UNIVERSIDADE FEDERAL FLUMINENSE

ESDRAS CALEB OLIVEIRA SILVA

JNS E NCL4WEB: AUXILIANDO O DESENVOLVIMENTO E DIVULGAÇÃO DE DOCUMENTOS NCL

NITERÓI

2013

UNIVERSIDADE FEDERAL FLUMINENSE

ESDRAS CALEB OLIVEIRA SILVA

JNS E NCL4WEB: AUXILIANDO O DESENVOLVIMENTO E DIVULGAÇÃO DE DOCUMENTOS NCL

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

Orientador:
DÉBORA CHRISTINA MUCHALUAT SAADE

NITERÓI

2013

ESDRAS CALEB OLIVEIRA SILVA

JNS E NCL4WEB: AUXILIANDO O DESENVOLVIMENTO E DIVULGAÇÃO DE DOCUMENTOS NCL

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

Aprovada em ?? de 2013.

BANCA EXAMINADORA

Prof. Débora Christina Muchaluat Saade - Orientador, UFF

Prof. ??, ???

Prof. ??, ???

Prof. ??, ???

Prof. ??, ???

Niterói

2013

Dedicatória: Dedico este trabalho a todos aqueles que buscam o desenvolvimento da humanidade.

Agradecimentos

Agradeço a Deus, pois tudo fez e tudo sustenta. Aos meus pais, pois me criaram neste mundo. Aos meus professores e mestres, pois não só me ensinaram mas acreditaram em mim. Ao Instituto de Computação da UFF e a todos os meus amigos que me ajudaram nestes anos na UFF.

Resumo

Este trabalho tem como objetivo facilitar o desenvolvimento de documentos NCL e promover a divulgação de aplicações NCL. Atualmente, um desenvolvedor que usa a linguagem NCL tem um grande mercado consumidor, o que inclui quase toda a América do Sul. Entretanto, o processo de divulgação de uma aplicação não é possível sem um sistema de transmissão de TV digital, algo que demanda grandes recursos financeiros. Este trabalho propõe a ferramenta NCL4WEB, capaz de traduzir um documento NCL em HTML5, permitindo assim, a divulgação de aplicações NCL na web e possibilitando ao desenvolvedor a facilidade para publicar internacionalmente o seu trabalho. NCL4WEB é apresentada mostrando as decisões de projeto em usar uma transformação híbrida com folha de estilos XSLT e uma biblioteca JavaScript. A ferramenta foi testada em diversos navegadores web e depende da compatibilidade dos mesmos com alguns formatos de video, como MPEG-4. Outra proposta desta dissertação é a linguagem JNS - JSON NCL Script, que é uma linguagem de script para autoria NCL baseada no formato JSON, bastante utilizado em aplicações web. A linguagem JNS foi projetada para autores com alguma experiência em programação, pois utiliza alguns conceitos de linguagens procedurais para autoria NCL. Uma das vantagens do uso de JNS é diminuir o tamanho do código do documento multimídia quando comparado a um documento NCL. Outra vantagem é flexibilizar a especificação de uma aplicação, não obrigando o autor a declarar todos os elementos de linguagem necessários em NCL. O estudo realizado aponta trabalhos relacionados que auxiliaram no projeto da linguagem JNS ou que utilizam uma abordagem similar, comparando-os com a proposta desta dissertação. Ainda é analisado o quanto é possível economizar em linhas e estruturas pelo uso do JNS quando comparado ao uso de NCL. Também foram feitos testes de usabilidade com a linguagem JNS. Com o propósito de permitir o desenvolvimento da linguagem JNS, foram criados um compilador de JNS para NCL e um complemento para o editor de textos gedit. Espera-se com este trabalho auxiliar o desenvolvimento de aplicações NCL e sua divulgação.

Palavras-chave: Autoria Multimídia, NCL, JNS, NCL4WEB, HTML5, XSLT.

Abstract

This work aims to ease the development of NCL documents and promote propagation of NCL applications. Currently, a developer who uses the language NCL has a large consumer market, which includes almost all of South America. However, the disclosure process of the application is not possible without a Digital Television transmission system, something that requires large financial resources. This work proposes the NCL4WEB tool, capable to translate an NCL document to an HTML5 document, enabling the release of NCL applications on the web and allowing the developer to publish internationally his work. NCL4WEB is present showing the project decisions in using the hybrid transformation with a XSLT stylesheet and a JavaScript Library. The tool was tested in different web browsers and depends on their compatibility with some video formats, such as MPEG-4. Another purpose of this dissertation is the language JNS - JSON NCL script, which is a scripting language for authoring NCL based in JSON format, widely used in web applications. The language was designed to JNS authors with some programming experience, because it uses some concepts of procedural languages for authoring NCL. One of the advantages of using JNS is to reduce the code size of the multimedia document compared to an NCL document. Another advantage is flexibility in the specification of an application, not forcing the author to declare all language elements needed in NCL. The study points out related work that assisted in the design of language JNS or that use a similar approach, comparing them with the purpose of this dissertation. Also is analyzed how you can save on lines and structures by using the JNS when compared to using NCL. Were also made usability testing with language JNS. In order to allow the development of language JNS, were created a compiler for JNS NCL and an addition to the text editor gedit. It is hoped that this work help develop NCL applications and their dissemination. Also expected to promote the emergence of new ideas for jobs that use JSON as the basis for a language like XSLT and translation tool for web, since their integration in this environment is simple.

Keywords: Multimedia Authoring, NCL, JNS, NCL4WEB, HTML5, XSLT.

Lista de Figuras

3.1 Estruturas do JSON 14

3.2 Valores do JSON 15

3.3 Comparação JNS e NCL 26

3.4 Resultado do teste de parâmetros de cognição 28

3.5 Classes do compilador JNS 29

3.6 Compilador JNS 31

3.7 Extensão do gedit para JNS 31

4.1 Diagrama da Folha de estilo ncl4web.xsl 36

4.2 Exemplo do problema que impede o uso de classes em regiões 37

4.3 Fluxograma da tradução do NCL em HTML5 39

4.4 Fluxograma da interpretação do corpo do documento 40

4.5 Fluxograma da tradução da regra 41

4.6 Fluxograma da tradução da região 43

4.7 Fluxograma da tradução da região 44

4.8 Fluxograma da tradução da mídia 45

4.9 Fluxograma da interpretação dos elos 47

4.10 Diagrama geral de uma função de ação 50

4.11 “Viva Mais"sendo executado no webNCL(acima) e no NCL4WEB(abaixo) . 56

Lista de Tabelas

2.1 Mapeamento de condições e ações do TAL [39] 11

4.1 Correspondência entre elementos NCL e sua forma em HTML5 com o NCL4WEV 37

A.3 Parâmetros de um connector 72

A.3 Parâmetros de um connector 73

A.3 Parâmetros de um connector 74

A.1 Parâmetros do descritor 87

A.2 Tipos e Subtipos de transições possíveis 88

A.4 Tipos possíveis a uma mídia 88

Lista de Abreviaturas e Siglas

AJAX	:	ASP+JavaScript+XML;
ASP	:	Active Server Pages;
DOM	:	Document Object Model;
HTML	:	HyperText Markup Language;
JNS	:	JSON NCL Script;
JSON	:	JavaScript Object Notation;
NCM	:	Nested Context Model;
NCL	:	Nested Context Language;
SMIL	:	Synchronized Multimedia Integration Language;
SVG	:	Scalable Vector Graphics;
TAL	:	Template Authoring Language;
W3C	:	World Wide Web Consortium;
XSLT	:	Extensible Stylesheet Language Transformations;
XML	:	eXtensible Markup Language;
YANL	:	YAML Ain't Markup Language;

Sumário

1 Introdução **1**

1.1 Motivação 1

1.2 Objetivos 2

1.3 Estrutura da Dissertação 3

2 Trabalhos Relacionados **4**

2.1 NCL 3.1 4

2.2 NCLRaw 5

2.3 Luar 6

2.4 SMIL State 7

2.5 WebNCL 8

2.6 SmilingWeb - SmileWebPlayer 9

2.7 TAL 10

2.8 Comentários Finais 12

3 JSON NCL Script - JNS **13**

3.1 JSON 14

3.2 Facilidades de JNS 15

3.2.1 Exemplos de JNS 23

3.2.2 JNS e NCL 26

3.2.3 Ferramentas de Apoio à Autoria com JNS 29

4 NCL4WEB **33**

4.1 Decisões de projeto 33
4.2 Trabalhando com XSLT 35
4.3 Descrevendo a folha de estilo ncl4w.xslt 35
4.4 Tradução do NCL para HTML5 36
4.4.1 Interpretando o cabeçalho(*head*) 39
4.4.2 Interpretando o corpo(*body*) 40
4.4.3 Interpretador de Regras 41
4.4.4 Interpretador de Portas 42
4.4.5 Interpretador de Propriedades 42
4.4.6 Interpretador de Contexto 42
4.4.7 Interpretador de Switch 42
4.4.8 Interpretador de Regiões 43
4.4.9 Interpretador de Descritores 43
4.4.10 Interpretador de Mídia 44
4.4.11 Interpretador de *Switch* de Descritores 45
4.4.12 Interpretador de Elos 46
4.5 Funcionamento do HTML5 resultante 47
4.5.1 Start 49
4.5.2 Stop e Abort 51
4.5.3 Pause e Resume 52
4.5.4 Set 53
4.5.5 Select 53
4.5.6 Tratamento de eventos 54
4.6 Uso e Comparação com o webNCL 54
4.7 Discussões, limitações do projeto e trabalhos futuros 57
4.8 Conclusão 59

Referências **61**

Apêndice A – Documentação JNS **65**

A.1 Head 65

A.1.1 Region 65

A.1.2 Descriptor 66

A.1.3 Transition 68

A.1.4 Rule 69

A.1.5 DescriptorSwitch 70

A.1.6 Connector 71

A.1.7 Include 75

A.1.8 Meta e metadata 76

A.2 Body 77

A.2.1 Media 77

A.2.2 Property 78

A.2.3 Area 79

A.2.4 Port 80

A.2.5 Link 81

A.2.6 Context 84

A.2.7 Switch 85

Anexo A – Linguagem NCL **89**

Capítulo 1

Introdução

1.1 Motivação

A televisão digital propõe mudanças na transmissão do sinal e em padrões de codificação do conteúdo a ser transmitido. Através da mesma, não só se tem um ganho na qualidade de imagem e som, mas se permite um novo nicho de aplicações que sejam interativas e personalizadas para o telespectador. Graças às leis brasileiras, a partir de janeiro 2013 [12], as empresas são obrigadas a fabricar 75% dos televisores com conversores digitais que sigam o SBTVD (Sistema Brasileiro de Televisão digital). É previsto que o mercado para aplicações de televisão digital irá crescer de modo incrivelmente rápido [44]. No entanto, apesar de um crescente mercado, as aplicações para televisão digital ainda não alcançaram índices popularidade e uso expressivos [8].

Desde a primeira transmissão de televisão digital no Brasil[6] em dezembro de 2007, não existem muitos aplicativos que explorem as funcionalidades do SBTVD. Nem mesmo propagandas eleitorais ou anúncios do governo são feitos com essas funcionalidades. Uma justificativa para isto é que muitos localidades no país ainda não possuem a transmissão digital. Outra justificativa é que nem todas as televisões com conversores digitais presentes nas casas dos telespectadores fazem uso do middleware Ginga [2], sistema parte do SBTVD que permite as aplicações interativas. O middleware Ginga é composto por duas partes: o Ginga-J [40], oferecendo aplicações procedurais em JavaDTV, e o Ginga-NCL [38], oferecendo aplicações declarativas em NCL - Nested Context Language. Essas são justificativas são verdadeiras, embora possam não sejam válidas. Por exemplo, no contexto dos smartphones, nem toda a população tem este tipo de dispositivo e isso não impede as empresas de investirem cada vez mais em aplicações de smartphones focadas no mercado brasileiro [27].

Na web, qualquer pessoa pode fazer alguma aplicação e publicar para que o mundo veja. Existem várias linguagens e ferramentas que facilitam a autoria, enquanto, na televisão, além de convencer a emissora sobre a utilidade ao mercado, ainda é preciso desbravar o mundo da programação para televisão digital, o qual atualmente tem somente duas opções de linguagens a serem utilizadas, o Java[20] e o NCL[35]). Não existe a possibilidade de se publicar uma aplicação sem o auxílio de uma emissora de televisão, ou de um transmissor de sinal de TV. Na internet, qualquer pessoa pode fazer algo e publicar para que o mundo veja, existem várias linguagens e ferramentas que facilitam tal coisa, enquanto, na televisão, além de convencer a emissora sobre a utilidade ao mercado, ainda é preciso desbravar o mundo da programação para televisão digital (o qual hoje tem somente duas opções o Java[20] e o NCL[35]). Não existe a possibilidade de se publicar uma aplicação sem o auxilio de uma emissora de televisão, ou de um transmissor de sinal de televisão.

Em outros sistemas de TV digital, as aplicações mais usuais são criadas no padrão JavaTV. O JavaTV está presente em parte no middleware Ginga-J[40], em um formato que é livre de royalties: o JavaDTV[19].Além do JavaTV o JavaDTV contém bibliotecas específicas do middleware Ginga. Embora, a princípio, a escolha mais comum seja o Ginga-J, o Ginga-NCL tem um poder maior de alcance. O Ginga-NCL é o padrão já adotado em outros países da América Latina (por exemplo, na Argentina [43]) e por ter sido escolhido pelo ITU (União Internacional de Telecomunicações) como padrão para transmissão de serviços IPTV [36].

Entretanto, linguagens declarativas não são tão atrativas aos desenvolvedores acostumados a programação procedural. De fato, NCL se torna um novo padrão a ser aprendido pela comunidade. Embora ele não seja muito complexo, o tamanho final dos códigos do programa pode atrapalhar o entendimento de quem está iniciando na linguagem como observado em [34].

1.2 Objetivos

O objetivo deste trabalho é propor soluções que facilitem a autoria de documentos NCL e ajudem a divulgar as aplicações NCL na web. A primeira proposta desta dissertação é o JNS, uma linguagem que torna a autoria de um documento NCL mais fácil de ser realizada por um programador acostumado com linguagens procedurais. O JNS oferece uma forma alternativa de descrever um documento NCL através de um objeto JSON -

Java Script Object Notation [10], o padrão de descrição para objetos JavaScript. Com o uso de JNS, espera-se reduzir a verbosidade da linguagem NCL e simplificar algumas estruturas do NCL, seguindo ideias de outros trabalhos publicados na literatura [34]. Também são propostos um compilador JNS, capaz de transformar um documento JNS em um documento NCL, e uma extensão para o editor de textos gedit [28], para auxiliar o desenvolvimento com a nova linguagem.

A segunda ferramenta proposta desta dissertação é o NCL4WEB, que facilita a divulgação do trabalho feito com a linguagem NCL. O NCL4WEB é capaz de transformar um código NCL [35] em um código HTML5 [47] usando o padrão XSLT - eXtensible Stylesheet Language Transformation [46] e funções JavaScript, de forma que qualquer navegador web que leia o arquivo NCL com a referência a folha de estilo XSLT, possa exibi-lo como um arquivo HTML5. Isso permite ao desenvolvedor divulgar o seu trabalho mais facilmente na web, conseguindo assim atrair a atenção para colocar o seu projeto em produção.

1.3 Estrutura da Dissertação

A estrutura deste trabalho é a seguinte: no Capítulo 2 serão apresentados trabalhos relacionados, que ajudaram a desenvolver as ideias propostas nesta dissertação, expondo contribuições complementares aos mesmos, e explorando soluções até então não exploradas.

No Capítulo 3, será apresentado o formato JNS, mostrando o que é o padrão JSON, como é a estrutura do JNS e as novas funcionalidades que o JNS proporciona ao programador. Será apresentado o compilador JNS, que traduz o código para NCL e a extensão do editor gedit, usado para auxiliar o programador. Por fim, divulga-se o resultado de testes de usabilidade de JNS, seguindo os critérios de usabilidade de Nielsen [4].

No Capítulo 4, será apresentado o NCL4WEB e como o mesmo foi projetado. Em seguida, é discutido como a transformação para HTML5 é realizada, e de que forma ocorre o funcionamento da emulação do formatador NCL, feito pelo JavaScript. Ao final, são mostrados os resultados e testes de carga de memória em comparação com outro sistema que tem objetivo similar. No Capítulo 4.8, serão discutidas as contribuições e sugestões para trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Neste capítulo, serão abordados alguns trabalhos que se relacionam com as propostas desta dissertação: alguns deles (NCL3.1, NCLRaw, Luar e SmilState) se relacionam com a proposta de JNS e outros (WebNCL, SmilPlayer, Smil timesheets e TAL) se relacionam com a proposta de NCL4WEB. Um resumo desses trabalhos, suas principais contribuições e uma comparação com as propostas desta dissertação são apresentados nas seções a seguir.

2.1 NCL 3.1

O NCL 3.1 [34] propõe uma nova versão para linguagem NCL. É proposta uma nova especificação para NCL, mantendo as funcionalidades da versão anterior e facilitando o desenvolvimento de alguns elementos de linguagem. O trabalho comça descreve a facilidade de reúso na linguagem NCL e alguns problemas acarretados pela mesma como as cadeias de dependência entre elementos da linguagem, a dependência oculta e que documentos muito grandes podem gerar problemas de visibilidade da parte útil do código, pelo fato de estar dividido em documentos diferentes. São propostas algumas soluções para tais problemas com o NCL 3.1 Enhanced DTV Profile. São discutidas as soluções adotas primeiramente pela inclusão de atributos na mídia NCL e segundo pela nova especificação dos elos NCL. No elemento mídia do NCL3.1, todos os atributos podem ser definidos como propriedades, incluindo parâmetros de navegação (antes presentes apenas nos descritores) e a capacidade de declarar transições dentro das propriedades da mídia. Desta forma os problemas de cadeias de dependências entre elementos são minimizados.

O trabalho comenta sobre o problema de não existir um modo de fazer o reúso de regras, mas não traz uma solução para o tal. A última solução apresentada é para os

elos que passam a poder declarar o seu conector internamente através de uma função no formato "condList **then** actList **end**" que fica no interior do elo. Com essa funcionalidade, espera-se evitar os problemas de cognição visual causados pela definição de relações previamente declaradas. O artigo mostra a especificação da função e apresenta alguns exemplos de funções em elos e seus equivalentes em NCL 3.0. Os exemplos mostram elos não só substituindo conectores mas também *switches* ateravés de uma estrutura alternativa com um contexto e conectores que façam o mesmo que o switch. Por fim, são discutidas as mudanças necessárias no formatador NCL para que o mesmo seja capaz de entender o NCL 3.1, o que é basicamente acrescentar acesso a mídia para os novos atributos e incluir um tradutor das novas funções do elo.

A principal contribuição do trabalho é a possibilidade de especificação de conectores dentro dos elos através de expressões lógicas. Essas expressões serviram de inspiração para expressões usadas nos conectores, elos, regras e *switches* em JNS. O trabalho fala de como a sintaxe atual do NCL 3.1 atrapalha um pouco a compreensão e cognitividade do desenvolvedor, algo que o JNS tenta resolver diminuindo o tamanho do código e permitindo uma melhor visualização do mesmo pelo usuário. A funcionalidade de evitar cadeias de dependência foi resolvida em JNS permitindo a declaração de elementos internamente a outros. Em JNS, é possivel declarar um região dentro de uma mídia por exemplo. Ao contrário deste trabalho, o artigo só apresenta de modo teórico a nova especificação sem implementar um tradutor ou interpretador NCL para a mesma. Para o JNS, foi criado um compilador capaz de transformar o código JNS em código NCL, que será apresentado no Capítulo 3.

2.2 NCLRaw

O trabalho [24] propõe o NCL Raw Profile, que descreve um subconjunto da linguagem NCL, que facilita o desenvolvimento de formatadores NCL. Dessa forma, o programa transmitido pelo servidor de conteúdo utilizaria essa linguagem, simplificando o trabalho do formatador NCL e permitindo maior compatibilidade com outras linguagens. O trabalho mostra as estruturas básicas do NCL Raw Profile, eliminando praticamente todo o cabeçalho do NCL EDTV, deixando somente os elementos de meta e metadata, e usando somente mídias, portas, propriedades, contextos e elos no corpo. Todos os ítens do cabeçalho se tornam propriedades na mídia e os *switches* são substituídos por uma estrutura equivalente formada por contextos e elos. Os elos, assim como no NCL 3.1, usam o conector com uma expressão interna. O trabalho explica teoricamente como ocor-

reria a conversão do NCL atual para o RawProfile, do RawProfile para o HTG [9]e da possibilidade de traduzir outras linguagens para o RawProfile. O artigo conclui que este é um passo em direção ao NCL 4.0. O trabalho não apresenta compiladores do NCL para o RawProfile nem prova que a interpretação do mesmo é de fato eficiente em dispositivos movies, como afirmado no texto.

O trabalho se relaciona a JNS pelo fato de buscar uma arquitetura mais simples para NCL, eliminando as redundâncias da linguagem. No entanto, o JNS não tem a intenção de modificar a implementação do formatador NCL. Assim como NCL Raw profile, é possível criar documento JNS sem precisar criar um cabeçalho, porém, isso é algo opcional e não a única possibilidade. A linguagem JNS tem o objetivo de facilitar a autoria e aumentar o poder de expressão de um documento NCL sem perder nenhuma das funcionalidades de reúso da linguagem NCL. Além disso, esta dissertação apresenta um compilador funcional de JNS para NCL 3.0, a especificação padrão de NCL no SBTVD.

2.3 Luar

Luar [5] é uma linguagem para definição de templates. Através desta, é possível criar templates completos, templates de componentes e mesmo criar documentos sem usar templates. A idéia básica é inserir um código Lua [17] em um documento NCL [35], de forma similar a um código PHP em um documento HTML. O funcionamento do Luar é baseado em um documento inacabado Luar que será combinado com um template Luar para então ser processado em um documento NCL final. Os documentos NCL podem ser criados de três maneiras no Luar: por templates completos, por templates de componentes e sem templates. Utilizando templates completos, o documento recebe o nome do template e os parâmetros do mesmo, assim um documento NCL é gerado pelo processador Luar. Nos templates de components, pode-se usar mais de um parâmetro no template e o documento pode usar mais de um template. Desta forma, o documento pode ser criado por vários templates que criam cada parte do documento NCL. Um documento sem templates usa o próprio código Lua para gerar o código NCL através da instrução “print”, aproveitando as facilidades das estruturas de repetição de Lua. O *kernel* Luar é composto por um processador de templates, responsável pela interpretação de templates e geração de documentos criados por templates completos, e o processador de aplicação, que interpreta os templates de componentes chamando o processador de template para cada template no documento. Além disso, o processador de aplicação interpreta os documentos sem templates executando o código Lua interno a eles. O *kernel*

sempre retorna um documento NCL como saída para os documentos de entrada. O kernel tem acesso aos templates criados pelo usuário e a uma base de templates externos de um portal chamado GingaCDN[1], um portal que já é usado para elementos do Ginga-J que permite aos autores compartilhar os templates criados. O artigo conclui apresentando o desejo de tornar o Luar parte do Composer e de ouvir o feedback dos desenvolvedores para avaliar a efetividade da linguagem.

O JNS não é uma linguagem de definição de templates, mas se relaciona ao Luar no ponto de ser uma linguagem de script, algo que o Luar permite parcialmente oferecendo templates compostos. JNS também usa uma sintaxe mais familiar para desenvolvedores de linguagens procedurais tal qual Luar. No entanto, ao contrário de Luar, não requer que o desenvolvedor tenha conhecimento completo de NCL para criar um documento, algo que o Luar só permite com a ajuda de templates. JNS facilita a criação de um documento NCL simplificando sua sintaxe ao invés de requerer um template para cada aplicação.

