Geração de Comandos de Edição na Autoria Textual de Documentos NCL

Rodrigo Silveira Alexandre
Rodrigo Costa Mesquita Santos
Thiago Alencar Gomes
Carlos de Salles Soares Neto

Laboratory of Advanced Web Systems - LAWS
Departamento de Informática – UFMA
Av. dos Portugueses, Campus do Bacanga São Luís/MA
CEP 65080-040 – Brasil

{rodrigo, rodrim.c, thiago}@laws.deinf.ufma.br, csalles@deinf.ufma.br

RESUMO

Este artigo apresenta métodos de autoria para o suporte à edição textual de documentos NCL (*Nested Context Language*) que podem ser aplicados para a geração de comandos de edição em tempo de exibição (ao vivo). NCL é a linguagem declarativa padrão do Sistema Brasileiro de TV Digital (SBTVD). Por meio da proposta deste artigo é possível oferecer mais facilmente a autores de aplicações interativas para TV digital o recurso de edição ao vivo, disponível no *middleware* Ginga-NCL, tornando suas aplicações passíveis de serem alteradas pela emissora durante a exibição. A proposta deste artigo foi integrada ao NCL Eclipse, ferramenta de autoria textual para NCL.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation – Hypertext/hypermedia, Multi/mixed media, Standards.

General Terms

Algorithms, Languages.

Keywords

Interactive Digital TV, NCL, SBTVD, NCL Eclipse, Live Editing.

1. INTRODUÇÃO

Aplicações interativas para TV digital podem ser representadas por documentos hipermídia, que definem o conteúdo de mídias e os relacionamentos entre os diversos objetos de mídia que compõem a aplicação. Habitualmente, a criação do documento e a apresentação do documento acontecem em fases e locais distintos: respectivamente em tempo de autoria e em tempo de exibição.

Em tempo de autoria, tipicamente o documento é especificado com o apoio de uma ferramenta especializada. Em um ambiente de TV digital, uma vez que esse documento está criado, sua transmissão é realizada por radiodifusão junto com o áudio e vídeo principais da emissora e suas diversas mídias inclusas. A apresentação do documento ocorre na casa do telespectador, em tempo de exibição. Um dispositivo ligado à TV, chamado conversor digital, é responsável por receber o sinal digital contendo o áudio e vídeo principais e os dados que formam a aplicação. Esses dados são interpretados numa camada de software intermediária, que é chamada de *middleware*, a qual é padronizada para garantir que uma mesma aplicação possa ser exibida em qualquer conversor, de qualquer fabricante. Percebe-se, portanto, que o documento que especifica a aplicação necessita estar disponível para o conversor digital momentos antes do início de sua exibição.

Quando a aplicação hipermídia é composta por objetos de mídia que o autor do documento possui conhecimento prévio de seus conteúdos, a especificação dos relacionamentos entre esses objetos pode ser feita sem que haja a necessidade posterior de alteração após o envio da aplicação pela emissora – salvo em ocasiões especiais onde por motivos externos é necessário intervir na apresentação dessa aplicação. Porém há situações onde não se sabe previamente o conteúdo das mídias, é o caso, por exemplo, de programações sendo transmitidas ao vivo. Por não ter ciência do conteúdo semântico das mídias, a especificação prévia de seus relacionamentos torna-se inviável. Considere, por exemplo, a transmissão de uma corrida de carros em que ao final é exibido o histórico do vencedor. Como não se sabe, *a priori*, quem vencerá a corrida e nem em quanto tempo ela irá terminar, não há como fazer uma aplicação estabelecendo relações espaço-temporais para a exibição de informações sobre o seu vencedor. Nesses casos tem-se a necessidade de alterar a aplicação durante a sua exibição.

