/>
203.             <bind component="ctx2aN_1_2" interface="pCtx6" role="start"/>
204.     </link>
205.
206.     <link id="link20" xconnector="connBase#onSelectionStart">
207.             <bind component="ctx2aN_1_2" interface="pP8" role="onSelection"/>
208.             <bind component="ctx2aN_1_3" interface="pCtx9" role="start"/>
209.     </link>
210.
211.     <link id="link21" xconnector="connBase#onSelectionStart">
212.             <bind component="ctx2aN_1_1" interface="pA4" role="start"/>
213.             <bind component="ctx2aN_1_2" interface="pCtx6" role="onSelection"/>
214.     </link>
215.
216.     <link id="link22" xconnector="connBase#onSelectionStart">
217.             <bind component="ctx2aN_1_2" interface="pA7" role="start"/>
218.             <bind component="ctx2aN_1_3" interface="pCtx9" role="onSelection"/>
219.     </link>
220.  </body>
221.</ncl>
```