2.4 SMIL State

O SMIL State [18] é uma extensão de linguagem capaz de trazer a SMIL [7] ou a SVG [13] a capacidade de guardar o estado de variáveis, permitindo anexar modelos de dados aos documentos, que podem ser manipulados para adaptação do conteúdo. O artigo apresenta um cenário de uma aplicação que faz um tour por uma cidade. A idéia é usar alguma ferramenta que permita ao usuário personalizar o seu tour. O artigo apresenta os requisitos para a aplicação e em seguida apresenta as tecnologias existentes e o que cada uma é capaz de fazer, comparando SMIL com Flash, JavaScript e com a solução apresentada, chamada SMIL State. O SMIL State utiliza ferramentas já disponíveis para funcionar com SMIL 2.1, e acrescenta uma ferramenta capaz de ler dados externos e interagir com o usuário, dependendo do momento. O suporte a Smil State foi implementado em um formatador SMIL em código aberto, o Ambulant [42], e testada em um navegador, o Safari, via plugin. O artigo mostra aplicações do SMIL State, como uma propaganda atrasada, onde uma ação do usuário pode determinar uma propaganda diferente aparecer, algo feito com ajuda das variáveis do SMIL State. O artigo conclui mostrando que SMIL State foi incorporado na última versão SMIL 3.0 [7] e demonstrando o desejo de integrar a ferramenta a outros componentes declarativos no ambiente web.

O JNS também acrescenta mais poder de expressão a autoria NCL, permitindo o uso

[1]Ginga CDN - Code Development Network http://gingacdn.lavid.ufpb.br/ acessado em 10 de março de 2013

de variáveis temporárias em regras, possibilitando que uma mesma regra seja reusada com diferentes variáveis. Pode-se dizer que a idéia de usar SMIL dentro de um navegador via plugin, também se relaciona ao NCL4WEB pelo fato de se basear nas tecnologias do W3C para trazer melhorias e implementar linguagens declarativas que descrevem documento multimídia no ambiente web. O SMIL State requer uma nova forma de interpretar o SMIL para que as variáveis funcionem. Entretanto, neste trabalho, as regras com variáveis temporárias se traduzem em várias regras NCL, uma para cada variável que a usou, tornando o código resultante compatível com a versão do NCL atual, necessitando apenas de uma transformação do código JNS para NCL.

2.5 WebNCL

WebNCL [26] é um framework JavaScript [11], capaz de ler um documento NCL e executar o mesmo em um navegador através do HTML5 [47]. O artigo começa mostrando as decisões de projeto em usar o JavaScript ao invés de usar o Flash. São expostas as bibliotecas auxiliares utilizadas como a PopCorn.js [2] e o JQuery [29]. O artigo apresenta a arquitetura do framework em três camadas. A primeira camada é responsável pela leitura: composta por um componente para ler o documento, e outro para estruturá-lo em um objeto NCL. A segunda camada gerencia o controle de eventos do arquivo, composta por cinco componentes: um responsável pelo controle dos contextos, um gerenciador de eventos, um gerenciador de interações, um gerenciador de exibição e um gerenciador de sincronismo. A última camada é responsável pela exibição do documento e é composta por quatro componentes: um componente para cada tipo de mídia, um para áudio outro para vídeo, um para texto e um para HTML. Para iniciar uma apresentação, a camada de contextos transforma cada elemento NCL em um objeto JavaScript. Os links têm o auxílio de um gerenciador de sincronismo responsável por guardar cada elo e executar o elo correspondente a ação enviada assincronamente, além de gerenciar pausas geradas por falhas no carregamento do vídeo. Para utilizar o WebNCL em uma página web é necessário invocar a biblioteca via JavaScript e iniciar o documento com a função WebNClPlayer que recebe como entrada o documento NCL e o id de um elemento div dentro da página HTML. É mostrada como a interação é feita através do controle remoto ou de teclas predefinidas pelo WebNCL, que incluem o mapeamento das teclas coloridas, numéricas, setas e o enter. Por fim, são discutidas as limitações do WebNCL em relação ao dispositivo que o executa e ao navegador que o utiliza. O trabalho conclui colocando como trabalho

[2]Popcorn.js -disponível em http://popcornjs.org/, Acesso em 11 de março de 2013

futuro a implementação de código Lua[17] usado dentro do WebNCL.

A contribuição mais importante de WebNCL é a possibilidade de execução de um documento NCL em um browser web. Além disso, WebNCL usa a biblioteca JQuery, assim como NCL4WEB, embora, também faça uso do Popcorn, que não foi usado no NCL4WEB, visto que o HTML5 já tem eventos que podem ser usados para criar os eventos do NCL. São comentados testes de funcionamento em múltiplas plataformas, em que se expõe ter se obtido um resultado de desempenho razoável nos mesmos.

A principal diferença do NCL4WEB em relação ao WebNCL está no fato de realizar uma transformação inicial do documento em HTML5, usando XSLT. O JavaScript é usado somente para o controle e sincronismo dos elementos. Isso traz melhorias de performance em relação ao WebNCL, que podem ser vistas em 4.6, além de permitir a utilização do código já traduzido (pré-compilado). O WebNCL também usa um mapeamento de teclas padrão e um teclado virtual para mapear os eventos do controle remoto, limitando a tela usada pela aplicação. Enquanto o NCL4WEB usa uma abordagem diferente, através de uma tela de configuração inicial, onde o usuário mapeia somente as teclas do controle remoto que serão usadas na aplicação. O WebNCL não apresenta e não comenta nada sobre a interpretação das regras e switches do NCL, enquanto as mesmas são interpretadas pelo NCL4WEB.

2.6 SmilingWeb - SmileWebPlayer

O SmilingWeb [14] é um player para SMIL que funciona dentro do navegador web, e usa uma biblioteca JavaScript [11] para controlar a apresentação. O Smiling web é apresentado, seu funcionamento consiste no uso do HTML com AJAX[15] e a biblioteca JQuery [29]. É possível se fazer as transições do SMIL com uma nova propriedade no CSS3, mas como nem todos os navegadores suportam o CSS3 a transição é feita através do JavaScript com a biblioteca JQuery. O SmilingWeb não interpreta as tags regPoint e priorityClass e todos os atributos ligados ao controle de volume do SMIL. É explicado como funciona o escalonador responsável por controlar as relações de sincronismo entre o início e fim de uma mídia. Assim o escalonador, calcula esses tempos fazendo uma busca por todos os elementos internos a mídia, colocando o resultado em uma tabela que mapeia todos os tempos de início e de fim. O sistema foi testado de duas maneiras: uma testando somente a capacidade de exibição e a outra testando a capacidade de sincronismo do escalonador. Em ambos os testes, o sistema obteve resultados satisfatórios. O artigo conclui mostrando

que o SmilingWeb permite a sincronização de páginas HTML sem a necessidade do conhecimento de AbobeFlash ou JavaScrip, além disso, mostra como a utilização de SMIL com XHTML possibilita vantagem quanto a exibição, permitindo flexibilidade de layouts automáticos que se adaptam a tela do usuário.

O NCL4WEB deixa o trabalho de escalonar eventos para funções JavaScript, criadas pelos elos NCL na transformação de NCL para HTML5 e ativadas por eventos dos elementos HTML no JavaScript. A capacidade de exibição de mídias é mais dependente do navegador que da ferramenta, como o próprio artigo do SmillingWeb cita. Testes realizados com NCL4WEB usam aplicações já conhecidas do repositório ClubeNCL[1], ao invés de aplicações criadas pelo próprio desenvolvedor como no SmilingWeb.

2.7 TAL

TAL [39] é uma linguagem de templates capaz de traduzir os seus templates para códigos NCL ou HTML5. Assim como NCL, a linguagem TAL também é baseada em XML. O artigo explica o funcionamento do TAL e a diferença entre criação de templates e documentos. Um documento que usa um template é processado para gerar um documento final. O artigo mostra a estrutura de um template TAL que contém: um vocabulário, que contém as relações e tipos a serem usados no template; as restrições, que definem regras sobre as classes definidas no vocabulário; os recursos, que definem objetos filhos que devem ser comuns a todos as composições criadas pelo template; e as relações, que definem relações comuns entre as classes, relacionando objetos filhos e recursos que deverão ser herdados por todas as composições que seguem o template. O artigo então apresenta duas aplicações que tem o mesmo princípio, ser um quiz onde o usuário tem que escolher uma resposta enquanto passa um vídeo, mostrando como funciona o template para as mesmas. É possível personalizar o log de mensagens do template através dos elementos *assert, report* e *warning*. Tais elementos especificam regras que dizem como devem ser as mensagens de erros ou avisos em determinados casos, assim como o Schematron [21] faz. O artigo mostra exemplos de documentos escritos como TAL explicando o funcionamento de cada parte deles. O processador TAL funciona com a ajuda de um interpretador, que é responsável por ler o documento que usa um template e o próprio template e traduzi-los em objetos para serem processados pelo processador interno e um processador estendido capaz de traduzir o TAL em NCL ou HTML5. Por fim um sintetizador escreve o documento final. É descrito um frontend criado para facilitar ao usuário o trabalho de usar o template para traduzir documentos incompletos em documento HTML ou NCL. É de-

Tabela 2.1: Mapeamento de condições e ações do TAL [39]

	TAL	NCL	HTML
Actions	Start	start	play/show
	Stop	stop	stop/hide
	Pause	pause	pause
	Resume	resume	play/show
	Abort	abort	abort
	Set	set	Script handling
Condition	onEnd	onEnd	Ended
	onBegin	onBegin	Playing
	onAbort	onAbort	Abort
	onPause	onPause	Pause
	onResume	onResume	Playing
	onEndAttribution	onEndAttribution	Script handling
	onBeginAttribution	onBeginAttribution	Script handling

scrita a transformação do TAL para NCL e HTML. A Tabela 2.7 mostra algumas relações das ações e condições na linguagem TAL para NCL e HTML. Note que as condições e ações de TAL são as condições e ações pré-definidas de NCL. O artigo comenta sobre a futura integração de TAL a ferramenta de autoria Composer [22] e sobre testar futuramente a linguagem com usuários não especialistas. Além disso, o artigo comenta a possibilidade de usar o TAL no dispositivo cliente, desta forma, o cliente é usado para gerar documentos dinâmicos ao invés de somente receber documentos pré-gerados pelo TAL. Neste tipo de cenário, seria necessário modificar o middleware GINGA, considerando o uso de TAL junto a NCL para aplicações de TV digital.

A abordagem de NCL4WEB é similar no quesito de se aproveitar eventos do HTML5, no entanto, em NCL4WEB, esses eventos chamam outros eventos customizados com o mesmo nome das condições de ação padrão do NCL 3.0. A estrutura usada para transformação do TAL também se assemelha a do NCL4WEB, porém, o NCL4WEB tem a vantagem de usar XSLT[46],o que torna a transformação transparente ao usuário. Além disso, a referência [39] sobre TAL não comenta se traduz os switches e regras do NCL, algo que o NCL4WEB faz. Apesar de não se especificar como o TAL traduz os nós de composição, ele, assim como o NCL4WEB, faz uso do *span* e do *div* para a organização interna do documento HTML final.

2.8 Comentários Finais

Considerando os trabalhos apresentados, nenhum se propôs a apresentar uma sintaxe alternativa a uma linguagem de autoria multimedia, como a proposta de JNS. Nenhum deles utiliza XSLT para fazer uma transformação completa de documentos NCL da mesma forma como NCL4WEB propõe. Como será mostrado no Capítulo 3, o JNS procurou manter todas as funcionalidades do NCL 3.0, incorporando novas facilidades para autoria. Por usar XSLT, NCL4WEB tornou transparente a conversão de NCL para HTML5 para o usuário final. Além disso, não foram encontrados trabalhos que usassem o JSON como base para uma linguagem de autoria multimídia, como o JNS o faz.

Capítulo 3

JSON NCL Script - JNS

Com o objetivo de incentivar o uso de uma linguagem declarativa para autoria multimídia entre autores que possuem alguma experiência em programação, esta dissertação propõe a linguagem JNS (JSON NCL SCRIPT)[?]. JNS é uma forma alternativa de se escrever um documento NCL. A linguagem NCL permite uma completa gama de opções para a criação de um documento hipermídia, oferecendo reúso de regiões, descritores, conectores, regras e nós. Entretanto, aplicações simples são penalizadas com a verbosidade da linguagem, pois para declarar cada elemento de mídia, criar elos e switches, é necessário declarar muitos elementos, fazendo o código NCL ficar extenso. Por exemplo, quando se usa apenas um elo no documento, seria prático declarar a semântica da relação (definida pelo conector) diretamente na definição do elo. Entretanto, NCL não dá esta opção ao autor, sendo necessário definir uma base de conectores, que por sua vez define o conector.

A idéia de uma linguagem de script alternativa a NCL, não era somente permitir que uma aplicação pudesse ser feita especificando menos elementos. Analisando trabalhos relacionados, outro objetivo foi incorporar novas facilidades a linguagem não só para torná-la menos verbosa, mas tornar o desenvolvimento de uma aplicação mais ágil. Tudo isso é feito oferecendo as funcionalidades originais da linguagem NCL, quando o autor utiliza JNS para autoria.

Ao especificar a linguagem JNS, procurou-se observar sempre a redução e eliminação de redundâncias na especificação de um programa. A idéia é prover um formato textual simplificado para programação de conteúdo multimídia que seja amigável a usuários acostumados com programação procedural, mas sem perder o foco de que é uma linguagem declarativa. Para se manter o paradigma declarativo, foi utilizado o padrão JSON(JavaScript Object Notation) [10], como base para a linguagem. Em vários testes

em ambientes web, o padrão JSON se mostrou menos verboso que a linguagem XML, e ambos são usados para descrição de dados semi-estruturados.

Este capítulo apresenta brevemente a notação JSON, discute detalhadamente a linguagem JNS e apresenta resultados de testes de usabilidade da linguagem e duas ferramentas de apoio à autoria JNS, o compilador JNS e uma extensão ao editor de texto gedit.

3.1 JSON

JSON (*JavaScript Object Notation*) [10] é parte da segunda versão do padrão ECMAScript [**?**]. ECMAScript descreve dois tipos básicos de estrutura: o objeto e o *array*. O objeto é uma estrutura que contém tuplas de nome de um atributo e seu valor, e o *array* é um conjunto de vários valores iguais ou diferentes. Para se delimitar um objeto são usados colchetes "{ }", suas tuplas são ligadas por dois pontos ":" e seus atributos são separados por uma vírgula ",". Para se delimitar um *array* são usadas chaves "[ ]" e cada um de seus valores são separados por uma vírgula ",".

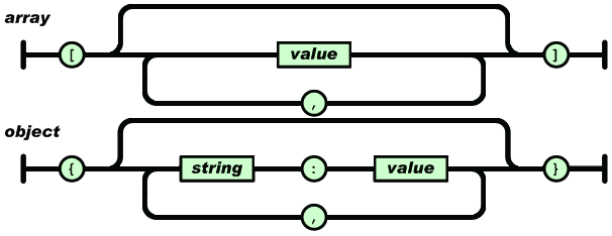

Figura 3.1: Estruturas do JSON

Além das duas estruturas, o padrão ECMAScript especifica sete valores possíveis: as estruturas objeto e *array*; um número que pode ser real ou inteiro; uma *string*, que deve sempre conter aspas; um booleano, que é uma *string true* ou *false* sem aspas; uma função, que é uma *string* que se refere a uma função do Script; e o nulo, que é a *string null*. No padrão ECMAScript, os nomes dos atributos não precisam de aspas como as strings, somente no caso em que os mesmos coincidem com o nome de uma função, porém, o padrão JSON adotado em 2002, exclui as funções como valores possíveis, logo, todas as strings incluindo os nomes de atributos de um objeto precisam receber as aspas.

O padrão JSON, adotado em 2002 como forma de transferência de dados, não teve muita popularidade, mas se tornou largamente usado após a criação do AJAX [15] em 2006. O nome AJAX vem de ASP+JavaScript+XML, onde o ASP seria responsável por realizar as funções mais complexas dentro de um servidor web, permitindo que páginas

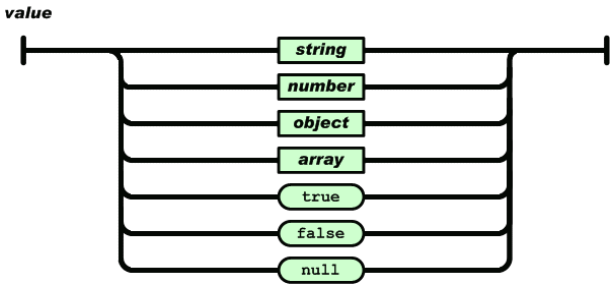

Figura 3.2: Valores do JSON

web usem JavaScript para fazer chamadas a outras páginas e se conectar a bancos de dados, oferecendo dessa forma uma página dinâmica. O JavaScript faria a parte dinâmica para o cliente web e o XML seria a linguagem de transferência de dados das páginas em ASP para o script em JavaScript. Porém, logo se percebeu que era mais fácil manipular um objeto JSON que já era nativo ao JavaScript, do que manipular um objeto XML. Dessa forma, o padrão JSON se popularizou e passou a ser usado em outras aplicações como banco de dados semi-estruturados [16], linguagem para transporte de dados em outros ambientes [31], e agora neste trabalho, a mesma será usada pela primeira vez como base para uma linguagem de autoria multimídia.

3.2 Facilidades de JNS

No intuito de manter um meio termo entre uma mudança completa de paradigma e um script auxiliar, a linguagem JNS manteve as mesmas estruturas básicas do NCL. Um programa JNS define um cabeçalho e um corpo, entretanto, boa parte do cabeçalho NCL se tornou opcional, permitindo o uso de menos estruturas para construção de um programa. A Listagem 3.1 mostra a estrutura básica de um programa JNS.

Listagem 3.1: Estrutura do JNS

```
1  {ncl:{
2          id:"stringLivre" ,
3          title: stringLivre ,
4          head:[
5                  {region:{?}},
6                  {descriptor:{?}},
7                  {rule:{?}},
8                  {descriptorSwitch:{?}},
9                  {connector:{?}},
10                 {transition :{?}},
11                 {include:{?}},
12                 {meta:{?}},
13                 {metadata: RDF tree} ,
14         ] ,
```

```
15          body:[
16                  {media:{?}},
17                  {link:{?}},
18                  {context:{?}},
19                  {switch:{?}},
20                  {property:{?},
21                  {meta:{?}},
22                  {metadata: RDF tree} ,
23                  {port: {"idsDaPorta": idDono }} // ou [{ id : no },...]
24          ]
25  }}
```

A especificação completa da linguagem JNS pode ser encontrada no Apêndice A.

A linguagem JNS traz algumas funcionalidades ao NCL além de usar menos linhas com o JSON. São apresentadas aqui algumas destas funcionalidades, espera-se com estas melhorias que a criação e edição de programas para a linguagem NCL se torne mais amigável, ágil e fácil para usuários com experiência em programação. A primeira funcionalidade do JNS é a omissão da declaração de bases para os elementos no cabeçalho. Assim regiões, descritores, regras, conectores e transições são diretamente declaradas no cabeçalho sem a necessidade de estar dentro de uma base. Desta forma, economiza-se o número de estruturas necessárias a um documento multimídia. A Listagem 3.2 exibe esta funcionalidade comparando a definição do cabeçalho de um documento em JNS e NCL.

Listagem 3.2: Cabeçalho em JNS e NCL

```
1
2  {ncl:{
3
4           head:[
5                   {region:{?}},
6                   {descriptor:{?}},
7                   {rule:{?}},
8                   {descriptorSwitch:{?}},
9                   {connector:{?}},
10                  {transition:{?}},
11          ]
12
13 }}
14
15 **Em NCL
16
17 <ncl>
18          <head>
19                  <regionBase>
20                          <region ... />
21                          ...
22                  </regionBase>
23                  <descriptorBase>
24                          <descriptor ... />
```

```
25                      <descriptorSwitch ... />
26                      ...
27                  </descriptorBase>
28                  <ruleBase>
29                      <rule ... />
30                      ...
31                  </ruleBase>
32                  <connectorBase>
33                      <connector ... />
34                      ...
35                  </connectorBase>
36                  <transitionBase>
37                      <transition ... />
38                      ...
39                  </transitionBase>
40          </head>
41  </ncl>
```

A segunda funcionalidade é a possibilidade de associar regiões a mídias sem a necessidade de um descritor. Em NCL, toda mídia deve fazer refêrencia a um descritor e este, por sua vez, pode fazer referência a uma região. Em JNS, uma mídia pode referenciar uma região diretamente, caso não seja necessário definir nenhum outro atributo do descritor, a não ser a própria região. Ao ser traduzido para NCL, o código irá conter um descritor para associar a mídia a região correspondente. A Listagem 3.3 demonstra esse conceito com duas mídias referenciando uma mesma região. Em NCL, o código possui duas mídias referenciando um descritor, que se refere a região em questão.

Listagem 3.3: Região diretamente na mídia

```
1
2  {media:{id:"v1", src:"v1.mpg", region:"rV1"}},
3  {media:{id:"v2", src:"v2.mpg", region:"rV1"}},
4
5  ** traduzido em NCL:
6
7  <descriptor id="rV1_Descriptor" region="rV1"/>
8
9  ...
10
11 <media id="v1" src="v1.mpg" descriptor="rV1_Descriptor"/>
12 <media id="v2" src="v2.mpg" descriptor="rV1_Descriptor"/>
```

A terceira é a possibilidade de se declarar as relações condicionais de uma regra na forma de uma expressão dentro do elemento da regra. A Listagem 3.4 mostra um exemplo desta funcionalidade. A expressão da regra pode tanto representar uma regra simples como uma regra complexa. A especificação desta expressão pode ser encontrada no Apêndice A na Subseção A.1.4.

Listagem 3.4: Regra declarada por expressão

```
1
2 {rule:{id:"rEn", expression:"val == ing" }},
3
4
5
6 ...
7
8 {media:{id:"variables" type:"application/x-ginga-settings",
      anchors:[{property:{"i":null}}}]}},
9
10 {switch:{id:"sAi",vars:[{"val":"i"}]
11       "rEn":{media:{id:"aE", src:"En.mp3""}},
12 }}
13
14 ** traduzido em NCL:
15
16 <rule id="rEn" var="i" comparator="eq"  value="ing"/>
17
18 ...
19
20 <media id="variables" type="application/x-ginga-settings">
21         <property name="i"/>
22 </media>
23
24 <switch id="sAi">
25    <bindRule rule="rEn" constituent="aE"/>
26    <media id="aE" src="En.mp3"/>
27 </switch>
```

A quarta é a possibilidade de se declarar regras com variáveis temporárias que serão associadas a variáveis reais no *switch* que usa a regra. Deste modo, é possível reusar regras e eliminar a necessidade de declarar uma variável, que é um elemento do corpo, no cabeçalho, para ter que criar a regra, que é um elemento do cabeçalho. Em NCL, isso gera um potencial problema de cognitividade, qu é comentado em [34], porém não apresentam uma solução para o mesmo. A Listagem 3.5 mostra um exemplo desta facilidade de JNS. Quando uma regra é usada em mais de um *switch*, regras são criadas no NCL resultante com o nome da regra original acrescido do nome da variável usada.