O *middleware* Ginga-NCL [1] oferece suporte a comandos que permitam a alteração, em tempo de exibição, da estrutura lógica da aplicação, ou seja, edição ao vivo[1], tornando possível o uso de aplicações que possuem alterações semânticas durante sua exibição. O Ginga é o *middleware* especificado pelo Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) [1] e em seu ambiente para linguagem declarativa NCL define um conjunto de comandos de edição, os quais servem para gerar documentos NCL ou modificá-los em tempo de exibição (edição ao vivo).

[1] Neste trabalho, o termo edição ao vivo diz respeito aos documentos NCL que já foram enviados pela emissora e já se encontram armazenados na base privada, podendo ou não estarem sendo exibidos.

Este artigo apresenta técnicas capazes de identificar diferenças entre documentos hipermídia NCL. A identificação dessas diferenças é útil em aplicações que gerem comando de edição ao vivo ou, de forma geral, que fazem o controle de versões de documentos NCL. Esse artigo propõe um mecanismo capaz de abstrair a complexidade da sintaxe dos comandos de edição do Ginga-NCL, fazendo com que autores gerem esses comandos editando um documento normalmente, como se estivessem em tempo de autoria.

O artigo está organizado como segue. A Seção 2 discute os comandos de edição definidos para a linguagem NCL. A Seção 3 discute as técnicas de comparação de documentos NCL e a geração dos comandos de edição automaticamente como produto dessa comparação. A Seção 4 exemplifica o uso dessas técnicas na ferramenta de autoria NCL Eclipse. Por fim, a Seção 5 fornece as conclusões deste trabalho e trabalhos futuros.

2. COMANDOS DE EDIÇÃO AO VIVO EM NCL

Esta seção apresenta alguns conceitos básicos da linguagem NCL e discorre sobre os comandos de edição definidos em [1].

2.1 Conceitos Básicos da NCL

NCL é uma linguagem hipermídia declarativa baseada em XML. Possui como domínio a especificação de documentos que estabelecem relacionamentos de causalidade entre objetos de mídia, sendo que estes objetos podem ser textos, vídeos, imagens, áudio, códigos imperativos, etc. Além do sincronismo entre mídias, NCL permite elaboração de aplicações com adaptação de conteúdo e exibição de mídias em múltiplos dispositivos, tornado-a uma escolha coerente como linguagem para especificação de programas interativos para um sistema de TV Digital.

A NCL define uma estrita separação entre o conteúdo das mídias e a estruturação lógica do documento. Assim, é possível estabelecer relações espaço-temporais independentemente do conteúdo semântico dos objetos de mídia que fazem parte da aplicação.

Nós de mídia que possuem alguma relação semântica entre si podem ser agrupados com o objetivo de melhor estruturação do código da aplicação. Nós de composição são responsáveis por tal agrupamento, sendo que eles podem ser de dois tipos: contexto (<*context*>) ou alternativa (<*switch*>). Nós de contexto podem agrupar nós de mídia, nós de composição recursivamente e elos – discutidos a seguir. Posteriormente, nós de alternativa dão suporte à adaptação de conteúdo e especificam a execução de nós baseando-se na avaliação de regras definidas na base de regras (<*ruleBase*>). A definição de interfaces que dão acesso aos nós internos das composições é feita por meio do elemento <*port*>.

Os elos (<*link*>) são os elementos da linguagem que estabelecem o relacionamento entre as mídias. NCL distingue os conceitos de relação e relacionamento. As relações que podem ser utilizadas pelos elos são definidas na base dos conectores (<*connectorBase*>), sendo que cada relação define um conjunto finito de papéis que podem ser exercidos pelas âncoras dos nós. Já o relacionamento em si, que utiliza uma das relações definidas e faz o mapeamento entre as âncoras dos nós e os papéis definidos por esta relação, é especificado pelos elos.