Listagem 3.5: Regras com variáveis temporárias

```
1
2 {rule:{id:"rEn", expression:"val == true" }},
3
4 ...
5
6 {media:{id:"variables" type:"application/x-ginga-settings",
      anchors:[{property:{"ing":null}},{property:{"jump":null}}]}},
7
8 {switch:{id:"sAi",vars:[{"val":"ing"}]
```

```
9         "rEn":{media:{id:"aE", src:"En.mp3""}},
10 }},
11
12 {switch:{id:"sVi",vars:[{"val":"jump"}]
13         "rEn":{media:{id:"vE", src:"En.mp4""}},
14 }}
15
16
17
18
19
20 ** traduzido em NCL:
21
22 <rule id="rEn_ing" var="ing" comparator="eq"  value="true"/>
23 <rule id="rEn_jump" var="jump" comparator="eq"  value="true"/>
24
25 ...
26
27 <media id="variables" type="application/x-ginga-settings">
28         <property name="ing"/>
29         <property name="jump"/>
30 </media>
31
32 <switch id="sAi">
33    <bindRule rule="rEn_sAi" constituent="aE"/>
34    <media id="aE" src="En.mp3"/>
35 </switch>
36
37 <switch id="sVi">
38    <bindRule rule="rEn_sVi" constituent="vE"/>
39    <media id="vE" src="En.mp4"/>
40 </switch>
```

A quinta é a possibilidade de se declarar a regra implicitamente no *switch* ou no *descriptorSwitch* que a usa. Para isso a expressão ocupa o lugar onde ficaria o ID da regra. A especificação desta expressão é a mesma da expressão da regra, no entanto, a variável usada pela expressão tem que ser uma variável presente no corpo do documento. A Listagem 3.6 apresenta esta facilidade com um *switch* e um *descriptorSwitch*. O switch é um elemento que contém outros elementos. Porém, ao receber uma ação (start, stop, set, abort...), essa ação será repassada somente ao elemento cuja regra seja válida (como em um switch na linguagem C). Caso nenhuma regra seja válida, o elemento default será escolhido. As regras usadas dentro do switch não podem usar qualquer propriedade como variável, elas necessitam de um propriedade filha de uma mídia do tipo application/x-ginga-settings (o qual o documento NCL só pode conter um). Essa propriedade é uma variável global e pode ser usada em qualquer regra no documento. O descriptorSwitch funciona da mesma forma, entretanto, ele determina a escolha de um descritor para uma

mídia, que é feita com base em regras.

Listagem 3.6: Regras internas a *switch* e *descriptorSwitch*

```
1
2  head:[
3
4          {descriptorSwitch:{
5                  id:"dSwitch",
6                  "b == 0":{descriptor:{id:"dt0",region:"rg0"}},
7                  "b == 1":{descriptor:{id:"dt1",region:"rg1"}},
8                  default:"dt0"
9          }}
10
11 ]
12
13 body:[
14         {media:{id:"variables" type:"application/x-ginga-settings",
               anchors:[{property:{"a":null}},{property:{"b":null}}]}},
15
16         {switch:{id:"sw",
17           "a == true":{media:{id:"video",descriptor:"dSwitch", src:"video.mp4"}},
18             default:"video"
19         }}
20 ]
21
22
23
24 ** traduzido em NCL:
25
26 <rule id="RulePadrao_0" var="a" comparator="eq"  value="true"/>
27
28 <rule id="RulePadrao_1" var="b" comparator="eq"  value="0"/>
29
30 <rule id="RulePadrao_2" var="b" comparator="eq"  value="1"/>
31
32 <descriptorSwitch id="dSwitch">
33         <bindRule constituent="RulePadrao_1" constituent="dt0"/>
34         <descriptor id="dt0" region="rg0" />
35         <bindRule constituent="RulePadrao_2" constituent="dt0"/>
36         <descriptor id="dt0" region="rg1" />
37         <defaultDescriptor component="dt0" />
38 </descriptorSwitch>
39
40 ...
41
42 <media id="i" type="application/x-ginga-settings">
43 <property name="a"/>
44 <property name="b"/>
45 </media>
46
47 <switch id="sAi">
48         <bindRule rule="RulePadrao_0" constituent="video"/>
49         <defaultComponent component="video"/>
50         <media id="video" descriptor="dSwitch" src="video.mp4" />
```

```
51 </switch>
```

A sexta é a possibilidade de se declarar as relações de conectores com expressões condicionais. A especificação completa da expressão usada no conector pode ser encontrada no Apêndice A na Subseção A.1.6. Essa funcionalidade pode ser vista na Listagem 3.7. Os parâmetros do conector ainda são declarados fora da expressão. Os conectores e elos descrevem as relações de sincronismo e interação entre as mídias e contextos de um documento. Tais relações se baseiam em uma condição (onBegin, onEnd, ...) e uma ação (start, stop, ...).

Listagem 3.7: Conetor declarado por expressão condicional

```
1
2 {connector:{id:"onEnd1StartM", "expression":"onEnd then start with max='unbounded'"}}
3
4 ...
5
6 {link:{xconector:"onEnd1StartM",
7         binds:[{"onEnd":"bVe"},{"start":"v2,sAi"}]
8 }}
9
10 ** traduzido em NCL:
11
12 <causalConnector id="onEnd1StartM">
13         <simpleCondition role="onEnd" />
14         <simpleAction role="start" max="unbounded" />
15 </causalConnector>
16
17 ...
18
19 <link id="repeater" xconnector="onEnd1StartM">
20
21
22 <bind component="bVe" role="onEnd" />
23
24
25 <bind component="v2" role="start" />
26
27
28
29 <bind component="sAi" role="start" />
30
31
32 </link>
```

A sétima é a possibilidade de se declarar a relação de um conector diretamente dentro dos elos através de uma expressão. Neste caso, a sintaxe da expressão do conector é levemente alterada para incluir os elementos que participam do relacionamento. Não existem parâmetros neste tipo de expressão. A especificação completa deste tipo de

expressão pode ser encontrada no Apêndice A na Subseção A.2.5. A Listagem 3.8 mostra esta funcionalidade.

Listagem 3.8: Conector declarado dentro do elo

```
1
2 {link:{expression:"onEnd v1 then start v1"}},
3
4 ** traduzido em NCL:
5
6 <causalConnector id='Connector_Padrao0'>
7     <simpleCondition role='onEnd'/>
8     <simpleAction role='start'/>
9 </causalConnector>
10
11 ...
12
13 <link xconnector='Connector_Padrao0'>
14    <bind role='onEnd' component='v1'/>
15    <bind role='start' component='v1'/>
16 </link>
```

Por fim, ainda é possível declarar uma região dentro de uma mídia. Este açúcar sintático permite criar um documento sem a necessidade de um cabeçalho. A região colocada dentro da mídia não necessita de um id, o mesmo será gerado automaticamente pelo compilador JNS, contendo o nome da mídia como regra de formação. A Listagem 3.9 exemplifica esta facilidade.

Listagem 3.9: Região declarada dentro da mídia

```
1
2 {ncl:{
3         body:[
4                 {media:{id:"media1",src:"vide.mp4", region:{height:"100%",width:"100%"}}}
5         ]
6 }}
7
8 ** traduzido em NCL:
9
10 <ncl id='jnsNCL' xmlns='http://www.ncl.org.br/NCL3.0/EDTVProfile'>
11         <head>
12                 <regionBase>
13                         <region id='Regiao_media1' height='100.0%' width='100.0%'/>
14                 </regionBase>
15
16                 <descriptorBase>
17                         <descriptor id='Regiao_media1_Descriptor'
                                region='Regiao_media1'/>
18                 </descriptorBase>
19         </head>
20         <body>
21                 <media id='media1' src='vide.mp4' descriptor='Regiao_media1_Descriptor'/>
```

```
22          </body>
23  </ncl>
```

Estas funcionalidades carregam consigo algumas pequenas limitações. Por exemplo, o uso de expressões necessita de palavras reservadas (como *with* e *then* no caso dos conectores), o que limita um pouco o uso de nomes para papéis e nós. Também já foram comentadas anteriormente as diferenças entre algumas estruturas como o *include*, que quando usado para importar uma base JNS, requer que nenhum nome usado na base seja usado no documento. Em JNS, há ainda a necessidade de declarar o atributo *parent* na região para que a mesma seja incluída em outra região, visto que no JSON não é possível incluir um objeto em outro do mesmo modo que no XML. No entanto, essas limitações são pequenas se comparadas aos benefícios que a linguagem JNS proporciona, reduzindo o tamanho do código da aplicação e facilitando sua especificação.

3.2.1 Exemplos de JNS

A fim de exemplificar o uso da linguagem JNS, são apresentados três exemplos de documentos nas Listagens 3.10, 3.11 e 3.12.

Listagem 3.10: Exemplo 1 - Documento contendo uma mídia

```
1   {ncl:{
2           body:[
3                   {port:{"ini":"midia1"}},
4                   {media:{id:"midia1",src:"exemplo.mp4",
5                   region:{height:"100%",width:"100%"}}}
6           ]
7   }}
```

A Listagem 3.10 traz um exemplo simples de como a linguagem JNS é utilizada. No exemplo 1, a mídia "exemplo.mp4" é executada, sendo declarada no elemento *media*. No mesmo elemento, é declarada uma região para essa mídia, o que é uma facilidade do JNS. No NCL não é possível declarar uma região no corpo, mas é possível declarar as propriedades de posicionamento e tamanho dentro da mídia através de propriedades internas a mesma.

Listagem 3.11: Exemplo 2 - Âncoras e elos

```
1   {ncl:{
2           body:[
3                   {port:{"porta1":"video"}},
4                   {media:{id:"video",src:"exemplo.mp4",anchors:[
5                           {area:{id:"ponto",begin:"5s",end:"25s"}},
6                           {property:{"visible":"true"}},
```

```
7                       {property:{"bounds":"0%,0%,100%,100%"}}
8               ]}},
9               {link:{expression:"onBegin video.ponto then set video.bounds with
                    value='25%,25%,50%,50%'"}},
10              {link:{expression:"onEnd video.ponto then set video.bounds with
                    value='0%,0%,100%,100%'"}}
11      ]
12 }}
```

O código da Listagem 3.11 mostra as estruturas internas a uma mídia chamada *video*. Ela possui dois tipos de âncoras: de propriedade (*property*), que podem alterar como a mídia é exibida e de conteúdo (*area*), que permite definir um trecho da mídia para ativar eventos específicos. No exemplo, a mídia diminui de tamanho após os 5 primeiros segundos em exibição, e volta ao tamanho normal após 25 segundos, marcação esta, feita pelo elemento *area*. Neste exemplo, o "video" é declarado sem uma região. As propriedades *visible* e *bounds*, se encarregam de definir a posição e visibilidade do mesmo. Os elos servem para mudar o tamanho do vídeo quando a âncora de conteúdo inicia ou termina. A ação *set* se encarrega de mudar o valor das propriedades responsáveis pelo tamanho da região em que o video é exibido. O objetivo deste exemplo é mostrar como se pode mudar uma propriedade de um elemento através de um *conector* (a ação *set* e o parâmetro *value* se encarregam disto). E também, como marcar um trecho de uma mídia através de uma âncora.

Listagem 3.12: Exemplo 3 - Switches

```
1  {ncl:{
2          head:[
3                  {region:{id:"rgSelecao1",height:"10%",width:"10%",top:"50%",left:"15%",
                       zIndex:2}},
4                  {region:{id:"rgSelecao2",height:"10%",width:"10%",top:"50%",left:"35%",
                       zIndex:2}},
5                  {region:{id:"rgSelecao3",height:"10%",width:"10%",top:"50%",left:"55%",
                       zIndex:2}},
6                  {region:{id:"rgSelecao4",height:"10%",width:"10%",top:"50%",left:"75%",
                       zIndex:2}},
7                  {region:{id:"rgFull",height:"100%",width:"100%", zIndex:1}},
8                  {descriptor:{id:"dSelecao1",region:"rgSelecao1", focusIndex:1, moveUp:2,
                       moveDown:4, moveLeft:4, moveRight:2}},
9                  {descriptor:{id:"dSelecao2",region:"rgSelecao2", focusIndex:2, moveUp:3,
                       moveDown:1, moveLeft:1, moveRight:3}},
10                 {descriptor:{id:"dSelecao3",region:"rgSelecao3", focusIndex:3, moveUp:4,
                       moveDown:2, moveLeft:2, moveRight:4}},
11                 {descriptor:{id:"dSelecao4",region:"rgSelecao4", focusIndex:4, moveUp:1,
                       moveDown:3, moveLeft:3, moveRight:1}},
12                 {connector:{id:"onSelectionSetStart", params:["var"],
                       expression:"onSelection then set with value='$var';stop;start"}}
13         ],
14         body:[
```

```
15                  {port:{"porta1":"selecao1", "porta2":"selecao2", "porta3":"selecao3",
                        "porta4":"selecao4"}},
16                  {media:{id:"variaveis", type:"application/x-ginga-settings",
                        anchors:[{property:{"selecao":null}}]}},
17                  {media:{id:"fundo",src:"fundo.png", region:"rgFull"}},
18                  {media:{id:"selecao1",src:"conector.png", descriptor:"dSelecao1"}},
19                  {media:{id:"selecao2",src:"contexto.png", descriptor:"dSelecao2"}},
20                  {media:{id:"selecao3",src:"descritor.png", descriptor:"dSelecao3"}},
21                  {media:{id:"selecao4",src:"regiao.png", descriptor:"dSelecao4"}},
22                  {switch:{
23                          id:"videos",
24                          "selecao==1":{media:{id:"video1", src:"videoConector.mp4",
                                region:"rgFull"}},
25                          "selecao==2":{media:{id:"video2", src:"videoContexto.mp4",
                                region:"rgFull"}},
26                          "selecao==3":{media:{id:"video3", src:"videoDescritor.mp4",
                                region:"rgFull"}},
27                          "selecao==4":{media:{id:"video4", src:"videoRegiao.mp4",
                                region:"rgFull"}},
28                          default:"video1"
29                  }},
30                  {link:{connector:"onSelectionSetStart", params:{"var":"1"},
                        binds:[{"onSelection":"selecao1"},
                        {"set":"variaveis.selecao"},{"stop":"fundo,videos"},
                        {"start":"videos"}]}},
31                  {link:{connector:"onSelectionSetStart", params:{"var":"2"},
                        binds:[{"onSelection":"selecao2"},
                        {"set":"variaveis.selecao"},{"stop":"fundo,videos"},
                        {"start":"videos"}]}},
32                  {link:{connector:"onSelectionSetStart", params:{"var":"3"},
                        binds:[{"onSelection":"selecao3"},
                        {"set":"variaveis.selecao"},{"stop":"fundo,videos"},
                        {"start":"videos"}]}},
33                  {link:{connector:"onSelectionSetStart", params:{"var":"4"},
                        binds:[{"onSelection":"selecao4"},
                        {"set":"variaveis.selecao"},{"stop":"fundo,videos"},
                        {"start":"videos"}]}}
34          ]
35  }}
```

O código da Listagem 3.12 trabalha com uma nova estrutura do *switch*. Neste código, é descrito um programa onde se tem quatro opções de vídeo que são mostradas pelas quatro imagens "selecao". Ao selecionar uma imagem, a variável que descreve as regras do *switch* (a propriedade "selecao" da mídia "variaveis" que é do tipo "application/x-ginga-settings") muda para o valor correspondente a imagem selecionada e o *switch* "video" é ativado. Desta forma, a regra correspondente ao vídeo pretendido é ativada e o mesmo é iniciado.

3.2.2 JNS e NCL

Como comentado, o JNS proporciona algumas mudanças em relação ao NCL. Nesta seção, serão mostradas algumas comparações feitas com a linguagem NCL. Para medir o quanto é possível economizar linhas de código com o JNS, foram feitas conversões dos exemplos contidos no livro "Programando em NCL 3.0" [33]. Os onze primeiros exemplos do livro mostram uma mesma aplicação que aos poucos vai recebendo mais componentes, e por fim, usa todas as funcionalidades do NCL. A conversão foi feita de duas formas: a primeira só se converteu o código de NCL para JNS mantendo-se as funcionalidades de reúso de NCL. Na segunda conversão, foi retirada a funcionalidade de reúso, a fim de se usar menos estruturas e linhas em JNS. Dois parâmetros foram medidos: o número de estruturas e o número de linhas, os resultados podem ser vistos na Figura 3.3. O eixo x representa cada programa utilizado na comparação.

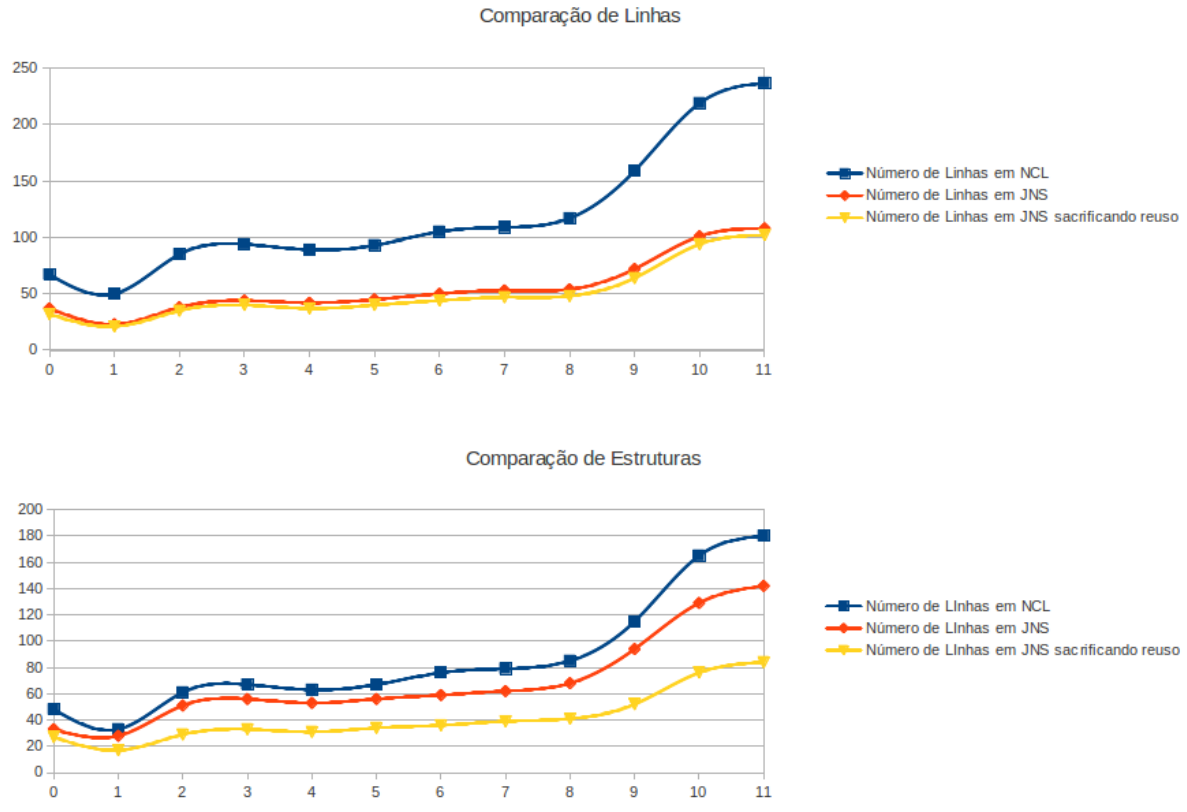

Figura 3.3: Comparação JNS e NCL

Na Figura 3.3, é possível ver que o código JNS tem 51.6% do número de linhas do código em NCL em édia, mantendo as estruturas de reúso, e 46.4% do número linhas do código NCL, retirando as estruturas de reúso. Quando se compara o número de estruturas (elementos de linguagem), o documento JNS tem 87.6% da quantidade de estruturas de NCL, mantendo o reúso, e 53.4%, sacrificando o reúso. Com base nestes resultados,

observa-se que o JNS permite especificar mais elementos de linguagem usando menos linhas.

Porém, isso não significa que o JNS seja mais simples para o autor, pois ele pode ocupar menos espaço mas ser de difícil leitura para o usuário. Foram feitos testes de usabilidade, a fim de avaliar a linguagem JNS. Tais testes consistiram em um roteiro contendo um mini tutorial de como usar a linguagem através de exemplos e exercícios e por fim um questionário para saber a opinião do usuário. Para validar os exercícios, foi usado um compilador da linguagem JNS para o NCL. Os testes seguiram os critérios de Nielsen [4], sendo realizados por dezoito usuários no total. O primeiro grupo, composto de 3 usuários, revelou problemas no roteiro que usava somente quatro exemplos e três exercícios, resultando em um segundo teste maior e mais didático com oito exemplos e quatro exercícios. No segundo teste, realizado com 3 usuários, os resultados foram um pouco melhores, no entanto, foi concluído que o teste ainda precisava ser melhorado para que usuários sem conhecimento de NCL tenham mais facilidade de fazê-lo. O último teste foi feito com oito usuários, três com conhecimento em NCL e cinco sem. Este teste consistia em um tutorial simples com três exercícios: dois bem simples e um último complexo que definia uma aplicação multimídia que era composta por em um vídeo interativo onde o usuário seleciona o idioma. No último teste, o tempo foi reduzido de duas horas para o máximo de 40 minutos. Os usuários sem conhecimento de NCL terminaram a tarefa em um tempo similar aos do que tinham o conhecimento de NCL. Todos os usuários que testaram a linguagem classificaram-na como útil ao desenvolvimento de NCL. A nota média dada a linguagem JNS foi 7.01 (variando numa escala de 0 a 10), considerando todos os testes. Além disso, 60% dos usuários do último teste lembraram as estruturas básicas para concluir uma aplicação após o teste.

Por fim foi realizado um outro teste com uma turma de sete alunos da disciplina de Fundamentos de Sistemas Multimídia, incluindo alunos de graduação e mestrado em Computação e Engenharia de Telecomunicações na Universidade Federal Fluminense. Inicialmente foi colocado um trabalho em os quatro alunos de graduação usariam o JNS para resolver um exercício e os três alunos de mestrado usariam o NCL. Em seguida, foi realizado um novo trabalho invertendo o uso das linguagens em cada grupo. O grupo que começou usando JNS tendeu a gostar mais de JNS, enquanto o outro preferiu usar NCL. O maior problema destacado pela turma foi a pouca documentação do JNS, que conta somente com um tutorial e uma página web de documentação, enquanto o NCL conta com um livro completo explicando as funcionalidades do mesmo, além de outros tutoriais e documentos disponíveis na web. O grupo que começou usando JNS mostrou-se capaz

de utilizar melhor as novas funcionalidades da linguagem do que o grupo que começou usando NCL. Ao fim do desenvolvimento das duas aplicações, a turma respondeu um questionário avaliando alguns aspectos de cognição das linguagens JNS e NCL.

Foram avaliados 9 aspectos de cognição[?]:Visibilidade, expressividade dos papeis, proximidade do mapeamento, verbosidade, comprometimento prematuro, dependências ocultas, propensão a erros, consistência e viscosidade. A figura 3.4 mostra a pontuação dada pelos conforme estes parâmetros, algumas perguntas perguntam o quanto de uma característica negativa (nos casos de verbosidade, comprometimento prematuro, dependências ocultas, propensão a erros e viscosidade) está presente na linguagem nestes casos a nota foi invertida para facilitar a compreensão da figura. Afim de ilustrar melhor a diferença entre a preferência de cada grupo o gráfico traz em separado as notas do grupo que começou com JNS e o que começou com NCL. Por fim foi perguntada a nota final para a linguagem JNS, o que na media foi maior mas para o grupo que começou usando NCL foi menor. Analisando com cautela as respostas dos usuários percebeu-se que somente dois realmente consideraram a linguagem JNS realmente pior que a NCL, tais usuários tiveram problemas na leitura da documentação e no teste do documento final.

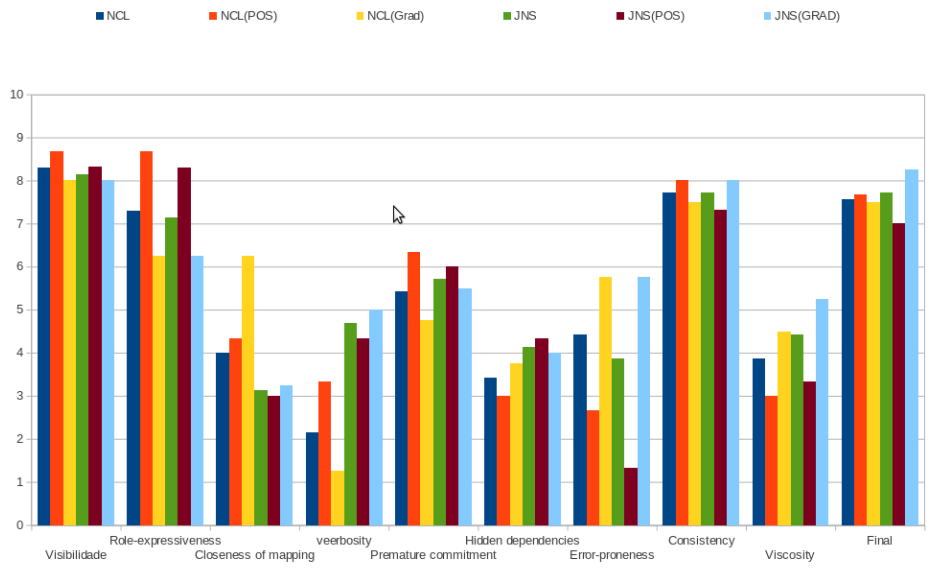

Figura 3.4: Resultado do teste de parâmetros de cognição

Os usuários avaliaram que o JNS é 25,6% menos verboso que o NCL e tem menos 5,7% de viscosidade no geral avaliaram ela em 1,4% melhor que o NCL. Os usuários que começaram usando o JNS o avaliaram melhor que os usuários que começaram usando NCL.

3.2.3 Ferramentas de Apoio à Autoria com JNS

Para permitir o uso da linguagem JNS para a criação de programas NCL, foi desenvolvido um compilador JNS, que traduz programas JNS em programas NCL, e uma extensão para editor de textos gedit [28]. O compilador JNS foi desenvolvido em Java com o auxílio da biblioteca aNa - API NCL para Autoria [32]. O compilador é estruturado em classes, que realizam a tradução de cada parte específica do documento. O conjunto de classes do compilador pode ser visto na Figura 3.5.

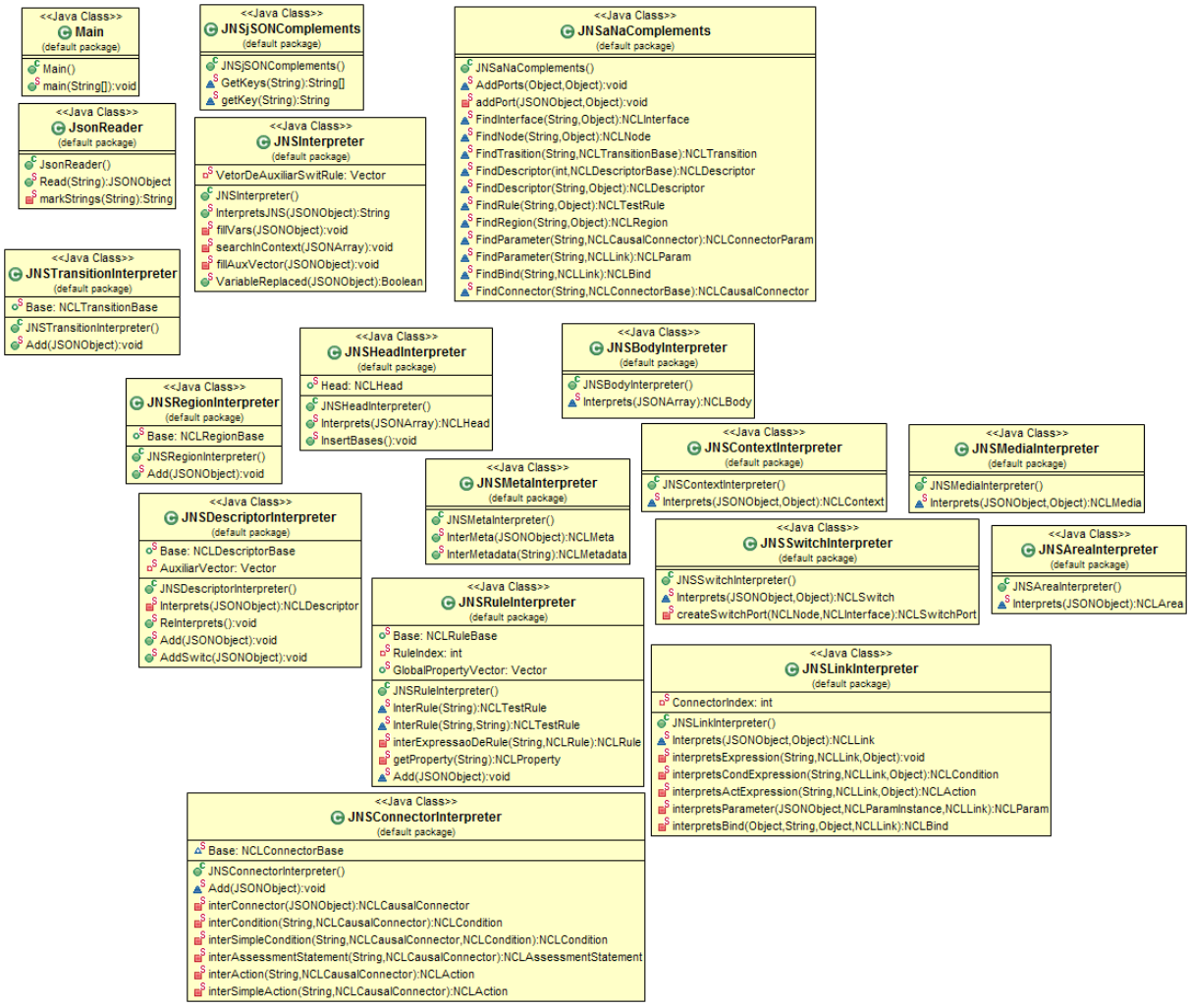

Figura 3.5: Classes do compilador JNS

O primeiro passo feito pelo compilador é extrair *id* e título do documento JNS. Caso não tenha *id*, o compilador automaticamente cria o *id* "jnsNCL". Em seguida, ele prepara um vetor com as variáveis temporárias usadas pelas regras, seguidas por suas variáveis reais que serão usadas. Depois, é passado este vetor ao interpretador de regras e o cabeçalho é interpretado pela classe responsável pela interpretação do mesmo. Então é feita uma varredura nos elementos internos ao vetor do cabeçalho, observando primeira-

mente se ele contém um elemento *include*, depois se é uma transição, descritor, região, *meta, metadata*, regra ou conector.

Caso seja a inclusão de um cabeçalho JNS, ele é guardado em vetor para futuramente ser adicionado ao cabeçalho, através da chamada recursiva da função de interpretação de cabeçalho, com um parâmetro indicando que ela foi chamada por um *include*. Se for uma transição, ela é simplesmente interpretada e adicionada à base de transições. Se for um descritor, ele é interpretado e adicionado a um vetor e à base de descritores. O vetor é lido posteriormente, para verificar se todos os atributos de navegação por teclas foram preenchidos corretamente, vendo se cada descritor indicado existe na base de descritores. Os *descriptorSwitches* são interpretados de forma semelhante a dos *switches*, podendo criar novas regras ou mesmo usar uma regra já criada. Se uma regra que faz uso de uma variável temporária, a regra irá ter o id igual ao *id* do *descriptorSwitch*, em conjunto com o *id* da regra, para assim, marcar a regra que usa a variável contida no *switch*. Caso seja uma região, ela é interpretada pela classe que interpreta regiões, esta classe checa se existe o atributo *device*. Se existir, ela coloca a região na base de regiões correspondentes ao *device*. Se não existir tal base, a classe a cria e coloca a região nela. Se o atributo *device* estiver em branco, a região é colocada em uma base de regiões com o atributo *device* vazio também. Na hipótese de alguma região ser filha de outra, uma busca é feita em todas as bases de regiões pela região pai. Caso não seja encontrada, a mesma é colocada em um vetor que será posteriormente lido para assim adicionar as regiões que não tinham a região pai presente no documento no momento em que foram interpretadas (isso pode ser evitado dependendo da maneira que o documento é feito, porém, é feita esta checagem para evitar que isso seja um pré-requisito). Se for um elemento meta ou *metada*, ele é interpretado e colocado no cabeçalho. Se for uma regra, a classe que a interpreta verifica se a regra usa uma variável temporária ou real (o que foi verificado no início da interpretação). Se for variável temporária, uma regra é criada para cada variável real que a usa. Se for variável real, a regra correspondente em NCL simplesmente é adicionada. Caso seja um conector, ele é interpretado e adicionado na base de conectores.

Após isso, é feita a interpretação do corpo do documento. A classe que interpreta o corpo utiliza a classe que interpreta o cabeçalho, pois algumas partes do cabeçalho (regiões, regras, elos) podem ser criadas no corpo, como foi comentado anteriormente.

Assim como no cabeçalho, uma varredura é feita pelos elementos contidos no vetor do corpo, verificando primeiro se é o *id* do corpo, uma mídia, um contexto, meta, *metadata*, uma propriedade, um elo, um *switch* ou uma porta. Uma mídia é interpretada, e se

ela contiver uma região, a mesma é incluída na base de regiões da mesma forma que seria se estivesse no cabeçalho do documento. Um contexto é interpretado por uma função parecida com a que interpreta o corpo, no entanto, ela tem um atributo extra que indica qual o contexto em que ele está contido. Essa função é chamada recursivamente quando um contexto contém outro contexto. Os elementos meta, *metada* e *property* são interpretados e adicionados no corpo. O elo é interpretado de forma a usar o conector a qual refere ou a criar o conector caso ele o declare explicitamente por uma expressão. O *switch*, de forma semelhante, usa a regra a qual se refere. Caso ele tenha usado uma regra com variável temporária, o id da regra será aglutinado ao *id* do *switch* que o usa, de modo semelhante ao *descriptorSwitch*. Uma porta especifica um componente e um de seus pontos de interface com base na notação "id.interface". Após interpretada , a porta é inserida no corpo. Ao fim, é observado se existe algum elemento nas bases de conectores, descritores, regras e regiões. Se existirem elementos filhos, as bases são adicionadas ao documento. Senão o documento fica sem a base, pois é inútil pôr no documento uma base vazia.

O compilador tem uma interface gráfica para facilitar o trabalho do usuário. Ao selecionar o arquivo JNS, o compilador cria um arquivo com o mesmo nome e terminação NCL. Uma imagem da interface do compilador pode ser vista na Figura 3.6.

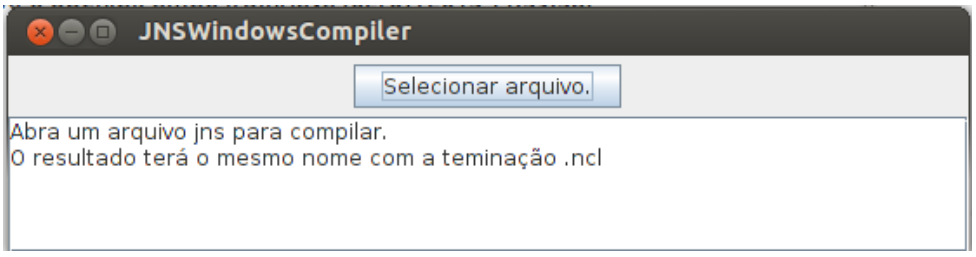

Figura 3.6: Compilador JNS

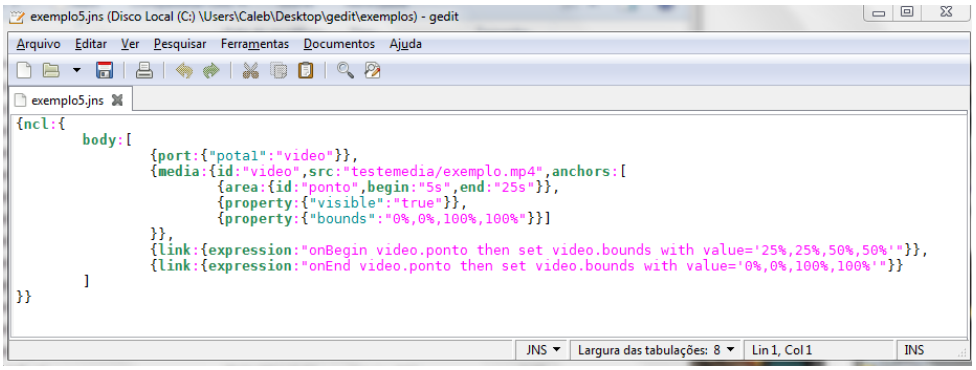

Figura 3.7: Extensão do gedit para JNS

Foi criado também uma extensão para o editor de texto gedit, que pode ser vista na

Figura 3.7. O editor de texto basicamente marca o código em JNS e alerta os erros de sintaxe marcando-os em vermelho, tornando mais fácil a edição em JNS. A razão para sua criação é encontrada na primeira bateria de testes de usabilidade, pois todos os usuários tiveram dificuldade com a sintaxe da linguagem. Nessa bateria 2/3 dos usuários que tinham conhecimento de NCL consideraram a linguagem JNS útil e mais simples de usar.

Foi criado um site contendo a documentação completa da linguagem JNS[1] e alguns exemplos de uso. A documentação de refêrencia contém também informações que podem ser usadas por desenvolvedores NCL, como valores possíveis de alguns atributos NCL e seus significados.

Este capítulo apresentou a linguagem JNS, uma das propostas desta dissertação. O próximo capítulo apresenta a ferramenta NCL4WEB, outra contribuição importante deste trabalho.

[1] http://www.midiacom.uff.br/čaleb/documentacaoJNS

Capítulo 4

NCL4WEB

NCL4WEB é uma ferramenta que tem como objetivo traduzir um código em NCL para HTML, utilizando ferramentas presentes nos padrões da W3C[1], para isso, a ferramenta utiliza algumas funcionalidades existentes no HTML5[47], a nova versão do HTML. O HTML5 permite a exibição de áudio e de vídeo através das tags específicas e executa eventos no código JavaScript, de acordo com o estado da mídia. Além do HTML5, foi utilizada a folha de estilo XSLT[46] para realizar de modo orgânico a transformação do NCL, de forma que o navegador abra o arquivo NCL e o mesmo seja automaticamente transformado em HTML5. O XSLT é um arquivo usado para modificar a maneira que um XML é apresentado, em geral, ele é usado para dar forma a arquivos de dados para que os mesmos sejam exibidos de forma mais amigável ao usuário. Portanto, como o NCL é um XML, ele pode ser modificado por um XSLT. Os detalhes de como é feita essa transformação, serão apontados nas seções 4.3 e 4.4.

4.1 Decisões de projeto

Para decidir usar a abordagem híbrida XSLT+JavaScript, ao invés de se utilizar somente o JavaScript. Pois, foi considerada a facilidade de se usar o trabalho, visto que ao se utilizar somente JavaScript seria necessário uma página HTML para chamar o código NCL. Por isso, foi criado um código XSLT para realizar a transformação. mas também foi criada um biblioteca complementar em JavaScript para controlar os eventos de sincronismo e interação com o usuário.

A razão de não se implementar tudo no código XSLT foi separar a transformação do código NCL em HTML do controle de eventos do NCL, tornando a sua implementação

[1] World Wide Web Consortium - www.w3.org

mais robusta. Deste modo, o JavaScript cuida dos eventos disparados pelo código HTML, gerado pelo XSLT. Isso permite algumas ações que não seriam possíveis sem o JavaScript. Por exemplo, a interpretação de propriedades é viável dentro do próprio XSLT, porém, muitas das ações que as propriedades iriam possuir, só podem ser realizadas durante a execução (volume de som, tamanho dependente do tamanho da tela e outras coisas). Diante disso, a solução híbrida admite aproveitar-se do melhor de ambos: o uso do XSLT, permitindo uma transformação orgânica do NCL e o uso do JavaScrpit, possibilitando um controle completo do todos os objetos dentro do documento e a programação mais fácil de algumas ferramentas.

Não foram desenvolvidas todas as funcionalidades do NCL. Por questões de projeto, não foram implementados: as CoumpoundAssessmentStatement, elos do tipo get e set, importação de documentos, o módulo device, âncoras (com exeção das âncoras temporáis, que foram implementadas), as transições que não sejam fadeIn e fadeOut, implementação de papéis personalizados para ações nos elos, algumas propriedades do descriptor e da mídia(*balanceLevel, bassLevel, trebleLevelfit, player, playerLife, reusePlayer, style, focusBorderWidth, focusBorderColor*), alguns atributos do descritor(*freeze,player*) e a utilização de scripts Lua. Todas essas funcionalidades podem ser implementadas, entretanto, requer um trabalho que não foi desenvolvido a contento em tempo hábil. Os scripts em Lua necessitam de um trabalho muito alto para serem implementados, pois requerem a completa tradução de um script Lua para JavaScript tal trabalho já tem sido feito pelo grupo que criou o WebNCL[26], assim com algumas das funcionalidades aqui citadas, já estão sendo implementadas no mesmo. Uma limitação que requer um estudo maior é a importação de páginas web que usam JavaScript, pois páginas importadas via *iframe*, entram em conflito de JavaScript com a página que o importou, isso serve para documentos NCL importados dentro do documento também. Para facilitar a manipulação dos elementos JavaScript foi usada a biblioteca JQuery[30], que permite o uso de funções XPath[3] para filtrar elementos do HTML e a manipulação dos mesmo com alguma funções de varredura.

Apesar das limitações, todas as aplicações do clubeNCL[1] que usam somente NCL foram traduzidas para HTML5 de modo satisfatório, com exceção de algumas transições e do último exemplo do primeiro joão que contém um documento NCL importado.

4.2 Trabalhando com XSLT

O funcionamento básico de um XSLT é ler o XML de forma recursiva, ou seja ao se deparar com uma tag XML, ele a compara às regras dos *templates* declarados dentro do XSLT. Assim, quando é encontrado um elemento que satisfaz uma regra do *template*, o elemento é interpretado pelas instruções contidas no *template* correspondente. Um *template* pode ordenar a reexecução da folha de estilo para os elementos internos ao elemento que ele interpretou com o comando *applyTemplates*. É possível criar *templates* sem regras, que são chamados por outros *templates* em momentos que forem convenientes, através da instrução *callTemplate*, o que possibilita o uso de funções recursivas. O XSLT contém funções complexas que permitem: navegar pelo documento através de uma expressão XPATH[3]; ler documentos XML externos ao documento. Nesta perspectiva, o XSLT do NCL4WEB não é apenas uma folha de estilo que lê e interpreta um documento NCL e vai descendo do elemento pai até o elemento filho, mas sim uma folha de estilo capaz de ler documentos importados, e também conseguir mudar a estrutura do documento NCL para adequá-lo ao HTML5. Expressões XPATH permitem a navegação pelo documento XML através da linguagem XPATH o conhecimento das mesmas é fundamental para trabalhar com transformações em XML. O XSLT é uma linguagem declarativa, ou seja, utilizá-lo para algo complexo não é uma tarefa trivial. O XSLT possui certas limitações que tornam difíceis algumas tarefas, por exemplo, embora seja possível criar uma variável, esta terá o valor sempre constante, sendo impossível trocar o seu valor somente recriar ela. Neste caso, outra variável será criada no contexto onde ocorreu a troca e quando sair do contexto o valor será perdido. Desta forma, não é possível fazer um contador do modo clássico (através de uma variável). Para se implementar um contador é necessário usar uma função recursiva e o contador funcionará exclusivamente no enlace recursivo. Sobrepujando essas limitações, o ncl4web.xsl foi criado contendo mais de 1625 linhas. Era possível quebrar o mesmo em vários arquivos de forma semelhante a um programa orientado a objeto, no entanto, isso iria requerer que o usuário do NCL4WEB baixasse todos os arquivos o que adicionaria complexidade a ferramenta.

4.3 Descrevendo a folha de estilo ncl4w.xslt

A criação do HTML e de toda a tradução gira em torno de um XSLT que tem 6 *templates* de regras de seleção (*NCL, head, body, context, switch* e *port*) e 7 *templates* principais, que servem para traduzir estruturas completas também, mas que são chamados pelos *tem-*

plates de seleção (*region, rule, descriptor, descriptorSwitch, link, mídia* e *property*), além de mais alguns *templates* secundários usados para ajudar na construção das estruturas interpretadas pelos principais. O fluxograma da figura 4.1 ilustra o funcionamento da folha de estilo.

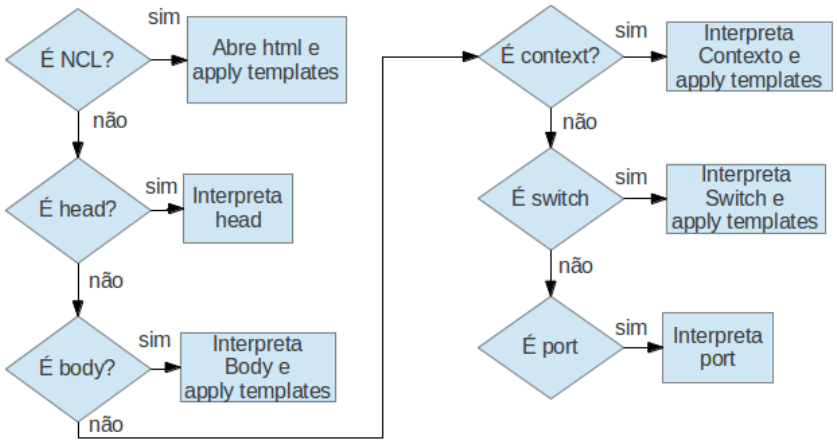

Figura 4.1: Diagrama da Folha de estilo ncl4web.xsl

Como pode se perceber, a folha faz uma varredura pelo documento ncl, começando pela *tag ncl*, após isso, o *template* é reexecutado de forma recursiva para o interior da tag NCL. De forma que o cabeçalho (*head*) e o corpo (*body*) são interpretados pelo *template*. Os elementos dentro do corpo também são processados pelo *template* recursivamente, para que o mesmo interprete os elementos: *port, context* e *switch*. O *context* e o *switch* também têm os elementos em seu interior processados pelo *template*, para que assim quaisquer contextos, *switchs* ou portas internas a eles (no caso somente contextos contém portas) sejam interpretados pelo XSLT. Os demais elementos do NCL serão interpretados pelos *templates* específicos, que serão executados pelo *template* do *Body, Context* ou *Switch*, como será explicado adiante.

4.4 Tradução do NCL para HTML5

Será explicado agora como é feita a transformação do NCL para HTML5. As mídias de NCL (vídeo, áudio, texto, páginas web e etc.) podem ser visualizadas em HTML com a ajuda de uma *tag* chamada *embed*. Entretanto, o *embed* não contém eventos e métodos necessários para controle de mídias contínuas (audio e vídeo), dificultando assim, o uso do mesmo como o correpondente a mídia NCL. Dessa forma, foi usado um *tag* diferente para cada tipo de mídia. A tabela 4.1 faz a relação entre os elementos NCL e elementos HTML correspondentes: Observa-se que o cabeçalho do documento NCL não foi transportado

Tabela 4.1: Correspondência entre elementos NCL e sua forma em HTML5 com o NCL4WEV

Tag NCL	**Correspondente no HTML**
media/audio	Áudio
media/video	Vídeo
media/settings	span(classe settings)
media/imagem	Img
media/texto ou html	Iframe
media - referida com instsame	span (classe instsame)
Context	div (classe atributo no nós filhos)
Region	div pai dos nós que a usam
descriptor	Div
link+conector	Funções e eventos JavaScript
Switch	ul e li invisível
Rule	Função JavaScript que retorna valor booleano
descriptorSwitch	Função JavaScript ligada as variáveis, disparada por qualquer troca nas mesmas.
Port	input/hidden
property	tag property
area	tag area

para um cabeçalho no documento HTML. A razão disso é que embora a região que um elemento no HTML ocupa possa ser especificada por uma classe presente no cabeçalho, as classes não podem ser aninhadas da mesma maneira que NCL aninha as regiões uma dentro da outra. A herança de classes em HTML funciona sobrescrevendo os atributos e não os somando logo em alguns casos não seria possível ter o funcionamento orgânico da tradução.