2.2 Comandos de Edição

No Ginga-NCL, os documentos são agrupados em uma estrutura de dados chamada base privada [1]. O módulo Gerenciador de Bases Privadas do Ginga-NCL é responsável por receber os comandos de edição de documentos NCL e também por editar os documentos que estejam na base, podendo estes documentos estarem sendo exibidos ou não no momento de sua edição[7]. Os comandos de edição são enviados pela emissora de televisão através do mesmo canal de difusão de dados que envia as aplicações, porém esta não é a única forma de gerar esses comandos. O canal de interatividade[2] e eventos gerados por códigos imperativos (NCLua [8] ou NCLet) são duas outras formas de emissão de comandos de edição.

Em [1], são definidos três grupos de comandos de edição: I) comandos que manipulam as bases privadas (abrem, fecham, ativam e desativam as bases); II) comandos que manipulam documentos NCL (adicionam, removem e salvam documentos nas bases privadas e iniciam, pausam, param e retomam a apresentação dos documentos; e III) comandos que manipulam entidades NCL. Os comandos do grupo III) são o foco deste trabalho por permitir a manipulação direta dos elementos da linguagem e também por serem responsáveis por permitir a edição de documentos durante sua apresentação. Em [3] encontra-se uma discussão mais detalhada sobre os comandos de edição de documentos NCL Deste ponto em diante, sempre que nos referirmos a comandos de edição estaremos nos referindo especificamente aos comandos do grupo III).

Há comandos de edição para adicionar (*add*) e remover (*remove*) elementos para cada entidade da linguagem NCL. Como o Ginga associa uma base privada por canal de televisão, esses comandos necessitam explicitar sobre qual base e sobre qual documento desta base eles irão atuar. Os comandos *add* adicionam novas entidades ao documento. Se já existir um elemento com o mesmo identificador do elemento que está sendo adicionado, ele será atualizado. Os elementos inseridos por meio de comandos de edição devem sempre manter o documento consistente, portanto, todos os atributos obrigatórios deste elemento devem ser definidos. Na sintaxe dos comandos *remove*, é necessário especificar o identificador do elemento que se deseja remover, logo não é permitida a remoção de elementos que não tenham o atributo *id* definido.

Os comandos de edição geralmente obedecem certa ordenação lógica. Por exemplo, considere que sejam emitidos comandos para inserção de um novo nó de mídia e de um novo elo no documento que relacione a exibição dessa nova mídia com um determinado instante no tempo. O ideal é que o comando de inserção do nó de mídia seja emitido antes do comando de inserção do elo, pois caso a situação inversa acontecesse, o elo iria relacionar um nó que ainda não existe. Em casos como esse, em que o documento fica inconsistente devido a referências a elementos que não existem, elementos são criados com valores

[2] Canal de interatividade é o mecanismo de comunicação que fornece conexão entre o conversor digital e um servidor remoto [1].

padrões para seus atributos a fim de manter o documento consistente. Voltando para o exemplo, se o comando de inserção do elo chegasse antes do comando de inserção do nó de mídia, um novo nó de mídia seria criado e, quando finalmente o comando de inserir o nó chegasse, o elemento criado seria atualizado.

A Tabela 1 exibe parte dos comandos de edição definidos em [1]. Percebe-se que a assinatura dos comandos *add* define também que seja enviado junto com o comando um arquivo XML que contém a definição do elemento sendo inserido. Esse XML segue uma notação sintática idêntica ao XML Schema da linguagem NCL. Os comandos mostrados na Tabela 1 foram escolhidos didaticamente apenas para exemplificar a sintaxe dos comandos de edição.

Tabela 1. Alguns comandos de edição.

Comando	Descrição
addRegionBase (baseId, documentId, xmlRegionBase)	Adiciona uma base de regões ao documento *documentId* na base *baseId*.
removeRegionBase (baseId, documentId, regionBaseId)	Remove a base de regões *regionBaseId* do documento com identificador *documentId* na base *baseId*.
AddRegion (baseId, documentId, regionBaseId, regionId, xmlRegion)	Adiciona uma região à base de regiões *regionBaseId* do documento com identificador *documentId* na base *baseId*.
removeRegion (baseId, documentId, regionId)	Remove a região *regionId* da base de regiões *regionBaseId* do documento com identificador *documentId* na base *baseId*.
addInterface (baseId, documentId, nodeId, xmlInterface)	Adiciona uma interface ao nó *nodeId* do documento com identificador *documentId* na base *baseId*.
removeInterface (baseId, documentId, nodeId, interfaceId)	Remove a interface *interfaceId* da base *regionBaseId* do documento *documentId* na base *baseId*.