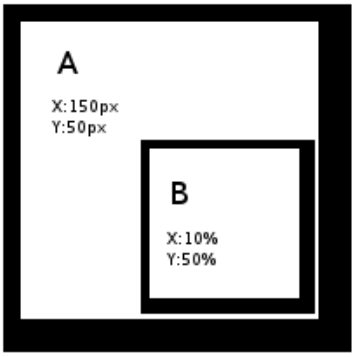

Figura 4.2: Exemplo do problema que impede o uso de classes em regiões

Por exemplo, tem-se uma região B dentro, esta se encontra aninhada dentro da região

A. Em linguagem NCL, a região B terá o posicionamento da A, somado ao seu posicionamento para calcular sua região na tela, além disso, se seu tamanho for descrito em porcentagem, esta será em relação à região A (que a contém) e não em relação ao tamanho da tela. Se fossem usadas classes a posição da região B, seria calculada com base no tamanho da tela e não do tamanho da região A. Portanto, tipo de relação só não pode ser expresso na classe da folha de estilo CSS, pois seria necessário conhecer o tamanho da tela que irá ser exibido para calcular algumas distâncias. Este problema é ilustrado na figura 4.2.

A solução adotada foi utilizar o cálculo de aninhamento de elementos já realizado pelo HTML, desta forma, cada região se tornou uma *div* e todos as regiões filhas se tornaram *divs* internas a *div* da região pai os atributos da região e uma classe atribuída a todos os elementos que estão abaixo da região, denominada *innerRegion*, que descreve quais atributos devem ser herdados. Logo, dentro da região existe um descritor que contém uma mídia, ambos descritor e mídia contém a classe *innerRegion* e irão herdar os atributos da região que estão contidos. Observa-se que algumas mídia bem como os switchs não são transformados em divs mas em outras estruturas como span e uma ul invisivel isso é feito para faciliatar a leitura por javascript. O switch é a unica esturtura feita com uma listagem(ul) dessa forma é lido mais rapidademente pelo JavaScript assim como as mídias transformadas em span.

Como decisão de projeto, decidiu-se optar por tornar a transformação do documento NCL a mais orgânica possível. Ou seja, fazer com que cada elemento que antes era interpretada pelo processador NCL seja interpretado pelo processador do HTML5, mantendo quando possível as facilidades de reuso do NCL. Por isso, optou-se por transformar os descritores em *divs*, que iriam estar aninhadas em regiões. As mídias contidas nos descritores irão usar os mesmos para navegação e irão herdar as propriedades internas aos mesmos quando iniciadas. O JavaScript se encarrega de ler as propriedades do descritor, realizar as operações de navegação e de incluir nas mídias as propriedades herdadas.

O fluxograma da figura 4.3 detalha em linhas gerais o processo de interpretação de um arquivo NCL, as linhas indicam o caminho seqüencial que a transformação pode seguir. Inicialmente, é criado um elemento principal do html a tag html, em seguida, é criado o cabeçalho do documento, onde serão colocadas as referências para os códigos JavaScript auxiliares e as funções, correspondentes as regras, serão criadas. Dentro do elemento html ainda é criado o elemento *body*, onde serão colocados: contextos *switchs*, regiões, descritores(que não tiverem regiões), mídias (que não tiverem descritores), propriedades,

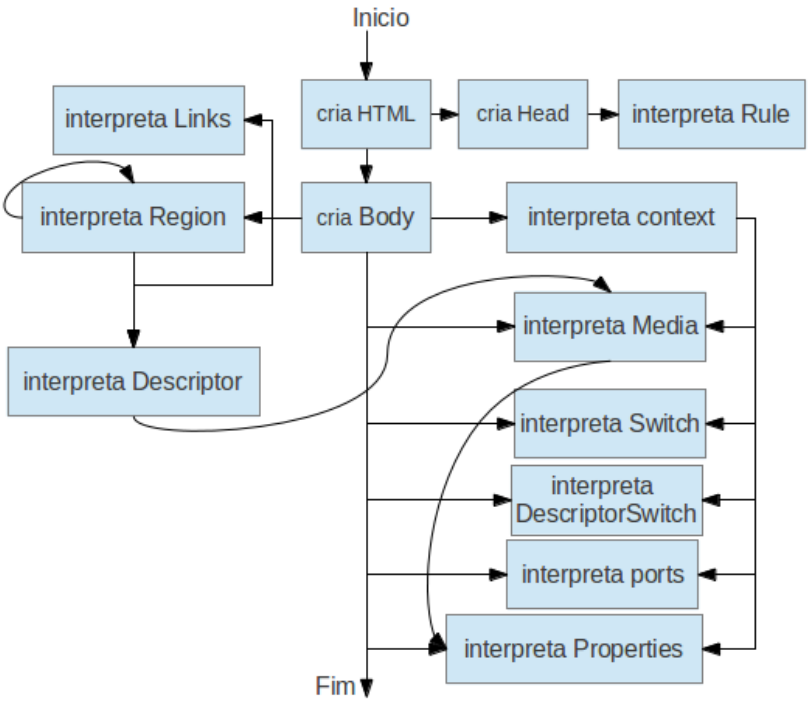

Figura 4.3: Fluxograma da tradução do NCL em HTML5

descriptorSwitchs e elos. Quando o corpo é criado coloca-se dentro do *body*, as *tags* correspondentes as regiões, descritores, contextos, portas, mídias e propriedade. Após isso, abre-se uma *tag script* e interpreta-se os elos e *descriptorSwitch* que são traduzidos em funções em JavaScript. O interpretador de contexto interpreta também todos os switchs, mídias, portas e propriedades internos ao contexto interpretado. O interpretador de regiões interpreta todos os descritores internos a região e se executa recursivamente para interpretar as regiões internas a região interpretada. O interpretador de descritores executa o interpretador de mídia para as mídias internas ao descritor interpretado. Cada uma das etapas destas transformações serão discutidas adiante.

4.4.1 Interpretando o cabeçalho(*head*)

O interpretador de cabeçalho não interpreta apenas as regras do cabeçalho NCL. Também se interpreta o atributo *title* do NCL e as transições. A razão de não se interpretar outros elementos é por causa da estrutura de aninhamento das regiões. Isto requer que as mesmas tenham que estar presentes dentro do documento. E os descritores possuem parâmetros que não podem ser traduzidos para elementos do CSS3. Além de interpretar estes elementos, o cabeçalho chama a biblioteca JavaScript (que complementa o NCL controlando o funcionamento do código HTML5 traduzido) e ainda cria algumas classes CSS3 padrão para os elementos que ativos, elementos inativos, elementos internos a uma

região e elementos que estão com o foco ativo. As transições são transformadas em objetos JavaScript que contêm todas as informações das mesmas, elas ficam contidas em um objeto chamado “transicoes", que é acessado pela biblioteca JavaScript quando for usar a mídia.

4.4.2 Interpretando o corpo(*body*)

O interpretador do corpo é um dos *templates* mais complexos. O fluxograma da figura 4.4 descreve o funcionamento do interpretador do *Body*, inicialmente a *tag body* do *html* é aberta. Em seguida, é colocado o *id* e é chamada a instrução que aplica faz os elementos internos do *body* serem lidos recursivamente. Após estes passos são interpretadas estruturas que requerem busca em contextos diferentes. Todas as regiões presentes no cabeçalho e em cabeçalhos importados são interpretadas e colocadas no documento, através da função que interpreta as regiões. A partir disso, é feita uma busca pelos descritores que não tem uma região vinculada e são traduzidos pela função que interpreta os descritores. A mesma busca é realizada nas bases de descritores importados, interpretando qualquer descrito importado, que não tenha uma região atrelada ao mesmo. Os *DescriptorSwitch* são in-

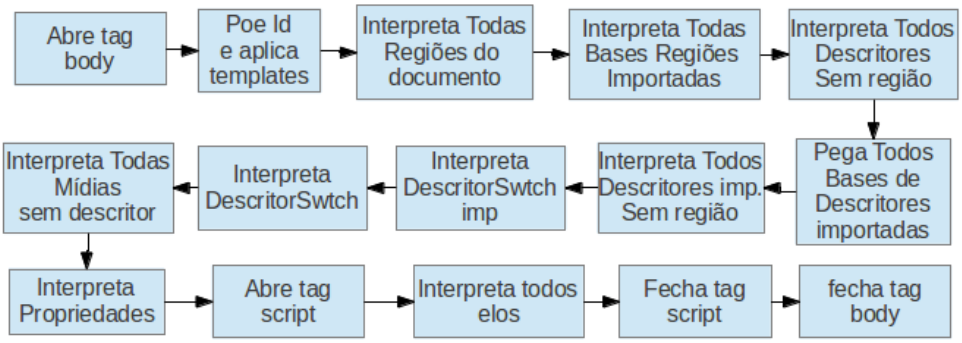

Figura 4.4: Fluxograma da interpretação do corpo do documento

terpretados por outra função semelhante a que interpreta os descritores, interpretando os presentes no documento e os que estão em bases de descritores importadas. Porém, eles não interpretam os descritores internos a eles, pois isso já foi feito anteriormente pelo interpretador de região ou pelo interpretador de descritores que não tem regiões. Desta forma, os *descriptorSwitch* se tornam uma função JavaScript que é ativada quando acontece uma mudança em uma variável global e troca o descritor da mídia de acordo com as regras contidas no *descriptorSwitch*. Após isso, são interpretadas as mídias internas ao corpo e as propriedades (as que estão internas a contextos e *switchs* são interpretadas dentro das funções que interpretam os mesmos). Em seguida, é aberta a *tag* de *script* e

são interpretados os elos, que se tornam funções atreladas a eventos dos elementos criados no corpo do documento. Por fim, são fechadas as *tags* do corpo e do *script*.

4.4.3 Interpretador de Regras

As regras irão se transformar em uma função JavaScript que irá pegar o valor da variável, a qual a função se relaciona e irá executar a comparação do valor da variável com o valor contido na regra, retornando positivo se for igual, e negativo, se for diferente. Segundo a norma a variável deve estar no único elemento media do tipo “application/x-ginga-settings"[38]) A função resultante será chamada com auxílio do objeto *window* do JavaScript, que pode chamar qualquer função global através do nome. Assim, é possível chamar a função através de uma variável. O fluxograma da Figura 4.5 mostra como acontece a transformação de uma regra em HTML. O *template* que interpreta as regras

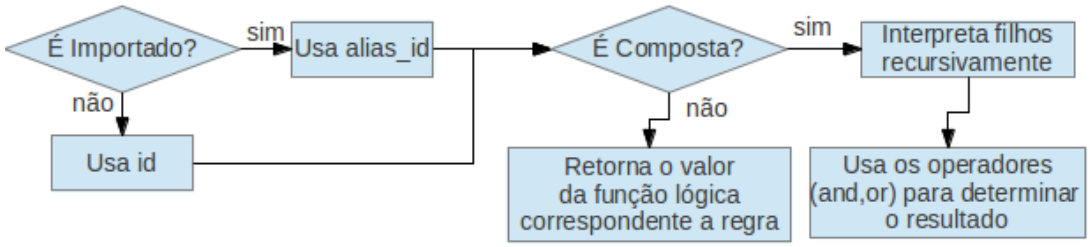

Figura 4.5: Fluxograma da tradução da regra

contém dois parâmetros: a regra, que contém o elemento correspondente a regra (o *rule*) e o *alias*, que contém o *alias* do documento importado, para o caso da regra que vem de uma base de regras que foi importada para o documento. Se o parâmetro *alias* estiver vazio, o *template* irá entender que a regra se encontra no mesmo documento, e irá criar uma função JavaScript com o nome do *id* da regra, que irá retornar valor booleano verdadeiro se a regra for verdadeira, e falso, se a regra for falsa. Se o parâmetro *alias* estiver preenchido o *template* irá entender que a regra vem de um documento externo e antes de criar a função irá mudar o seu nome para “alias_idDaRegra", diferentemente do documento NCL, onde o *id* e o *alias* são separados por uma tralha “#"em um JavaScript tal coisa não é possível, pois a tralha(#) é um caractere proibido a um nome de função. Se a regra for uma regra composta, ela ira pegar cada regra interna e chamar o mesmo *template* recursivamente para interpretar as regras compostas, e o *template* de regra simples, para interpretar as regras simples. Devolvendo como resposta um *or* (caso o atributo *operator* seja *or*) ou um *and* (caso atributo operador seja *and*) das regras internas interpretadas pelo *template*. Para organizar melhor o código, existe um *template* diferente

para interpretar regras compostas e regras simples.

4.4.4 Interpretador de Portas

As portas são traduzidas em elementos *inputs* do tipo *hidden* com a classe *port*, contendo os atributos *id, value* e *interface*. Onde *id* é o id da porta no NCL e o *value* é o atributo *component* em NCL. Caso exista uma interface, um atributo *interface* será adicionado à porta contendo o valor da interface da porta em NCL.

4.4.5 Interpretador de Propriedades

As propriedades são traduzidas em elementos *inputs* do tipo *hidden* com a classe *property*. Contendo os atributos *id* e *value*, onde *id* é o id da propiedade no NCL e o *value* é o atributo *value* em NCL.

4.4.6 Interpretador de Contexto

O contexto se torna também uma *div*, contendo a classe *context* e o atributo *context* que indica quem é o contexto que o contém (os contextos estão aninhados no HTML, mas este atributo agiliza o processo do JavaScript). Caso o contexto esteja dentro do corpo o atributo é preenchido com *body*. Cada porta, *switc* e contexto interno ao contexto são interpretados pela folha de estilo, pois o contexto usa a instrução *apply-templates* fazendo que estes elementos sejam interpretados pelo XSLT recursivamente. Além disso, são interpretadas as propriedades e qualquer mídia, que não tenha um descritor associado, internos ao contexto.

4.4.7 Interpretador de Switch

O *switch* se torna uma *tag* do tipo *ul*. Aninhando todos os *bindRules* em *tags* do tipo *li*, dentro destes elementos *li* estão o id da regra (elemento *rule*) atrelada e o id do elemento que deve ser ativado (elemento *constituent* da regra). O elemento *default* é um elemento *li* com o atributo *default*, contendo o *id* do elemento padrão do *swtich*. Os *switchPorts* são transformados em elementos *ul* internos a um elemento *li* vazio. Os *switchPorts* tem dentro de si o *id* e a *interface*, que devem ser ativados por ele (elementos *component* e *interface* do *switchPort*). O *ul* do *switchPort* carregam também o *id* da regra atrelada (mesma *rule* do *bindRule* do elemento).

4.4.8 Interpretador de Regiões

Uma região é interpretada através de um *template* genérico que serve tanto para regiões importadas como regiões internas ao documento. Dessa forma, contém os atributos: região, que contém a região em si; *alias*, que contém o *alias* de uma região importada; *regionURI* que descreve a URI de origem da região e por fim o *originalDoc* que contém o documento original que importou a região. Apenas o segundo atributo, só é usado quando a região é importada de um documento externo. O *template* inicialmente coloca o *id* e marca a região com a classe *region*, para facilitar a identificação da mesma no JavaScript. Os atributos internos de localização (*top, left, rigth* e *bottom*), assim como os atributos de tamanho (*heigth* e *width*) e o atributo *zIndex* são colocados dentro do atributo *style* da região. O *zIndex* caso não esteja presente é definido como automático e será tratado pelo JavaScript. Caso o atributo *alias* esteja presente, significa que o documento é importado, dessa forma todos os descritores que chamam a mesma pela combinação *alias#id* são encontrados e interpretados. Se a região for local, os descritores que a chamam são interpretados. Em seguida, é feita uma busca no documento por descritores importados, que chamem pela região que está sendo interpretada. Por fim, é realizada uma busca por regiões internas a que está sendo interpretada, e as mesmas são interpretadas pelo mesmo *template* recursivamente. O fluxograma da Figura 4.7 mostra as linhas gerais dessa transformação.

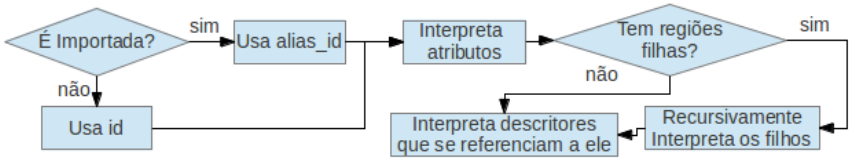

Figura 4.6: Fluxograma da tradução da região

4.4.9 Interpretador de Descritores

Um descritor também é interpretado através de um *template* híbrido capaz de interpretas descritores importados de outro documento e descritores internos ao documento. O *template* tem como parâmetros: o descritor, que contém o descritor em si; o *alias*, que contém o *alias*, caso ele seja importado; e o *originalDoc* que contém o documento original que importou o descritor. No caso de um descritor interno ao documento principal, apenas o segundo atributo não é preenchido. Após colocar o *id*, é testado se o descritor tem uma região, sendo positivo, ele ganha a classe que indica que ele está dentro de uma

região (para assim herdar os atributos da mesma), além da classe que indica que ele é um descritor. Em seguida, são colocados os atributos de navegação: *focusIndex, moveLeft, moveRigth, moveUp* e *moveDown*. Por fim, são colocados os parâmetros de descritores, que são interpretados como elementos *html input* do tipo oculto (*hidden*) e serão interpretadas pelo JavaScript quando uma mídia interna a ele for executada. O fluxograma da Figura 4.7 mostra em resumo essa transformação.

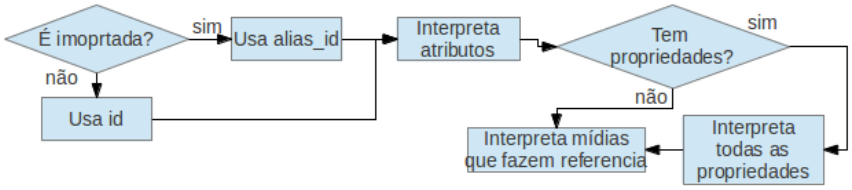

Figura 4.7: Fluxograma da tradução da região

4.4.10 Interpretador de Mídia

O *template* que interpreta a mídia tem dois parâmetros a mídia (que contém o objeto XML da mídia) e o *id* (para os casos de mídia referenciada). Caso a mídia seja uma imagem, esta é traduzida como um elemento *img* e suas âncoras são interpretadas fora do elemento (pois alguns interpretadores de XSLT não interpretam elementos internos a *tag img*) com um atributo *parent* para identificar o elemento que o contém. Se for uma página *web* ou um texto, a mídia é traduzida como um *iframe* e também tem suas âncoras externas ao elemento. Caso seja um vídeo, a mídia é traduzida pela *tag video* e tem suas âncoras internas ao elemento. Sendo um áudio é traduzida pela *tag audio* e tem suas âncoras internas ao elemento. Uma mídia do tipo “application/x-ginga-settings"se transforma em uma *tag* do tipo *span* com a classe *settings* que o identifica para o JavaScript como o nó de variáveis globais do Ginga-NCL. As âncoras são interpretadas pelos *templates interpretaPropriedades* (mesmo usado no corpo e contexto) e *interpretaArea*, através de um *template* genérico chamado *interpretaAtributosMedia*. E além de chamar estes *templantes*, interpreta os parâmetros internos comum a todas as mídias: *id*, classe (se é do tipo *settings*), *src* e o atributo *context* que assim como na *div* do contexto contém o contexto ao qual a mídia é filiada (caso seja o corpo ele tem o valor *body*). O fluxograma da Figura 4.8 mostra as linhas gerais da transformação da mídia em HTML.

As ancoras internas a mídia no momento são somente as temporais e estas embora façam uso da tag “area” é o javaScript quem controla a interação da ancora que no momento só suporta a ancora temporal.

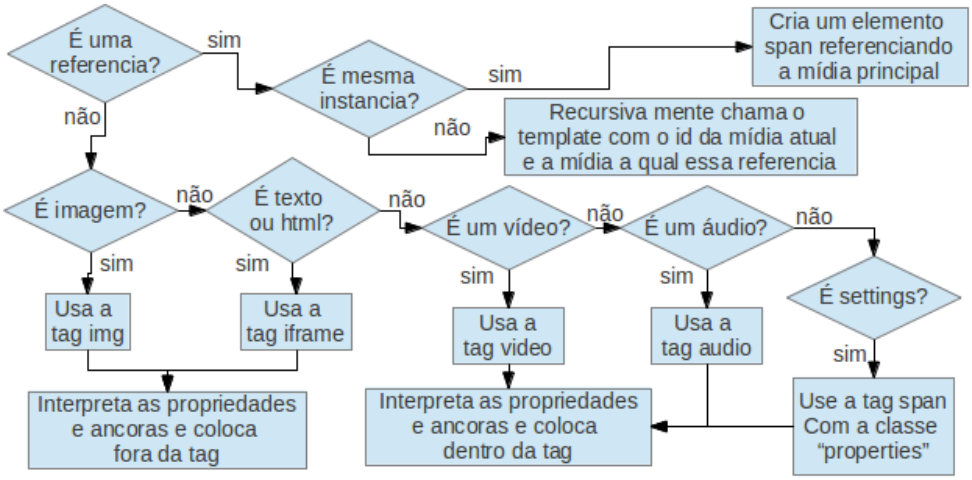

Figura 4.8: Fluxograma da tradução da mídia

Após uma mídia ser interpretada, é feita uma busca por mídias que a referenciam, e elas também são interpretadas. Pois estas devem estar dentro do mesmo contexto, e região mesma posição da tela que a mídia principal. Caso elas estejam em um contexto diferente da mídia, a função *interpretaPropriedades* se encarregará de preencher o atributo *context* com contexto correto.

As mídias que se referenciam a outras mídias podem ser de dois tipos: um é a mídia que tem uma instância diferente da mídia, a qual ela referencia (atributo *instance* igual a *new*), neste caso, ela será interpretada com o objeto da mídia que ela referencia, porém, com um *id* diferente. O outro caso é quando a mídia tem a mesma instância da mídia referenciada (o atributo *instance* igual a "instSame"), e a mídia é interpretada como uma *tag span* da classe *instsame*, que só serve para marcar que a instância da mídia está em outro nó.

4.4.11 Interpretador de *Switch* de Descritores

Tal qual o *template* que interpreta o descritor, o *template* que interpreta o *descriptorSwitch*, tem um *template* genérico para interpretar tanto *descriptorSwitchs* do documento como *descriptorSwitch* importados. O *template* abre uma *tag script* e cria-se uma função JavaScript com a identificação do *descriptorSwitch* no documento: caso seja interno ao documento principal, o *id*; caso seja importado, *alias_ id* (o "#"é substituído pelo mesmo motivo que na regra, a tralha(#) é um caractere proibido a funções em JavaScript). Para cada bindRule, o *template* chama um *template* para procurar a regra na base de regra, que é capaz de encontrar elementos em bases importadas (*encontraNaBase*), e após encontrar o nome da regra no documento, é criada uma instrução JavaScript que verifica se

a regra é verdadeira. Sendo verdadeira, a função coloca todas as mídias que referenciam este *descriptorSwitch* (inclusive mídias que referenciam essas mídias) no descritor cuja o *constituent* se refere (esta troca é feita por uma função da biblioteca JavaScript chamada *trocaDescritor*). O template *descriptorSwitchInterpreter* chama o template que interpreta os descriores para interpretar os descritores internos a ele. Então, são interpretados todas as mídias que referenciam o *descriptorSwitch* e é também ligado um evento de início do documento a uma execução da função do *descriptorSwtich*, afim de colocar as mídias que se referenciam a ele no descritor correspondente no estado inicial do documento. A função que troca o descritor é chamada pela mídia que usa o descritor no momento que a mesma é iniciada, para que ela então coloque a mídia no descritor correspondente.

4.4.12 Interpretador de Elos

Os elos se transformam em um conjunto de funções que representam a ação do elo e uma ligação feita ao evento em questão no(s) elemento(s) que ativa(m) o elo. É o *template* mais complexo em todo o XSLT do NCL4WEB. A figura 4.9 mostra esta interpretação, inicialmente, é necessário colocar um nome único na função que irá simbolizar as ações que o elo irá realizar, esse nome único é formado pelo *id* do elo (caso presente), ou por uma conjunção de todas as condições, ações que o elo uso e elementos que o elo liga, terminando pelo *id* do *conector*. Dessa forma, um elo que ao terminar o nó *media1*, inicia o nó *media2*, cujo conector se chama *conector1* iria ficar: *onEndmedia1startmedia2conector1*. Em seguida, é verificado se o conector usado no elo é importado, caso positivo, o conector é pego do arquivo externo e é interpretado.

A função com o nome criado (o *id* ou a tupla única) contém as ações descritas no elo dentro de si. Se for uma ação com mais de um alvo, pode se ter a execução paralela ou sequencial. Enquanto que no caso sequencial as funções de ação podem ser chamadas sequencialmente. No caso paralelo é necessário criar funções que executam as ações e elas são atreladas a um evento que quando disparado executa todas em paralelo. No caso de ações complexas, o procedimento é o mesmo. Se as ações forem sequenciais, elas simplesmente são interpretadas sequencialmente. No caso das ações serem paralelas, elas são colocadas em uma função e disparadas pelo mesmo evento. As condições são convertidas em eventos ligados aos elementos que são referenciados pelo elo. Sendo uma condição complexa, todas as condições “filhas"ativam uma função intermediária que tem um objeto auxiliar, que tem o nome formado pelo do nome da função principal acrescido de uma variável (para manter a unicidade do nome). Este objeto guarda quais condições

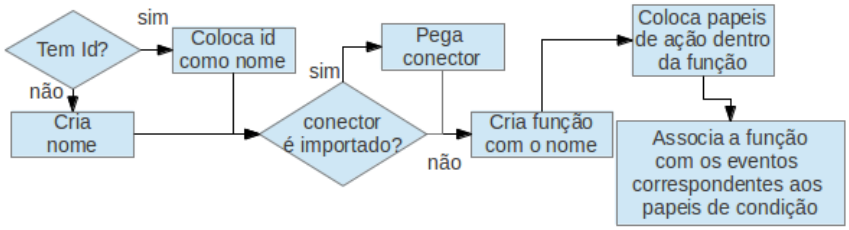

Figura 4.9: Fluxograma da interpretação dos elos

foram ativadas em um período de um segundo, e caso exista algum teste de atributo na condição, ele é feito dentro desta função intermediária. Caso a condição complexa seja satisfeita, a função da ação é executada. Na hipótese da condição ser uma tecla especial, uma função auxiliar também será criada para executar quando uma tecla for pressionada e ativar a função de ação correspondente ao elo. Antes de executar a função ação, será verificado se o elemento que disparou a ação está ativo no documento, caso positivo, a função ação é ativada, caso negativo, nada acontece. Sendo uma tecla, variáveis serão criadas para indicar que a aplicação irá usar uma tecla, permitindo assim, a biblioteca JavaScript criar uma tecla de configuração para que o usuário selecione a tecla correspondente a tecla que a aplicação usa (isso será explicado detalhadamente mais adiante). Caso a condição contenha o atributo *delay*, ao invés de ativar a função de ação, a condição irá ativar uma instrução *setTimeout* do JavaScript para que após o tempo descrito no atributo *delay*, a função de ação venha a ser executada. Os elos podem potencialmente aumentar em muitas linhas o documento, o que causa um impacto na performance do documento, porém o maior impacto é percebido quando se tem muitas mídias continuas ativas no documento.

4.5 Funcionamento do HTML5 resultante

Após terminar a tradução, é necessário controlar o código resultante através de funções que auxiliem o funcionamento do código NCL como os papéis, predefinido de um elo (*start, stop, pause, resume* e *set*) e o controle das teclas pressionadas pelo usuário. Para isso, a biblioteca JavaScript *ncl-complements.js* é chamada pelo código HTML resultante da transformação do XSLT. O *ncl-complements.js*, além de conter as rotinas citadas, faz a inicialização do documento criando uma tela inicial para o mapeamento das teclas do controle remoto (quando necessário). Para o controle do documento, são levadas em conta duas estruturas: uma é a associação visual e a outra é associação lógica dos Elementos. A associação visual dos elementos é realizada pelas estruturas regiões aninhada entre si,

que podem conter vários descritores que por sua vez, podem conter várias mídias, ou seja, uma região pode conter n regiões, uma região pode contar n descritores, e um descritor pode conter n mídias. Esta estrutura é acessada pelos *descriptorSwitch* ou por conectores que associam uma mídia a um descritor, essa estrutura também é usada na navegação e é usada para procura de propriedades quando uma mídia é ativada. A associação lógica é usada pelos contextos e *switchs*, que são elementos de associação que tem outros elementos dentro de si no NCL, porém, no HTML esta associação é comandada de outra forma: no caso do contexto, ela é feita por um atributo interno a cada elemento, o atributo *context*, que indica o contexto ao qual ele pertence; no caso do *switch*, o próprio *switch* contém as informações de quais elementos estão associados a ele.

Quando o documento HTML5 é iniciado, o JavaScript executa uma função ligada ao evento de carregamento e procura as variáveis que dizem se o documento recebe eventos de teclas, caso positivo, uma tela de configuração é executada para mapear as teclas do controle remoto no teclado do usuário. É possível se optar por salvar essas configurações para a próxima vez que a aplicação for aberta, para isso, a aplicação usa um *cookie*, além de gravar as configurações de teclas, é possível pedir a aplicação para não exibir mais a tela de configurações. Após isso, o documento é iniciado, para o tal, as mídias do tipo vídeo recebem um ajuste nos eventos JavaScript para que os mesmos ativem os eventos do NCL. Os eventos usados pela página HTML gerada vão ter o mesmo nome dos eventos de condição padrão no NCL (*onBegin, onEnd, onPause...*), pois serão eventos customizados criados pelo NCL4WEB. Dessa forma, será mais simples ter o controle de quando os eventos acontecem. Assim, se mantém a facilidade do HTML5 em disparar os eventos de mídia continua. Os eventos do HTML5 que são ligados aos eventos do NCL: *ended* é ligado ao *onEnd*, e *abort* é ligado ao *onAbort*. Apenas estes são ligados, pois são os únicos que podem acontecer num documento NCL onde todo controle que um usuário possa ter da mídia contínua é feito pela própria aplicação, logo, não tem sentido ligar outros eventos, visto que eles só serão ativados pelas próprias funções internas do documento. Desta forma, elas já são programadas pelo NCL4WEB para ativar os eventos customizados. Em seguida, todas as mídias recebem uma classe CSS que a retira do documento até que a mesma seja iniciada pela aplicação NCL.

Por fim, o evento “inicio", ativa todas as funções de *descriptorSwitch*, fazendo com que as mídias que se referenciam a eles sejam colocadas no descritor correspondente no estado inicial do documento. Posteriormente, é executado o contexto inicial de um documento NCL, o body, através da função auxilar “startContext", função que basicamente recebe um contexto e inicia todas as portas relacionadas a ele. A função que controla como

lidar com o evento de pressionar de teclas de um usuário, funciona da seguinte forma: primeiro verifica-se o foco, caso exista um elemento com o foco selecionado, a função procede para ver qual tecla foi pressionada, caso negativo, a função coloca o foco no elemento ativo com o menor valor de *focusIndex* (o valor *focusIntex* está presente no descritor, a função procura um elemento ativo que seja filho de um descritor com o menor *focusIndex*). Depois de lidar com o elemento que tem o foco, a função procura observar qual tecla foi apertada, caso seja uma das teclas direcionais (de navegação), a função vai procurar qual é o atributo do elemento que tem o foco relativo aquela direção apertada (*moveUp, moveDown, moveLeft* e *moveUp*). Em seguida, verifica se existe um elemento ativo, cujo descritor tenha um índice de foco corresponde ao valor do atributo de direção, caso positivo, o foco é atribuído ao mesmo (isso é feito colocando a classe css *active* no descritor da mídia). Ao atribuir o foco ao elemento, muda-se a variável que se refere ao foco (contém o nome "service.currentFocus") através da função *set*, as variáveis são marcadas como propriedades internas a um nó mídia de configuração (caso exista essa variável a função *set* irá mudar o foco, caso contrário a função que controla as teclas muda o foco). Caso a tecla pressionada seja o ENTER, a função irá executar o evento de clicar no elemento, o que por padrão, ativa a seleção do elemento através da função *select* (este elo entre o clique e a função *select* foi feito via o XSLT que cria o atributo *onclick* em cada elemento mídia acrescentando no mesmo uma chamada para função *select* contendo o elemento como atributo). Além do controle exercido, existem funções auxiliares para lidar com elementos que funcionam na mesma instância, funções para lidar com a troca de descritor em um *descriptorSwitch*, a associação de uma mídia a um novo descritor através de um elo e a função que controla todos os eventos criados por funções do *ncl-complements*. Dentro do *ncl-complements* existe uma função para cada papel de ação, predefinido do NCL: *start, stop, pause, resume, abort* e *set*. Essas ações tem alguns pontos em comum que são ilustrados na Figura 4.10. Repare que estas instruções podem ser executadas sobre mídias, *switchs* ou contextos, quando a instrução se refere a qualquer um destes chamaremos o nó de elemento. Os detalhes de cada função de ação serão explicados nas sub-seções abaixo.

4.5.1 Start

A função verifica se o nó em questão já foi iniciado (isso é feito verificando o atributo de estado da mídia e se ela contém a classe de visibilidade), caso positivo, a função para o processamento logo no inicio. Depois, verifica se o elemento que recebeu a instrução referencia a outro elemento e tem a mesma instância (neste caso o elemento é apenas

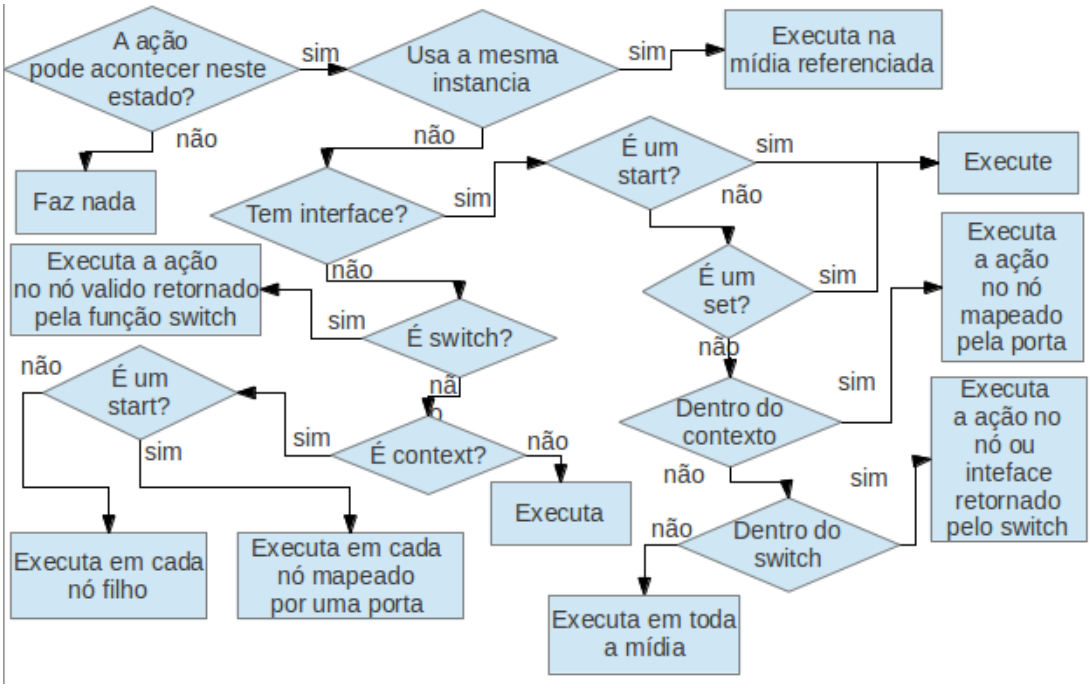

Figura 4.10: Diagrama geral de uma função de ação

uma referencia e não tem nenhuma informação de conteúdo), caso afirmativo a função *start* é executada no elemento principal (o que é referenciado). Após isso, é verificado se trata de uma mídia contínua, caso afirmativo, o atributo *preload* é modificado, para que o vídeo comece a ser buferizado enquanto é executado; caso a instrução tenha sido dada a uma âncora da mídia, é criada uma função, que quando a mídia contínua começa sua execução, a função a coloca na posição da âncora e lança o evento de início da âncora. Caso a mídia contínua, já tenha sido carregada sua posição é recolocada no 0; e por fim, executa-se a mídia continua. Em seguida, é verificado se a elemento é um *switch*, caso afirmativo uma função percorre o *switch* e encontra qual é o elemento tem a regra válida, se a função receber uma interface a busca é feita nos *swtchPorts*. Se não existir elemento com regra válida nem opção padrão a função retorna um valor vazio e a função start para a execução. Caso exista um elemento correspondente a regra válida, a função retorna este elemento e a função se executa recursivamente para o valor retornado (que pode ser um *id* ou *id* e *interface*). Se não for um *switch* é verificado se o elemento é um contexto, caso afirmativo o contexto é iniciado com a mesma função usada para iniciar o corpo do documento (*startContext*), caso exista uma porta atrelada a instrução somente o elemento referenciado na porta é iniciado. Caso não seja nem *switch* nem contexto se conclui que o elemento é uma mídia e é verificado se o elemento é filiado a uma região, caso positivo ele recebe um índice no eixo z de acordo com a região que ele se encontra (se a região tem um *zIndex* ele recebe o dela senão ele recebe um do contador de *zIndex* do NCL4WEB).

É testado se a mídia tem um descritor com um índice de foco, caso positivo e caso não exista um descritor com o foco, a mídia recebe o foco (caso exista uma variável do tipo *service.currentFocus* ela recebera o índice do foco atual). Em seguida, é o momento de executar a visualização da mídia, caso o descritor dele especifique uma transição, essa transição será iniciada e ao fim da mesma a visualização é feita (no momento todas as transições são interpretadas como *fadeIn* porem o objeto com as informações de cada transição está presente no documento e foi traduzido pelo *ncl4web.xsl*). Na visualização serão interpretadas as propriedades do elemento (através de uma função especifica que procura pelas propriedades da mídia e do descritor da mídia e as interpreta), o mesmo recebera o estado ativo e ira iniciar o evento de ativação do mesmo pela função responsável por isso. Se a transição não existir, a visualização é feita instantaneamente. Em seguida, são interpretados as âncoras que podem estar incluídas internamente na mídia ou fora dela, sendo referenciada a mesma por um atributo *parent* (o XSLT não coloca elementos filhos em todos os elementos usados para mídias). A busca pelas âncoras é feita de modo eficiente, de modo que, se não for encontrado um elemento filho é procurado um elemento com o atributo *parent* igual ao *id* do elemento. Essas âncoras se tornam elementos dentro da mídia que irão marca o tempo que elas estão em execução, uma função será executada cada intervalo de meio segundo para incrementar este tempo e ativar os eventos das ancoras quando necessário. Existindo uma propriedade na mídia ou no descritor de duração explícita, a mídia é marcada com um elemento dur, contendo a sua duração explicita. A mesma função que controla o tempo das âncoras será responsável por observar o atributo *dur*, e parar a mídia no tempo descrito. Essa função permanece em execução somente enquanto houver algum elemento que precise dela, seja por causa de uma âncora que ainda não terminou, seja por causa de alguma instrução de duração explicita que ainda não foi executada.

4.5.2 Stop e Abort

Essas duas ações têm praticamente o mesmo efeito prático, diferindo apenas no evento que elas disparam. Primeiro, é verificado se o elemento em questão está ocorrendo, para não se parar algo que já está parado, isso é feito verificando o atributo *state* do elemento. Assim como na instrução *start* (e em todas as demais instruções), o atributo *refer* e *instance* são verificados, para descobrir se o elemento é uma referência e tem a mesma instância. Caso afirmativo a instrução sera somente executada no elemento original (a função que controla os eventos se encarrega de repetir o evento nos elementos que referenciam o elemento original). Se for uma mídia contínua, ela receberá a instrução para parar e seu

atributo HTML *preload* será colocado em *none* (para parar a buferisação do conteúdo). Caso seja um contexto, sem especificar uma porta, todos os elementos iniciados que o referenciam serão receberão a mesma instrução (*stop* ou *abort*), caso seja especificada uma porta somente o elemento que a porta referencia receberá essa instrução (porta pode referenciar uma interface). Sendo um *switch* todos os elementos ao qual ele se refere receberão a mesma instrução. Caso seja uma mídia que esteja com o foco sobre ela, este foco será retirado e todas as âncoras que por algum motivo estejam sendo processadas pela função, que varre as âncoras, são excluídas. Os atributos que controlam a visualização da mídia são mudados para tornar o mesmo não só invisível como fora da área de visualização do documento (como no estado inicial da mídia). Se a instrução for *stop* ainda existe uma verificação, se há alguma transição, caso positivo, a mesma é feita (embora se tenha a informação da transição no documento, no momento o *ncl-complements* só realiza a transição de *fade*). É verificado se existe algum elemento em execução no contexto que o elemento pertence, se não houver, o contexto ao qual o elemento pertence recebe o evento de término que o elemento recebeu (*abort* ou *end*). Por fim, o elemento recebe o estado de sleeping.

4.5.3 Pause e Resume

São duas funções onde uma faz o inverso da outra. Ambas testam o estado do elemento. A função *pause* verifica se o elemento esta ocorrendo, caso afirmativo ela continua. A função resume verifica se o elemento está pausado antes de continuar. Caso o estado verificado não seja o desejado, a função para a execução. Se o elemento referencia outro e usa a mesma instância, a função é executada sobre o elemento referenciado somente. É verificado se o elemento é uma mídia contínua, sendo confirmado, ela é pausada (no caso do *pause*), ou recebe o *play* de novo (no caso do resume). Caso não seja, somente o seu estado é mudado. Em seguida, é verificado se o elemento em questão é um contexto, caso afirmativo, ele executa a instrução no elemento que a porta referencia, caso nenhuma porta seja especificada, todas as portas do contexto recebem a instrução. Se o elemento for um *switch* a instrução é executada em todos os elementos do *switch*, visto que o *pause* e resume só terão efeito sobre o elemento se ele estiver pausado ou em execução e um *switch* só pode ter um elemento ativo e os elementos inativos não receberão nenhuma ação visto que não estão no estado esperado. Logo os outros elementos não serão afetados e se a regra que iniciou o elemento do *swich* não for mais verdadeira, o elemento que está em execução sofrerá ação do mesmo jeito.

4.5.4 Set

A instrução set serve para modificar o valor de uma propriedade de um elemento. Assim como nas outras funções a primeira coisa a ser feita é verificar se o elemento se refere a algum outro e é da instância dele, caso afirmativo o elemento referenciado recebe a instrução set. O evento de atribuição é iniciado tanto no elemento que contem a propriedade como na propriedade em si (que é uma ancora do elemento principal). Caso seja especificada uma interface, é verificado se o elemento é uma mídia do tipo “settings"ou se é uma mídia comum. Se for uma mídia do tipo “settings"a propriedade é interpretada após ser modificada no elemento, se for o “service.currentFocus"o descritor referenciado por ele recebe o foco no lugar do descritor que tem o foco ativo. Se for um contexto a instrução é aplicada no elemento e interface que a porta do mesmo se refere. Se for um *switch* a instrução é aplicada no elemento e interface válidos pelo *switchPort* referenciado. Caso não seja especificada uma interface é verificado se o elemento é uma propriedade, se afirmativo ela recebe o valor que lhe foi dado pela instrução set caso negativo nada é feito. A instrução que gerencia os eventos inicia a atribuição do elemento principal quando se confirma que ele tem uma interface ou quando se confirma que o mesmo é uma propriedade, após fazer a mudança de variáveis o mesmo inicia o evento de fim de atribuição, estas funções deverão ser alteradas quando se implementar as propriedades avançadas de papéis customizados, algumas delas permitem mudar o tempo em que uma atribuição acontece.

4.5.5 Select

A função *select* não é um papel predefindo NCL, mas é a funçãoexecutada toda vez que uma mídia recebe um clique . Essa chamada é feito pelo *ncl4web.xsl* que ao criar a mídia coloca no atributo “onClick” uma chamada a função select com o id da mídia. O clique também é ativado quando o usuário aperta o botão *enter* selecionando a mídia em foco. A função *select* verifica se a instrução foi enviada a um elemento que faz referencia a outro e tem a mesma *instância* dele. Caso isso aconteça, somente o elemento referenciado recebe a instrução *select*, pois a função que gerencia os eventos se encarregará de enviar o evento na mídia que fez a referencia e tem a mesma *instância*. Após isso, é verificado se o elemento se encontra visível verificando suas classes, caso positivo, é verificado se o elemento contém um descrito com índice de foco, se isso for verdade, o foco é retirado do elemento que está com o foco (caso exista algum) e colocado no descritor do elemento que foi selecionado. Em seguida, a variável “service.currentFocus"é atualizada com o novo

valor de foco pela função *set* (da mesma forma que quando se usa uma tecla e navegação para navegar trocando o elemento em foco). A função ainda trata a seleção a contextos e *swtch*, selecionando o elemento especificado pela porta do contexto ou o nó especificado pelo *switch*, caso não exista um nó corresponde no caso do *switch* ou caso o contexto não especifique uma porta, a função não faz nada além de enviar o evento de seleção à função que gerencia os eventos.

4.5.6 Tratamento de eventos

A função que gere os eventos foi criada para separar das funções de ação o processamento de eventos, que muitas vezes é problemático e custoso (por conta de elementos que tem a mesma *instância*), além disso, ela torna as tarefas de depuração mais simples, visto que todos os eventos passam por elas antes de serem ativados. Essa função recebe o evento que será executado, o elemento, a interface (se houver), e uma variável que indica que a chamada foi feita por um nó de composição (essa variável é preenchida pelos contextos e *switchs* quando os mesmos chamam a função de ação novamente para os nós internos a eles. Neste caso não é necessário enviar o evento ao contexto que contem o elemento). A função verifica se há uma interface, se existe, ela envia o evento pela interface e então verifica se a chamada veio de um nó de composição, caso não tenha vindo, ela executa o evento em qualquer porta ou *switchPort* que referencia o elemento e sua interface. Caso não tenha sido enviado uma interface, a função envia o evento pelo elemento enviado e executa outra função que procura os elementos com a mesma instância e que se referem ao elemento enviado. Esta função coloca o estado deles igual ao do elemento original e envia o evento enviado pelo nó original aos nós que se referem a ele. Após isso, a função verifica se a chamada veio de um nó de composição, em caso negativo, todos os *switchs* e contextos que contém o nó recebem o mesmo evento. Por fim, todos os *switchPorts* que se referenciam ao elemento recebem o evento. Pois, iniciar o *switch* não implica em iniciar a porta do *switch* e a porta do *switch* não precisa necessariamente ter uma interface e ela pode se referenciar a um elemento que compartilha a instância com outros.