3. MECANISMO DE GERAÇÃO DE COMANDOS DE EDIÇÃO AO VIVO

A Seção 3.1 descreve um novo algoritmo proposto neste trabalho para a comparação de documentos NCL, o qual é passível de ser aplicado para identificar diferenças entre dois documentos diferentes ou entre versões de um mesmo documento. A Seção 3.2 apresenta como a lista de diferenças encontradas é usada para a geração de comandos de edição ao vivo. O artigo não apresenta o algoritmo de modo formal devido a restrições de espaço.

3.1 Controle de Versão entre Documentos NCL

Documentos NCL podem ser representados como árvores, sendo esta representação crucial para a comparação. A navegação em árvore torna viável a iteração entre os elementos do documento e mantém o controle sobre o nível em que está o elemento de trabalho nos documentos sendo comparados. Uma alternativa para a representação em árvore é o uso da API DOM [9]. O primeiro passo para realizar a comparação é transformar os dois documentos em árvores.

Para melhor entendimento do algoritmo aplicado entre dois documentos, os parágrafos seguintes tratam o primeiro deles por "documento original" e o segundo é denominado "documento modificado". Essa nomenclatura é útil quando se trata do mesmo documento em duas versões, sendo a segunda delas o documento final ou atual.

A comparação se dá em duas passagens. Na primeira, o documento principal será o modificado e seus elementos serão os elementos de trabalho. Nessa passagem é feita a identificação dos elementos que foram adicionados ou alterados, pois, para cada elemento encontrado no documento modificado, é verificada sua presença ou alteração no documento original. Já na segunda passagem, tomando o documento original como principal, é possível identificar os elementos que foram removidos.

Os elementos são percorridos recursivamente, atentando para o fato de que o foco da comparação dever estar, obrigatoriamente, no mesmo nível nas duas árvores. Durante esse percurso, o elemento de trabalho, que pertence ao documento modificado é comparado com todos os elementos do documento original que estão no mesmo nível, à procura de alterações. Isso acontece porque alterações de posição entre elementos do mesmo nível não alteram a estrutura semântica do documento.

A comparação só é feita entre elementos do mesmo tipo. Comparar um elemento <*region*> com um elemento <*rule*> não seria possível, pois esses dois elementos têm características distintas e atributos completamente diferentes. O resultado de uma comparação desse gênero geraria uma enorme quantidade de erros.

Uma vez encontrada uma coincidência na natureza dos elementos, a comparação é realizada de acordo com o elemento. Alguns deles podem ser diferenciados apenas comparando seus atributos, como é o caso do elemento <*region*>. Outros, porém, exigem mais atenção, como é o caso de elementos <*bind*>, que exigem primeiro a comparação pelos atributos *role* e *component*. Após isso, são comparados os outros atributos e, depois de passar por esses dois testes, ainda é necessária a comparação dos elementos filhos, do tipo <*bindParam*>, para se poder afirmar se os elementos são iguais. A Tabela 2 mostra como alguns elementos da NCL podem ser comparados.

Quando o elemento de trabalho não é encontrado no documento original, essa alteração é registrada como inserção de um novo elemento e, logo após isso, a iteração segue para o novo elemento de trabalho. Note que esse novo elemento de trabalho não poderá ser filho do elemento de trabalho atual, caso ele exista, já que este nível não existe no documento original.