4.6 Uso e Comparação com o webNCL

Para usar o NCL4WEB existem duas possibilidades a primeira é colocar a linha “<?xml-stylesheet type="text/xsl"href="ncl4web.xsl"?>” no código NCL, ligando o mesmo a folha de estilo NCL4WEB. Para o tal o NCL4WEB necessita estar dentro do mesmo

servidor que o código NCL (caso seja executado em um navegador). Para o executar localmente é necessário usar uma opção no navegador habilitando o acesso do mesmo a arquivos locais. A segunda possibilidade é pré-compilar o código com a ajuda de um interpretador de XSLT(o proprio navegador por exemplo) e gravar o resultado como um arquivo HTML e usar este como a aplicação. Nenhum destes modos requer conhecimento de HTML somente o conhecimento básico de como interpretar um XSLT. O NCL4WEB foi testado em todas as aplicações do clubeNCL[1] que usam somente o NCL (A ferramenta ainda não suporta um código Lua). Todas funcionaram perfeitamente. As aplicações testadas foram: Roteiro do Dia, Jogo da Velha em NCL(*tictactoe*), Viva mais pratos, Primeiro João, Tur do Maranhão e Porta Retrato. A do Tur do Maranhão teve o código traduzido corretamente, no entanto, a aplicação ficou muito lenta nos navegadores testados. Nas demais aplicações, o funcionamento foi normal sem picos no uso de memória. A aplicação também foi testada em dispositivos móveis e apresentou limitações no *apple iOS* pelo mesmo só possibilitar a execução de um vídeo. O maior problema das aplicações é a compatibilidade com os navegadores, pois alguns não suportam nativamente o formato MP4 usado no Ginga-NCL (como por exemplo o FireFox), outros diapositivos móveis dizem suportar o XSL porém o mesmo não é executado da mesma forma que num navegador comum (impossibilitando a tradução no navegador). A fim de testar o trabalho pronto, foi feita uma comparação de desempenho do NCL4WEB com o webNCL, uma ferramenta capaz de traduzir o código NCL usando somente JavaScript. A versão atual do mesmo requer o uso de um servidor para o funcionamento, enquanto o NCL4WEB funciona nativamente em qualquer dispositivo se for pré-compilado, caso contrario apresenta algumas limitações em dispositivos android que embora tenham o XSLT instalado não fazem a tradução em seu navegador padrão nas versões mais antigas. Foram testadas duas aplicações que são exemplos do clubeNCL e estão disponíveis para teste em ambas aplicações: o Programa Viva Mais e o Primeiro João. Uma das principais diferenças das ferramentas em ambas as aplicações é que no webNCL é necessário usar um controle virtual que ocupa parte da tela enquanto no NCL4WEB aparece uma tela de configurações antes do inicio da aplicação. Para o teste foi usado o google-chrome versão 22.0.1229.94, no sistema operacional windows 7. Na primeira aplicação o NCL4WEB usou 22 kilobytes de memoria enquanto o webNCL usou 40 kilobytes, o google chrome faz uso de aceleração por GPU e nela a diferença não foi perceptível. No Primeiro João, a diferença foi de 37 kilobytes no NCL4WEB contra 50 kilobytes no webNCL. As aplicações do webNCL foram testadas nos *links* de exemplo da página do mesmo[2].

[2]webNCL - http://http://webncl.org/

A Figura 4.11 mostra a aplicação “Viva Mais"sendo executada no webNCL e no NCL4WEB, é possivel ver as diferenças entre um e outro, a primeira é o uso da tela que no NCL4WEB é toda a tela e no webNCL é menor para o uso do controle remoto (o NCL4WEB usa as teclas do teclado como controle remoto).

Figura 4.11: “Viva Mais"sendo executado no webNCL(acima) e no NCL4WEB(abaixo)

A diferença de otimização entre as duas aplicações é pouca, embora, o NCL4WEB só faça uso de uma função de sincronismo quando necessário e utilize o controle de eventos do próprio html5. Seu desempenho não foi muito melhor que o do webNCL ficando apenas usando apenas 30% a menos de memoria que o webNCL. Porém, o maior atrativo do NCL4WEB é que o programador não precisa ter nenhum conhecimento de HTML ou linguagem web para usá-lo, necessitando apenas saber usar o NCL, ou um editor do mesmo, como o NEXT[45] ou Composer[22]. Outra vantagem é que o webNCL ainda não é capaz de lidar com regras, *switchs, descriptorSwitch* ou importação de documentos[38], enquanto o NCL4WEB tem todas estas funcionalidades já implementadas.

4.7 Discussões, limitações do projeto e trabalhos futuros

Todas as implementações descritas neste capitulo foram criadas por se perceber seu uso em aplicações criadas no portal clubeNCL[1], como desenvolver toda a referência do NCL não foi possível em tempo hábil (isso é difícil até para os fabricantes de conversores) optou-se por desenvolver o que já estava sendo usado. Por exemplo, nenhuma aplicação do clubeNCL usou um papel diferente dos prê definidos, com exceção para casos de teste de valores de atributos, e nestes casos, o NCL4WEB é capaz de fazer a interpretação do papel (*atributoAssassement* e *valueAssassement*).

Caso queira tornar o código capaz de interpretar qualquer papel criado pelo usuário teria que se fazer o XSLT interpretar os papeis padrão e devolver uma função que sempre incluísse o *eventType* e o *transition* (para condições) ou *actionType* (para ações) e modificar a maneira como os eventos são processados e como as condições são feitas, pois hoje todas se baseiam em funcionar com base nos papeis padrão. Se fossem ser utilizados papéis customizados teria que ser a função de condição, sempre dependendo de uma função intermediaria ativada pela *transition* e *actionType* do papel especificado. É um trabalho bem grande para um resultado que ainda não se justifica. Nenhuma aplicação usa um papel fora do padrão e não existe uma razão para fazê-lo, pois os papéis padrão cobrem todos os tipos de interações possíveis.

Ainda não foram implementadas todas as transições, em parte, por que a biblioteca do JQuery[30] usada para manipular os elementos do HTML não implementa transições, nada além de um *fadIn* e *fadeOut*. Poderiam ser implementadas todas as transições, mas somente um exemplo do primeiro joão usa um tipo diferente de transição (o *barWipe*). Para se implementar todos seria necessário criar essas transições no JavaScript já que o XSLT do NCL4WEB já coloca as informações das transições no documento em um objeto JavaScript. No momento caso exista uma transição por padrão o NCL4WEB faz um *fadeIn* ou *fadeOut* com o tempo contido no atributo *dur* da transição. Só é suportado ancoras temporais no momento.

O NCL4WEB procurou usar o atributo *preload* das *tags* HTML *video* e *audio* em *none* por padrão para otimizar o processamento de documento, evitando que seja necessário se carregar vários vídeos quando apenas um é exibido. Quando uma mídia é iniciada, este atributo volta ao valor padrão que é auto. Outra grande limitação é a incapacidade de se exibir documentos que tenham código em Lua, o que pode ser contornado colocando

conteúdo dos códigos em Lua dentro do documento e traduzindo os mesmos para um *script* correspondente em JavaScript, embora *possível*, não é algo facilmente *implementável*.

A norma do NCL também permite a importação de todo um documento no cabeçalho[38] isso não foi implementado no NCL4WEB ainda, pois embora seja relativamente fácil colocar no documento os elementos importados isso iria requerer uma nova abordagem dos *templates* fazendo com que todos os *templates* fossem chamados por um maior e apenas o cabeçalho e o corpo fossem chamados por *templates* de filtragem. Pois todos os elementos poderiam do documento ou do documento importado, além disso, o documento importado pode importar outros documento o que pode gerar um *loop* de importação possivelmente infinito. O que não é impossível de se controlar no XSLT, mas não é trivial.

Também no cabeçalho é possível definir-se diferentes dispositivos (multi-device) para cada base de regiões desta forma uma mesma aplicação será exibida de um modo diferente em outro dispositivo, embora tal funcionalidade seja implementável com a ajuda do objeto "navigator" do JavaScript ela não foi implementada ainda no NCL4WEB.

A melhor solução para implementação desses recursos seria uma parceria com o projeto WebNCL[26] que já implementou algumas transições e tem se concentrado em traduzir o Lua para JavaScript. Uma possível parceria entre os projetos poderia beneficiar ambos, sem que eles perdessem, sua principal característica que é a abordagem: o WebNCL se concentra em exibir um documento NCL num ambiente web sem se preocupar com manter as características do mesmo como uma mídia única(visto que não usa toda tela para o tal). Já o NCL4WEB procura transportar a aplicação NCL para o ambiente web, mantendo suas características o mais parecido com as originais quando exibido na televisão digital sem requerer nenhuma alteração no código ou trabalho extra por parte do desenvolvedor.

4.8 Conclusão

O JNS procurou extender o poder de expressão e simplificar a linguagem NCL, embora talvez não tenha sido a melhor abordagem ele demonstrou algumas estratégias que podem ser aproveitadas em novas linguagens ou mesmo numa nova versão do mesmo.

O JNS tem um site contendo a sua documentação e um tutorial de como usá-la. O uso é aberto e livre a todas as aplicações gratuitas. O site ainda contém um compilador e um complemento para o editor de texto gedit. O projeto se inspirou em alguns trabalhos para conceber algumas de suas facilidades, mas inovou ao usar o JSON. A abordagem minimalista e a utilização de mais estruturas que sejam familiares a programadores foram às chaves para a construção da linguagem.

Em trabalhos futuros, se espera aperfeiçoar a linguagem com o uso de YAML[3] no lugar de JSON tornando ela compatível com JSON mas também compatível com um formato de identação sem os colchetes ou chaves tornando a linguagem mais agradável ao usuário. O projeto está disponível em [4] e se espera que a comunidade ajude a melhorá-lo com críticas ao mesmo.

O NCL4WEB permite a exibição de códigos em NCL que seguem a norma, implementado boa parte da norma NCL e permitindo que todas as aplicações que usam somente NCL divulgadas no clubeNCL possam ser exibidas em um navegador que tenha suporte ao HTML5, e a vídeos no formato mpeg4[25] algo que os principais navegadores já implementam[5]. A ferramenta tem um desempenho ligeiramente melhor que o webNCL e a capacidade de interpretar estruturas que o mesmo ainda não é capaz. Por usar um XSLT a ferramenta não requer que o programador tenha qualquer conhecimento de HTML5. Espera-se tornar mais fácil a apresentação de aplicações para televisão digital, permitindo que desenvolvedores possam divulgar seus portfólios na internet sem necessitar ir até o cliente e usar um artifício extra para exibir sua aplicação.

A ferramenta está disponível para modificação e utilização em aplicações não comerciais, gratuitamente e para aplicações comerciais mediante a consulta, no site da mesma[6]. Ainda existem alguns detalhes da norma que não foram implementados, porém, já é possível criar aplicações bem complexas com isso. Uma utilização da ferramenta foi o tutorial da linguagem JNS que usa a ferramenta para exibir os exemplos ao usuário [7].

[3]YAML Oficeal WebSite - http://www.yaml.org/
[4]JNS documentação - http://www.midiacom.uff.br/ caleb/documentacaoJNS
[5]Video Formats and Browser Support - http://http://www.w3schools.com/html/html5_video.asp
[6]NCL4WEB site - http://www.midiacom.uff.br/ caleb/ncl4web
[7]Aprenda JNS - http://www.midiacom.uff.br/čaleb/documentacaoJNS/index.html?pagina=aprenda

Nos próximos trabalhos, a expectativa é implementar a importação de documentos e o uso de papéis customizados. Entretanto, o mais útil é sem duvida, criar um modo de traduzir um código Lua para o javaScript e tornar a ferramenta ainda mais poderosa no quesito de exibir aplicações NCL.

Espera-se que estas duas aplicações, mesmo que não sejam usadas, sirvam de inspiração e base para o desenvolvimento de tecnologias para a linguagem NCL, buscando aumentar o poder de expressão da mesma e facilitar a exibição de documentos da mesma em outros meios.

Referências

[1] Oficial site of ClubeNCL, 2011. Disponível em http://clube.ncl.org.br , Acesso em 18/04/2013.

[2] ABNT, N. 15606. *Associação Brasileira de Normas e Técnicas, Digital terrestrial television–Data coding and transmission specification for digital broadcasting* (2011).

[3] Berglund, A.; Boag, S.; Chamberlin, D.; Fernandez, M. F.; Kay, M.; Robie, J.; Siméon, J. Xml path language (xpath) 2.0. *W3C recommendation 23* (2007).

[4] Bevan, N.; Barnum, C.; Cockton, G.; Nielsen, J.; Spool, J.; Wixon, D. The magic number 5: is it enough for web testing? In *CHI'03 extended abstracts on Human factors in computing systems* (2003), ACM, pp. 698–699.

[5] Bezerra, D.; Sousa, D.; Burlamaqui, A.; Silva, I., et al. Luar: a language for agile development of ncl templates and documents. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web* (2012), ACM, pp. 395–402.

[6] Brasil, P. Industria eletrônica digital - tv digital, primeiro paragrafo., 2012. Disponível em http://ww.brasil.gov.br/sobre/ciencia-e-tecnologia/industria-eletronica-digital/tv-digital, Acesso em 18/04/2013.

[7] Bulterman, D.; Rutledge, L. *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer Publishing Company, Incorporated, 2008.

[8] Claudia Tozetto, i. S. P. Apos 5 anos a televisão digital ainda engatinha no brasil., 2012. Disponível em http://convergenciadigital.uol.com.br/cgi/cgilua.exe/sys/start.htm?infoid=31700&sid=111, Acesso em 18/04/2013.

[9] Costa, R. M. d. R.; Moreno, M. F.; Gomes Soares, L. F. Intermedia synchronization management in dtv systems. In *Proceedings of the eighth ACM symposium on Document engineering* (2008), ACM, pp. 289–297.

[10] Crockford, D. Json: Javascript object notation, 2011. Disponível em http://json.org, Acesso em 18/04/2013.

[11] Ecma, E. 262: Ecmascript language specification. *ECMA (European Association for Standardizing Information and Communication Systems), pub-ECMA: adr,* (1999).

[12] Federal, G. Portaria interministerial mdic/mcit no140, de 23 de fevereiro de 2012., 2012. Disponível em http://www.apet.org.br/noticias/img/integra_Portaria_Interministerial_MDIC_MCIT_n140 2012.pdf, Acesso em 18/04/2013.

[13] Ferraiolo, J.; Jun, F.; Jackson, D. Scalable vector graphics (svg) 1.1 specification. *World Wide Web Consortium (W3C). URL http://www. w3. org/TR/SVG11* (2003).

[14] Gaggi, O.; Danese, L. A SMIL player for any web browser. *DMS 2011* (2011), 114–119.

[15] Garrett, J. J., et al. Ajax: A new approach to web applications.

[16] Han, J.; Haihong, E.; Le, G.; Du, J. Survey on nosql database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on* (2011), IEEE, pp. 363–366.

[17] Ierusalimschy, R.; De Figueiredo, L.; Filho, W. Lua-an extensible extension language. *Software Practice and Experience 26*, 6 (1996), 635–652.

[18] Jansen, J.; Bulterman, D. Smil state: an architecture and implementation for adaptive time-based web applications. *Multimedia Tools and Applications 43*, 3 (2009), 203–224.

[19] JavaDTV, A. Java dtv api 1.3 specification, sun microsystems, 2009, 2010.

[20] JavaTV, A. Java tv specification 1.1-jsr 927. *Sun Microsystems. Disponível em URL: http://jcp. org/en/jsr/detail, Acesso em 18/04/2013* (2008).

[21] Jelliffe, R. The schematron: An xml structure validation language using patterns in trees. *URL: http://xml.ascc.net/resource/schematron/schematron.html* (2001).

[22] Laiola Guimarães, R.; Monteiro de Resende Costa, R.; Gomes Soares, L. Composer: Authoring tool for itv programs. *Changing Television Environments* (2008), 61–71.

[23] Lassila, O.; Swick, R. R., et al. Resource description framework (rdf) model and syntax specification.

[24] Lima, G.; Soares, L.; Neto, C.; Moreno, M.; Costa, R.; Moreno, M. Towards the ncl raw profile. *WebMedia2010* (2010).

[25] Marpe, D.; Wiegand, T.; Sullivan, G. J. The h. 264/mpeg4 advanced video coding standard and its applications. *Communications Magazine, IEEE 44*, 8 (2006), 134–143.

[26] Melo, E.; Viel, C.; Teixeira, C.; Rondon, A.; Silva, D.; Rodrigues, D.; Silva, E. Webncl: A web-based presentation machine for multimedia documents. In *Brazilian symposium on Multimedia and the web* (2012), pp. 403–410.

[27] Notícias, R. Desenvolvimento de aplicativos para smartphones cresce no brasil., 2012. Disponível em http://www.redetv.com.br/Video.aspx?118,31,293430,jornalismo,redetv-noticias,desenvolvimento-de-aplicativos-para-smartphones-cresce-no-brasil, Acesso em 18/04/2013.

[28] Project, T. G. gedit text editor., 2007. Disponível em http://projects.gnome.org/gedit/, Acesso em 18/04/2013.

[29] Resig, J., et al. jquery: The write less, do more, javascript library. *Disponıvel em http://jquery. com/, Acesso em 18/04/2013 18*, 04 (2009), 2009.

[30] Resig, J., et al. jquery: The write less, do more, javascript library. *Disponıvel em http://jquery. com/, Acesso em 18/04/2013 18*, 04 (2009), 2009.

[31] Rodrigues, C.; Afonso, J.; Tomé, P. Mobile application webservice performance analysis: Restful services with json and xml. *ENTERprise Information Systems* (2011), 162–169.

[32] Santos, J. *Multimedia and hypermedia document validation and verification using a model driven approach.* Tese de Doutorado, Universidade Federal Fluminense, 2012.

[33] Soares, L. *Programing in NCL 3.0: Developing applications to the Ginga middleware Ginga: digital TV and Web.* Elsevier, 2009.

[34] Soares, L.; Lima, G.; Soares Neto, C. Ncl 3.1 enhanced dtv profile. *Workshop De Tv Digital Interativa em WebMedia 2010* (2010), 1–2.

[35] Soares, L.; Rodrigues, R. Nested context language 3.0. Tech. rep., Informatics Department, PUC-Rio, Rio de Janeiro, 2006.

[36] Soares, L. F. G.; Moreno, M. F.; De Salles Soares Neto, C. Ginga-ncl: declarative middleware for multimedia iptv services. *Communications Magazine, IEEE 48*, 6 (2010), 74–81.

[37] Soares, L. F. G.; Rodrigues, R. F. Nested Context Model 3.0 Part 1 - NCM Core, May 2005.

[38] Soares, L. F. G.; Rodrigues, R. F.; Moreno, M. F. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society 12*, 4 (2007), 37–46.

[39] Soares Neto, C.; Soares, L.; de Souza, C. TAL-Template Authoring Language. *Journal of the Brazilian Computer Society* (2012), 1–15.

[40] Souza Filho, G. L. d.; Leite, L. E. C.; Batista, C. E. C. F. Ginga-j: The procedural middleware for the brazilian digital tv system. *Journal of the Brazilian Computer Society 12*, 4 (2007), 47–56.

[41] Strings, L. Backus-naur form. *Formal Languages syntax and semantics Backus-Naur Form 2 Strings, Lists, and Tuples composite data types* (2010).

[42] Team, A. The ambulant open source smil player, 2010. Disponível em http://www.ambulantplayer.org/, Acesso em 18/04/2013.

[43] UOL, A. P. L. C. D. Tv digital: Países da al harmonizam o uso do ginga., 2010. Disponível em http://convergenciadigital.uol.com.br/cgi/cgilua.exe/sys/start.htm?from_info_index=11&infoi , Acesso em 18/04/2013.

[44] UOL, A. P. L. C. D. Mercado de televisões promete ter crescimento similar ao de celulares., 2012. Disponível em http://tecnologia.ig.com.br/especial/apos-cinco-anos-ginga-ainda-engatinha-no-brasil/n1597734544041.html, Acesso em 18/04/2013.

[45] Varanda da Silva, J.; Christina Muchaluat-Saade, D. Next: graphical editor for authoring ncl documents supporting composite templates. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web* (2012), ACM, pp. 387–394.

[46] W3C. XSL transformations (XSLT) version 1.0, 1999.

[47] W3C. HTML5: A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html5/, Acesso em 18/04/2013, 2011. World-Wide Web Consortium Working Draft.

APÊNDICE A – Documentação JNS

Neste apéndice será apresentada a documentação do JNS completa incluindo os valores possíveis a cada atributo e forma geral de cada elemento.

A.1 Head

No cabeçalho estão as estruturas que definem a forma do documento, as áreas que serão utilizadas pelas mídias e as relações de sincronismo e interatividade que o documento contém. É possível especificar transições para os descritores e regras para switches no cabeçalho. A Listagem A.1 mostra a forma de um cabeçalho:

Listagem A.1: Estrutura do Head

```
1        head:[
2        {region:{?}},
3        {descriptor:{?}},
4        {rule:{?}},
5        {descriptorSwitch:{?}},
6        {connector:{?}},
7        {transition :{?}},
8        {include:{?}},
9        {meta:{?}},
10       {metadata: RDF tree } ,
11       ] ,
```

Pode-se perceber que em JNS não existe bases de regiões, descritores, conectores, transições ou regras, todos os elementos são declarados diretamente no cabeçalho.

A.1.1 Region

A região é a estrutura que define o espaço que um elemento deve ocupar, ou seja, tudo que ela define é possível ser definido em um parâmetro do descritor ou mesmo dentro de uma propriedade da mídia, entretanto, a região pode especificar diferentes regiões de exibição,

de acordo com o aparelho em que o documento é interpretado através da propriedade *device*.

Listagem A.2: Especificação da região

```
{region:{
                id:"string",
                parent: idDeUmaRegion ,
                device:"string",
                title:"string",
                left:Inteiro ou porcentagem,
                right:Inteiro ou porcentagem,
                top:Inteiro ou porcentagem,
                bottom:Inteiro ou porcentagem,
                height:Inteiro ou porcentagem,
                width:Inteiro ou porcentagem,
                zIndex:Inteiro
        }}
```

O atributo *id* é obrigatório, responsável pelo nome único o qual se referencia aquela região. O atributo *parent* define a região que contém a região (como não é tão trivial aninhar objetos em JSON a referencia da região "pai" é posta nas regiões "filhas"). Caso não tenha, permanece em branco. O atributo *device* indica o dispositivo que irá exibir essa região, funcionalidade que permite um mesmo documento NCL ser visível em vários dispositivos diferentes. O *title* declara um titulo para a região. Os atributos *top, bottom, left* e *right* revelam a posição da região e podem ser definidos por um inteiro representando pixels, ou uma % em relação ao tamanho total. Os atributos *width* e *height* descrevem o tamanho da região. O atributo *zIndex* é um inteiro que varia de 0 a 255 e define que regiões com maior valor, serão posicionadas sobre regiões de menor valor.

A.1.2 Descriptor

O descritor descreve como uma mídia será exibida. Nele estão os elementos de navegação que ditam como o documento será navegável pelas teclas do controle remoto, através do valor do índice de foco e também como será a borda do documento quando o mesmo está em foco. É possível especificar uma duração explícita ao conteúdo que o referencia. Além disso, o descritor define como será a transição no início e fim do documento. Existe ainda uma série de parâmetros do descritor que definem outras opções para exibição do documento, que serão explicadas na tabela. A estrutura do descritor pode ser vista na Listagem A.3.