Quando o elemento de trabalho é encontrado com modificações nos seus atributos, essa modificação é registrada como atualização de um elemento existente e a escolha do novo elemento de trabalho se dá da mesma forma descrita anteriormente.

Após essa primeira passagem, é realizada uma segunda, só que dessa vez os elementos de trabalho serão os elementos do documento original, que serão comparados aos elementos do documento modificado. Nessa segunda passagem, a comparação é bem mais simples, pois só o que é analisado é a presença ou ausência de cada elemento. Em outros termos, se o elemento de trabalho não for encontrado no documento modificado, significa que o mesmo foi removido, e registra-se essa alteração. É importante notar que alterações em elementos existentes já foram observadas na primeira passagem.

Tabela 2. Alguns elementos e seus modos de comparação.

Elemento	Modo de comparação
regionBase	1. Atributo id 2. Atributo device 3. Elementos filhos (apenas elementos "importBase")
region, media, context, area, port, connectorParam,	Comparação direta dos atributos.
causalConnector	1. Atributo id 2. Elementos filhos (elementos connectorParam, simpleCondition, compoundCondition, simpleAction, compoundAction)
link	1. Atributo xconnector 2. Atributo id 3. Elementos filhos (elementos "linkParam" e "bind")
compoundCondition	1. Comparação direta dos atributos 2. Elementos filhos (elementos simpleCondition, compoundCondition), assessmentStatement, compoundStatement)

Ao final das duas passagens, obtém-se a lista de modificações realizadas entre duas versões de um documento NCL ou mesmo entre dois documentos diferentes. Essa lista de alterações será vazia quando os documentos forem semanticamente iguais.

De posse dessa lista de alterações, o autor poderá analisar facilmente se o arquivo sofreu alguma modificação e quais foram essas modificações. O autor pode, por exemplo, voltar a editar o documento e desfazer algumas delas, no caso de sua aplicação não funcionar como o planejado. O autor pode, ainda, manter um histórico das alterações que o documento NCL sofreu.

3.2 Geração dos Comandos de Edição ao Vivo

Utilizando o mecanismo de comparação descrito na Seção 3.1 para identificar as diferenças entre dois documentos NCL, pode-se gerar os comandos de edição ao vivo adequados para transformar o documento original no documento modificado.

Para a maioria dos elementos NCL essa tarefa é trivial: se for identificado que um elemento foi adicionado ou teve seus atributos atualizados, então é gerado um comando de inserção (*add*) ou, se for identificado que um elemento foi removido, então é gerado um comando de remoção (*remove*). Para elementos que não possuem comandos de edição que os altere diretamente, deve ser feito um processamento a mais, pois nesse caso é preciso utilizar comandos que alterem seus elementos pais.

No momento em que são gerados os comandos de inserção, deve ser gerada uma descrição do elemento que está sendo adicionado ou atualizado. Isso é feito gerando-se um arquivo XML auxiliar para cada elemento adicionado ou atualizado. Esse XML conterá uma cópia exata do elemento modificado, cópia essa extraída do documento modificado.

A execução do algoritmo de comparação será ilustrada tomando dois trechos de código como exemplo: o primeiro é o documento original, na Figura 1(a); o segundo é o documento modificado, na Figura 1(b).

```
<regionBase device="systemScreen(1)">
   <region id="rgTV" width="1080" height="768">
      <region id="rgVideo1" left="448" top="100"
              width="1024" height="768" zIndex="1"/>
      <region id="rgOpcao1" left="200" top="100"
              width="200" height="50" zIndex="2"/>
   </region>
</regionBase>
```

Figura 1 (a). Trecho de código original.

```
<regionBase device="systemScreen(1)">
   <region id="rgTV" width="1080" height="768">
      <region id="rgVideo1" left="448" top="100"
              width="1024" height="768" zIndex="1"/>
      <region id="rgOpcao1" left="200" top="100"
              width="250" height="50" zIndex="2"/>
      <region id="rgOpcao3" left="200" top="200"
              width="200" height="50" zIndex="2"/>
   </region>
</regionBase>
```

Figura 1 (b). Trecho de código modificado.

Percebe-se que o elemento <*region*> com o *id*="*rgOpção1*" teve o valor do seu atributo *width* alterado de 200 para 250 e ainda que um novo elemento, identificado pelo atributo *id*="*rgOpção3*", foi incluído. O resultado dessa comparação será os comandos *addRegion* (*baseId*, *documentId*, "*IdDaRegionBase*", "*rgOpção1*", "*Region1.xml*") e *addRegion* (*baseId*, *documentId*, "*IdDaRegionBase*", "*rgTV*",

"*Region2.xml*"). Como um elemento foi modificado e outro foi adicionado, ambos exigiram o uso de comandos *add* com a referência para arquivos XML: *Region1.xml* e *Region2.xml*. Como explicado, cada um desses arquivos XML, que são gerados no momento em que os próprios comandos estão sendo gerados, contém cópias exatas dos elementos alterados no documento modificado.

Já nos casos em que os elementos modificados são do tipo que não possuem comandos de edição que os altere diretamente, o funcionamento é exemplificado a seguir, nas Figuras 2(a) e 2(b).

```
<connectorBase>
   <causalConnector id="onSelection1SetNStopNStartN">
      <connectorParam name="var"/>
      <simpleCondition role="onSelection"/>
      <compoundAction operator="seq">
         <simpleAction role="set" value="$var"
                  max="unbounded" qualifier="par"/>
         <simpleAction role="stop" max="unbounded"
                                    qualifier="par"/>
         <simpleAction role="start" max="unbounded"
                                    qualifier="par"/>
      </compoundAction>
   </causalConnector>
</connectorBase>
```

Figura 2(a). connectorBase original.

```
<connectorBase>
   <causalConnector id="onSelection1SetNStopNStartN">
      <connectorParam name="var"/>
      <simpleCondition role="onSelection" key="RED"/>
      <compoundAction operator="seq">
         <simpleAction role="set" value="$var"
                  max="unbounded" qualifier="par"/>
         <simpleAction role="stop" max="unbounded"
                                    qualifier="par"/>
         <simpleAction role="start" max="unbounded"
                                    qualifier="par"/>
      </compoundAction>
   </causalConnector>
</connectorBase>
```

Figura 2(b). connectorBase modificado.

Neste caso, apenas o fato de acrescentar o atributo *key* no elemento *simpleCondition* força a uma alteração no elemento *causalConnector* como um todo, gerando o comando *addConnector (baseId, documentId, "connector.xml").*

O resultado desse algoritmo será uma lista de comandos de edição ao vivo que torne os documentos iguais, além dos arquivos XML que descrevem os elementos adicionados ou atualizados, se for necessário.

Os comandos de edição ao vivo do tipo *remove* exigem o conhecimento prévio do atributo *id* do elemento que está sendo removido. Caso o autor tente remover um elemento sem *id*, então não é possível gerar o comando apropriado. Uma ferramenta de autoria deveria, por exemplo, alertar isso ao autor e abortar a geração dos comandos de edição.

4. INTEGRAÇÃO COM O NCL ECLIPSE

A única ferramenta disponível para a geração de comandos de edição ao vivo de documentos NCL é o Composer [5]. O Composer usa abstrações sobre os documentos criando várias visões. Cada visão mostra um aspecto diferente da aplicação e todas estão sincronizadas com um modelo central para manter consistência. O Composer possui as seguintes visões: temporal, que mostra o comportamento da aplicação NCL por uma linha do tempo; a estrutural, que evidencia a forma como ocorrem as interações entre os elementos da linguagem; a textual, que permite visualizar e editar o código fonte diretamente; e a de leiaute, que permite a edição das regiões. Todas essas visões são notificadas com qualquer mudança feita, quando o modo de edição ao vivo está selecionado, gerando os comandos de edição em tempo de construção da aplicação, que podem ser exportados para que possam ser transmitidos pela emissora como eventos de fluxo.

O NCL Eclipse [2] é uma das ferramentas de autoria mais utilizadas pela comunidade de programadores NCL. É um *plug-in* textual para a IDE (*Integrated Development Environment*) Eclipse [4] e implementa diversas funcionalidades que auxiliam o desenvolvimento de documentos NCL. Alguns exemplos são a sugestão de código contextual, coloração sintática dos elementos, navegação no documento por meio de hiperelos.

Para validar a proposta deste artigo, foi realizada uma implementação do algoritmo descrito na Seção 3.1 junto ao NCL Eclipse. Mais precisamente, o componente *NCLInTime*, que implementa o algoritmo, foi acrescentado à arquitetura do *plug-in*, se comunicando com o núcleo. Esse componente é iniciado sempre que um botão, inserido na barra de ferramentas, é pressionado. A partir daí, o *NCLInTime* salva o estado atual do documento e passa a registrar as alterações no mesmo até que o botão seja pressionado uma segunda vez. Quando isso ocorre, inicia-se a comparação do documento que foi salvo quando o botão foi pressionado pela primeira vez com o documento modificado. A Figura 3 exibe a barra de ferramentas do NCL Eclipse com o botão que ativa o *NCLInTime* destacado.

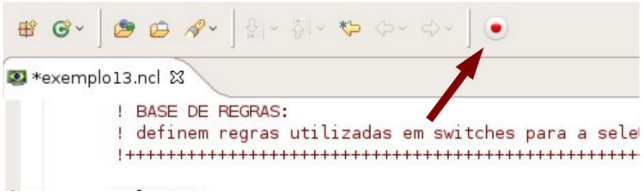

Figura 3. Botão que aciona o componente *NCLInTime*.

Uma vez tendo sido identificadas as diferenças entre os documentos, os comandos de edição são gerados. Para a exibição desses comandos, foi criada uma nova visão no *plug-in*: a *Console*. A Figura 4 exibe o NCL Eclipse em funcionamento após ter sido ativado o componente *NCLInTime*. Observa-se na parte inferior da figura a visão *Console*, exibindo os comandos de edição gerados.

Sempre que o mecanismo de geração de comandos de edição é acionado, uma pasta é criada, dentro do diretório da aplicação, contendo os arquivos XML que descrevem os elementos modificados. Esse pacote com os comandos e arquivos XML é o suficiente para um gerador de carrossel de dados [6] transmitir a aplicação por difusão, junto com os comandos para atualizá-la ao vivo.

Uma das principais diferenças entre o mecanismo de geração de comandos de edição apresentado neste trabalho e implementado junto ao NCL Eclipse com o mecanismo implementado no Composer é que neste último, quando a opção de geração de comandos de edição está ativada não é possível alterar o documento NCL utilizando a visão textual, apenas as outras visões. Isso pode ser considerado como um ponto negativo por impedir os programadores de utilizar essa funcionalidade editando diretamente o código fonte. Tal inconveniente não é observado no NCL Eclipse, uma vez que o objetivo dessa ferramenta é a edição textual.

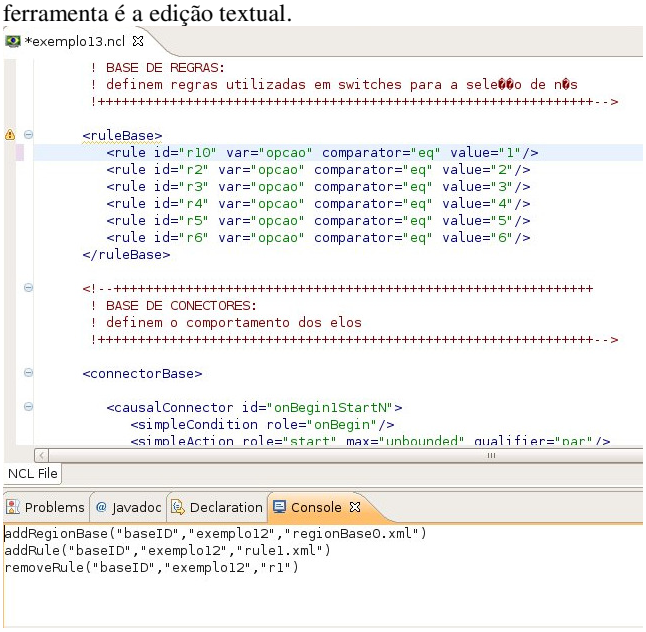

Figura 4. NCL Eclipse com geração de comandos de edição.

5. CONCLUSÃO E TRABALHOS FUTUROS

Este artigo apresentou uma abordagem para comparação de documentos NCL, gerando automaticamente e em tempo de exibição, comandos de edição ao vivo. Do ponto de vista do autor, isso é obtido como se ele estivesse editando um documento em tempo de autoria, o que é interessante para tornar transparente a forma de geração desses comandos e abstrair a complexidade da sintaxe dos mesmos. Usando a API DOM, são geradas árvores das versões do documento, que são comparadas de acordo com o algoritmo proposto neste trabalho.

A presente abordagem foi integrada com a ferramenta NCL Eclipse, criando um arquivo com os comandos de edição que pode ser utilizado pelas emissoras, refletindo as modificações realizadas na autoria sobre os programas em execução.

Um importante trabalho futuro é oferecer ao usuário do NCL Eclipse uma geração de *logs*, durante a criação de documentos novos, que possuem os comandos necessários para criação daquela aplicação usando comandos de edição.

6. REFERÊNCIAS

[1] ABNT - Associação Brasileira de Normas Técnicas (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 XM L para codificação de aplicações” .

[2] Azevedo, R. G. A., Teixeira, M. M., e Soares Neto, C. S. 2009. NCL Eclipse: Ambiente Integrado para o Desenvolvimento de Aplicações para TV Digital Interativa em Nested Context Language. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), Recife, 2009.

[3] Costa, R. M. R., Moreno, M. F., Rodrigues, R. F., e Soares, L. F. G. Live editing of hypermedia documents, Proceedings of the 2006 ACM symposium on Document engineering, October 10-13, 2006, Amsterdam, The Netherlands

[4] Eclipse SDK. Disponível em: http://www.eclipse.org.

[5] Guimarães, R. L., Costa, R. M. R., ande Soares, L. F. G. 2007. Composer: Ambiente de Autoria de Aplicações Declarativas para TV Digital. XIII Simpósio Brasileiro de Sistemas Multimídia e Web WebMedia2007. Gramado, Brasil - Outubro de 2007.

[6] Pessoa, B. J. S.; Souza Filho, G. L.; Cabral, L. A. F. Metaheurísticas aplicadas à geraçao de carrossel no sistema brasileiro de TV Digital. In: Brazilian Symposium on Multimedia and the Web. Proceedings of the 14th Brazilian Symposium on Multimedia and the Web. Vila Velha, Brazil.

[7] Moreno, M. F., Costa, R. M. R., Rodrigues, R. F., e Soares, L. F. G. 2005. Edição de Documentos Hipermídia em Tempo de Exibição. XI Simpósio Brasileiro de Sistemas Multimídia e WEB – WebMedia, Poços de Caldas, Brasil, Dezembro de 2005.

[8] Sant'Anna, F., Cerqueira, R., and Soares, L. F. 2008. NCLua: objetos imperativos lua na linguagem declarativa NCL. In *Proceedings of the 14th Brazilian Symposium on Multimedia and the Web* (Vila Velha, Brazil, October 26 - 29, 2008). WebMedia '08.

[9] W3C. *Document Object Model (DOM) Level 3 Load and Save Specification*. 2004. URL: http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407/.