Listagem A.3: Especificação do descritor

```
1         {descriptor:{
2                 id:"string",
3                 player:"ferramentaDeAresentacao",
4                 explicitDur:inteiro,
5                 region:"idDaRegiao",
6                 freeze:boolean,
7                 focusIndex:inteiroUnico,
8                 moveLeft:umFocusIndex,
9                 moveRight:umFocusIndex,
10                moveUp:umFocusIndex,
11                moveDown:umFocusIndex,
12                transIn: idTransicao ,
13                transOut :idTransicao ,
14                focusBorderColor:"nomeDeUmaCor",
15                focusBorderWidth:"numero",
16                focusBorderTransparency:"porcentagem",
17                focusSrc:"caminho De uma Media A Ser Exibida Em caso De Foco",
18                focusSelSrc:"Idem focusSrc mas quando selecionado",
19                selBorderColor:"nomeDeumaCor",
20                descriptorParams:[{"StringParametro":"StringValor"}] // objetos
21        }}
```

O *id* contém o identificador único como nos demais elementos. O *player* é uma string responsável por especificar a ferramenta de apresentação da mídia que referencia o descritor. O *explicitDur* especifica em segundos a duração que a mídia que referencia o descritor deve ter. O atributo *region* referencia a região que será utilizada. O *freeze* define se ao término da apresentação, o último frame da mídia continuará a ser exibido. O *focusIndex* revela o índice do descritor, utilizado para definir a navegação pelas teclas em conjunto com os *moveLeft, moveRight, moveUp* e *moveDown*. Os atributos move contêm o índice do descritor que deverá ser selecionado ao se usar o botão de navegação do controle remoto. Os atributos *transIn* e *transOut* indicam qual transição será usada na entrada e na saída, transição esta, criada no objeto transition. O *focusBorderColor* sugere uma cor para a borda, as cores são as cores padrão: *white , black , silver , gray , red , maroon , fuchsia , purple , lime , green , yellow , olive , blue , navy , aqua* e *teal*. O *focusBorderWidth* é um inteiro que diz o tamanho da borda, variando de 0 a 1. O *focusBorderTransparency* identifica a porcentagem da borda que deve ser transparente. O *focusSrc* indica o caminho de uma mídia que deve ser exibida quando o conteúdo estiver em foco. O *focusSelSrc* especifica uma mídia alternativa para quando o elemento for selecionado pela tecla de seleção. O *selBoderColor* também é uma cor padrão para a borda quando o elemento for selecionado. Os descriptorParams podem assumir os valores descritos na tabela A.1.

Listagem A.4: Exemplos de descritor

```
        {descriptor:{id:"dSelecao1",region:"rgSelecao1",focusIndex:1,moveUp:2,moveDown:3,
                        moveLeft:3,moveRight:2}},
        {descriptor:{id:"dSelecao2",region:"rgSelecao2",focusIndex:2,moveUp:3,moveDown:1,
                        moveLeft:1,moveRight:3}},
        {descriptor:{id:"dSelecao2",region:"rgSelecao3",focusIndex:3,moveUp:1,moveDown:2,
                        moveLeft:2,moveRight:1}},
```

A Listagem A.4 traz o exemplo de descritores para um menu de navegação com três opções. Verifique o atributo *focusIndex* de cada um deles, é ele que é referenciado pelos atributos *move*. Como é visto na A.2.2 muitos destes atributos também podem ser definidos dentro da mídia.

A.1.3 Transition

As transições marcam a mudança de uma mídia ao ser exibida e ao ser encerrada. A transição é ligada ao descritor da mídia.

Listagem A.5: Especificação da transição

```
        {transition:{
                id:"string",
                type: stringDoTipo ,
                subtype: stringDoSubtipo ,
                dur: inteiro,
                startProgress: real de zero a 1 ,
                endProgress: real de zero a 1 ,
                direction: forward ou reverse ,
                fadeColor: cor,
                horzRepeat: inteiro ,
                vertRepeat: inteiro ,
                borderWidth: inteiro ,
                borderColor: cor blend
        }}
```

O elemento *dur* mostra a duração em segundos, caso não seja especificado, o padrão é 1. Os elementos *startProgress* e *endProgress* definem a quantidade de efeito no início e no final da transição, variando de 0 para nenhum efeito a 1 para com efeito completo, os valores padrão são 0 e 1 respectivamente. O elemento *direction* diz em qual direção ocorrerá a transição, os valores permitidos são forward e reverse, o padrão é forward e nem todas as transições interpretam essa propriedade de modo relevante. O *fadeColor* especifica a cor para a qual a transição irá, ele é uma cor podendo assumir uma das cores predefinidas: *white , black , silver , gray , red , maroon , fuchsia , purple , lime , green , yellow , olive , blue , navy , aqua* e *teal*. Os atributos *horzRepeat, vertRepeat, borderWidth* e *borderColor* se originam da especificação SMIL. O *horzRepeat* descreve

o número de repetições da transição no eixo horizontal e o *vertRepeat* no vertical. O *borderWidth* é a grossura da borda e o *borderColor* que diz a cor da borda, ou pode ser uma mistura das cores das fontes de mídia (blend). O *type* e *subtype* definem o tipo de transição. O type indica a animação e o subtype especifica a maneira como a mesma se comporta, no entanto, existem somente cinco tipos obrigatórios e cinco subtipos padrão para os mesmos conforme a tabela A.2. Dependendo do aparelho que executa o código NCL é possível usar outros tipos e subtipos.

A.1.4 Rule

Uma regra declara o atributo id para ser identificada. Em um programa JNS, não existe mais uma base de regras e todas ficam guardadas diretamente dentro do cabeçalho. As regras são declaradas através de uma expressão booleana, semelhante a uma expressão booleana em C. Onde se compara uma variável, declarada no documento por uma propriedade mídia do tipo *application/x-ginga-settings*, a uma string usando um comparador("==" ; "<" ;">" ; "!=" ; "<=" ; ">=" ; "eq" ; "lt" ; "gt" ; "ne" ; "gte" ; "lte"). A forma léxica desta expressão é descrita na forma BNF[41] ("string" se refere a uma cadeia de caracteres qualquer, "number" a um número e | indica uma alternativa) apresentada na Listagem A.6.

Listagem A.6: Especificação da expressão da regra

```
1 <expression> ::= <var><comparador><valor> | "("<expressão>")"<operador>"("<expressão>")"
2 <var> ::= "string"
3 <comparador> ::= "==" | "<" | ">" | "!=" | "<=" | ">=" | "eq" | "lt" | "gt" | "ne" |
      "gte" | "lte"
4 <valor> ::= "string" | number
5 <operador> ::= "and" | "or"
```

Na regra, o único elemento além da expressão é o *id* obrigatório e único. A expressão pode conter uma variável declarada no corpo ou uma variável temporária, que será substituída por uma real dentro do *switch* ou *descriptorSwitch* que a usa.

Listagem A.7: Especificação da regra

```
1         {rule:{
2                 id:"stringId",
3                 expression:"expressão descrita" descriptorSwitch
4         }}
```

É possível ver alguns exemplos de regra na Listagem A.8, nestes exemplos a regras comparam a variável idioma a varias siglas.

Listagem A.8: Exemplos de regras

```
1        {rule:{ id:"idiomaIngles", expression:"idioma==ingles"}}
2        {rule:{ id:"idiomaPortugues", expression:"idioma==portuges"}}
3        {rule:{ id:"idiomaEspanhol", expression:"idioma==espanhol"}}
```

Também é possível se especificar as regras dentro do *switch* que a usa (*descriptorSwitch* ou *switch*).

A.1.5 DescriptorSwitch

O *descriptorSwitch* é utilizado para fazer uma seleção entre descritores, com base em determinadas regras. Através de regras declaradas em um elemento *rule* ou por meio de expressões declaradas, dentro do próprio *descriptorSwitch*. Caso o elemento venha usar uma regra que tenha uma variável temporária, ele deve ligar a variável temporária a uma real, através do atributo *vars*. O *id* é a identificação única. Para se referenciar a regra, basta colocar o id da regra como nome do atributo. Para criar uma regra dentro do *descriptorSwitch*, usa-se a própria expressão como nome do atributo. O id da regra ou a expressão devem indicar um novo descritor ou um id de um descritor já existente no descriptorSwitch. O atributo *default* indica qual deve ser o descritor padrão a ser ativado no *descriptorSwitch*, especificando um novo descritor ou referenciando um descritor já existente no descriptorSwitch com o seu *id*. A expressão interna ao *descriptorSwitch* é idêntica a presente na regra, exceto que neste caso, a variável deve existir no corpo do documento.

Listagem A.9: Especificação do descriptorSwitch

```
1        {descriptorSwitch:{
2                id:"stringLivre",
3                vars:[?], // {"stringDaVarNaRule":"idDaVariavel"},
4                "expressao":{descriptor:{?}|| idDescritor },
5                "idDaRegra":{descriptor:{?}|| idDescritor },
6                default: {descriptor:{?}|| idDescritor },
7        }}
```

Um exemplo de descriptorSwitch pode ser visto na Listagem A.10. Neste, é selecionado um descritor diferente dependendo do valor da variável propaganda.

Listagem A.10: Exemplo de descriptorSwitch

```
1        {descriptorSwitch:{
2                id:"dTela",
3                "propaganda='false'":{descriptor:{id:"dTelacheia",region:"rgFull"}},
4                "propaganda='true'":{descriptor:{id:"dMeiaTela",region:"rg50"}},
5                "propaganda=0":"dTelacheia",
```

```
6                   "propaganda=1":"dMeiaTela"
7           }}
```

A.1.6 Connector

Um conector é utilizado para descrever relações de sincronismo e reações a eventos feitos por interações do usuário. O não especifica os elementos participantes das relações apenas descreve-as, os elementos são especificados pelo elo. As relações dos conectores são especificadas por uma condição e uma ação. Dentro da condição, pode haver varias sub condições e parâmetros que podem ser preenchidos mais tarde no elo, assim como a ação pode conter outras sub ações e parâmetros. O conector deve ser referenciado por um elo para ser usado no documento NCL, o elo irá fazer o uso das condições e ações definidas no conector, relacionando elas a elementos do corpo do documento NCL. A estrutura de um conector pode ser vista na Listagem A.11.

Listagem A.11: Especificação do conector

```
1 {connector:{
2         id:"stringLivre",
3         params:[ stringparâmetro0 , stringparâmetro1 ,...,stringparâmetroN] ,
4         expression:"stringDaExpressão"
5 }}
```

O conector é composto pelo extitid, a expressão que o compõe, e pelo atributo extitparams, este define os atributos que serão usados dentro do conector. Assim como na regra, a função dos conectores é definida por uma expressão. Tal expressão é especificada na Listagem A.12.

Listagem A.12: BNF da expressão do conector

```
1         <expression>::=<condList>"then"<actList>
2         <condList> ::=
              <assessment>|<condition>|"("<condList>")"<operador>"("<condList>")" |
3 <condList> "with" "delay" "=" "string"
4         <assessment>::= <attr><comparador><attr>|<attr><comparador><valor>
5         <attr> ::= string "with" <params>
6         <params> ::= "idref" "=" 'string'| <params>" , "<params>
7         <comparador> ::= "=="|"<"|">"|"!="|"<="|">="|"eq"|"lt"| "gt"|"ne"|"gte"|"lte"
8         <valor> ::= "string"|number
9         <condition> ::= <condRole>|<condRole> "with" <params>
10        <condRole> ::= "onAbort" | "onBegin" |"onBeginAttribution" | "onEndAttribution"
              | "onEnd"|"onPause"|"onResume" | "onSelection"|"String"
11        <actList> :: = <action>| "("<action>")"<sincronis>"("<action>")"|<actList>
              "with" "delay" "=" "string"
12        <sincronis> ::= "||" | ";"
13        <action> ::= <actionRole>|<actionRole> "with" <params>
```

```
14          <actionRole> ::= "abort" | "pause" | "resume" | "start"| "stop"| "string" | "set"
15          <operador> ::= "and"|"or"
```

Os símbolos "||" e ";" se referem a como as ações devem ser executadas quando estão em conjunto, significando paralelo e seqüencial respectivamente. As declarações de instrução de avaliação (*assessmentStatement*) são feitas através do novo papel (*role*), sendo igualado a um valor ou a um parâmetro, com seus devidos atributos *eventType* e *attributeType* e explicitados com a ajuda do *with*. As instruções compostas são declaradas através de uma expressão aninhada com parênteses, separada por *and* ou *or*. A *string* que o *idref* referencia deve estar entre ', não pode conter espaços em branco (" ") e a vírgula que os separa os parâmetros deve ter o espaço antes e depois (como exibido). Os valores possíveis para os *idrefs* estão especificados segundo a norma NCL[35] e são listados na tabela A.1.6.

Tabela A.3: Parâmetros de um connector

Parâmetro	**Valor**
delay	Define um atraso até que a condição seja ativada ou a ação aconteça. O valor é um número real ou um parâmetro previamente declarado em params com o "$" antes do início do nome. O *delay* é o único parâmetro que pode ser usado para conjuntos, seja de ações ou de condições.
min	Define o número mínimo de componentes que referenciam o *role*. O valor é um número inteiro. Pode ser usado tanto em condições como em ações.
max	Define o número máximo de componentes que referenciam o *role*. É um número inteiro ou a palavra reservada *unbounded* (caso o número máximo seja infinito). Pode ser usado tanto em condições como em ações.
qualifier	Define como os múltiplos componentes que referenciam o *role* devem ser qualificados. No caso de *roles* de condição é possível assumir os valores: *and* (verdadeira somente se todas as condições associadas forem verdadeiras) e *or* (verdadeira se qualquer condição associada for verdadeira). No caso de *roles* de ação é possível assumir os valores: *seq* (todas as ações associadas são executadas na seqüência declarada) e *par* (todas as ações são executadas em paralelo).
eventType	Define o tipo de evento é usado quando se é usado um role diferente do padrão ou um *assessment*. Os valores possíveis são: *presentation, attribution* e *selection*. Sendo o *selection* usado somente para condições.

continua na próxima página

Tabela A.3: Parâmetros de um connector

Parâmetro	**Valor**
key	Exclusivo para condição e *assessment*. Ele define uma tecla que deverá ser apertada para validar a condição. Os valores possíveis são as teclas: “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”, “I”, “J”, “K”, “L”, “M”, “N”, “O”, “P”, “Q”, “R”, “S”, “T”, “U”, “V”, “W”, “X”, “Y”, “Z”, “*”, “#”, “MENU”, “INFO”, “GUIDE”, “CURSOR_DOWN”, “CURSOR_LEFT”, “CURSOR_RIGHT”, “CURSOR_UP”, “CHANNEL_DOWN”, “CHANNEL_UP”, “VOLUME_DOWN”, “VOLUME_UP”, “”, “ENTER”, “RED”, “GREEN”, “YELLOW”, “BLUE”, “BACK”, “EXIT”, “POWER”, “REWIND”, “STOP”, “EJECT”, “PLAY”, “RECORD”, “PAUSE”. Ou um parâmetro previamente declarado no *params* colocando um “$” antes do inicio do nome.
transition	Específico para condição. Serve para determinar o tipo de transição que ativa uma condição em um role diferente do padrão. Os valores possíveis são: *starts, stops, aborts, pauses* e *resumes*.
actionType	Usado somente em roles de ação. Especifica o tipo da ação em um role de ação diferente do padrão. Os valores possíveis são:*start, stop, pause, resume* e *abort*.
value	Atributo que define o valor que será atribuído no elemento que referencia o role *set* ou um role criado pelo usuário com o *eventType* do tipo *atribuition*. É possível usar um parâmetro previamente declarado em *params* colocando um “$” antes do inicio do nome do parâmetro como valor. Esta propriedade funciona somente com roles de ação.
repeat	Define o número de repetições que a ação deve ocorrer. O valor é um número inteiro positivo.
repeatDelay	Especifica o tempo que se deve esperar entre as repetições da ação. Seu valor é um número real ou um parâmetro previamente declarado em *params* com o “$” antes do inicio do nome.
duration	Define o tempo que demora para se atribuir um valor a um elemento ao executar uma ação com o *eventType*=“attribution”. O valor é um número real ou um parâmetro previamente declarado em *params* com o “$” antes do início do nome.

continua na próxima página

Tabela A.3: Parâmetros de um connector

Parâmetro	**Valor**
by	Numa ação este atributo especifica de modo que a alteração do valor irá ocorrer na atribuição (*eventType*="attribution"). O valor é um número real ou um parâmetro previamente declarado em *params* com o "$" antes do início do nome.
offset	Propriedade usada em *assessment* que define um valor de compensação ao realizar as comparações do *assessment*. O valor do mesmo é um número inteiro ou um parâmetro previamente declarado em *params* com o "$" antes do início do nome.
attributeType	Usada em *assessment* para especificar o tipo de atributo varia de acordo com o *eventType*. Os seus valores possíveis que são: "nodeProperty", que define se uma propriedade do nó será avaliada; "occurrences", que especifica como será testado quantas vezes o evento ocorreu; "repetition", que denota como será analisado o número de repetições do evento;"state", indicando qual o estado do evento será analisado . Se o *eventType*="presentation" então o *attributeType* pode assumir os valores "occurrences", "repetitions" ou "state". Se *eventType*="selection" o *attributeType* é opcional e pode conter o valor "occurrences" ou "state". Com eventType="attribution" o *attributeType* também é opcional e pode assumir os todos os seus valores possíveis.

Podem ser vistos alguns exemplos de conectores na Listagem A.13.

Listagem A.13: Exemplos de conectores

```
onBeginStartN {connector:{
        id:"onBegin1StartN",
        expression:"onBegin then start with max='unbounded'"
}}

onKeySelectionStopNStartN {connector:{
        id:"onKeySelectionStopNStartN",
        params:["key"] ,
        expression:"onSelection with key='$key' then stop with max='unbounded' || start
            with max='unbounded'"
}}

onEndAttributionTestStopNStartN {connector:{
        id:"onEndAttributionTestStopNStartN",
```

```
14          params:["value"] ,
15          expression:"onEndAttribution and
16                          test with eventType='attribution' , attributeType='nodeProperty'
                                == '$value' then
17                          stop with max='unbounded' || start with max='unbounded'"
18  }}
```

Na Listagem A.13O primeiro conector serve para iniciar um numero ilimitados de nós assim que nó começar. O segundo serve para que após a seleção de um nó com uma tecla um número ilimitado de nós termine e um numero limitado de nós comece. Por fim o terceiro e mais complexo conector inicia-se após o fim de uma atribuição e testa se a propriedade de um nó é igual a um valor especificado por um parâmetro (sempre que um parâmetro é usado em um atributo ele deve ter um "$" e deve ter sido declarado no conector).

A.1.7 Include

Podem ser incluídos arquivos externos ao JNS através de elemento include, estes arquivos podem ser partes de cabeçalhos NCL ou um cabeçalho JNS externo como mostra a Listagem A.14.

Listagem A.14: Especificação do Include

```
1           {include:{
2           descriptorURI:"caminho ate um documento ncl com a base de descritores",
3           regionURI:"caminho ate um documento ncl com a base de regions",
4           connectorURI:"caminho ate um documento ncl com a base de conectores",
5           transitionURI:"caminho ate um documento ncl com a base de transition",
6           jnsURI:"caminho ate um documento jns com um header externo ",
7           ruleURI:"caminho ate um documento ncl com a base de regras",
8           documentURI:"caminho ate um documento ncl com bases externas",
9           alias: stringDoalias ,
10          baseId: stringDaBaseDeregiao , //opcional
11          region: stringDaregiao //opcional
12          }}
```

Os elementos *regionURI, descriptorURI, transitonURI, connectorURI, ruleURI* ou um *documentURI* definem um caminho até um base externa NCL do tipo equivalente ao nome (região, descritores, transição, conectores, regras ou um documento completo), o alias é o nome usado como prefixo para cada elemento da base externa (todo elemento de uma base externa deve ser referenciado por *alias#Id*). Para o caso de uma base de regiões, o elemento *region* diz o nome de uma região que irá conter as regiões chamadas, fazendo que elas sejam filhas desta. No caso de um documento JNS, o cabeçalho externo é

inserido no documento e o documento funciona como se estivesse neste, ou seja, o atributo alias não é usado, e os elementos são referenciados pelo seu próprio *id*. O formato deste tipo de documento é exibido na Listagem A.15.

Listagem A.15: Exemplo de cabeçalho externo JNS

```
[
        {region:{?}},
        {descriptor:{?}},
        {rule:{?}},
        {descriptorSwitch:{?}},
        {connector:{?}},
        {transition :{?}},
        {include:{?}},
        {meta:{?}},
        {metadata: RDF tree } ,
]
```

A Listagem A.17 traz exemplos de importação de cabeçalhos, lembrando que para referenciar-se a um elemento importado de um documento NCL é necessário usar a nomenclatura *alias#id*, para documento JNS isso não é necessário.

Listagem A.16: Exemplos de include no JNS

```
[
        {include:{ descriptorURI:"descriptorBase.ncl",alias:"iDesBase"}},
        {include:{ regionURI:"regionBase.ncl",alias:"iRegBase"}},
        {include:{ connectorURI:"connectorBase.ncl",alias:"iConBase"}},
        {include:{ transitionURI:"transitionBase.ncl",alias:"iTranBase"}},
        {include:{ jnsURI:"head.jns"}},
        {include:{ ruleURI:"ruleBase.ncl",alias:"iRulBase"}},
        {include:{ documentURI:"documentBase.ncl",alias:"iDocBase"}},
]
```

A.1.8 Meta e metadata

A *meta* e *metadata* trazem meta dados que descrevem o conteúdo do documento. Na meta a chave do objeto JSON é o nome do conteúdo.

Listagem A.17: Especificação da meta e da metadata

```
        {meta:{
                "nome" : "conteudo"
        }},

        {metadata: "RDF tree" }
```

A *metadata* traz uma árvore no formato RDF[23]. No seu interior, este elemento é usado somente para descrever o conteúdo do documento.

A.2 Body

No corpo do documento se encontram os elementos que descrevem o documento: as mídias que serão exibidas; os elos de sincronismo; as variáveis para realizar as operações lógicas das regras; os *switches* e contextos que são os elementos de agregação. Os itens *meta* e *metadata* tem sua estrutura idêntica a usada no cabeçalho.

Listagem A.18: Estrutura do Body

```
body:[
        {id:"string"},
        {media:{?}},
        {link:{?}},
        {context:{?}},
        {switch:{?}},
        {property:{?},
        {meta:{?}},
        {metadata: RDF tree} ,
        {port: ?}
]
```

É possível se declarar um documento interativo em JNS usando apenas o corpo, pois as regras e conectores podem ser declarados, de modo implícito, no *switch* e elo.

A.2.1 Media

A mídia descreve o conteúdo que será exibido no documento multimídia. Também é o principal elemento do documento e faz uso de um descritor para especificar seus atributos ou os descreve internamente em suas propriedades. No JNS é possível se referenciar diretamente uma região, e o compilador irá criar um descritor vazio com a região referenciada. Ainda é possível criar a região dentro da mídia. Se houver a declaração de uma região e um descritor, a região referenciada pelo descritor irá sobrescrever a região declarada na mídia.

Listagem A.19: Especificação da mídia

```
        {media:{
        id:'stringLivre',
        src:'stringDePath',
        type :'stringDoTipo',
        descriptor: 'idDoDescritor',
        region:'idDaRegiao',
        region:'{?},
        refer:'idDeUmaMedia',
        instance:'string',
        anchors:[
```

```
11                  {area:{?}},
12                  {property :{?}}
13          ]
14          }}
```

O *id* é uma *string* única, o *src* define a localização da mídia, o *type* define o tipo da mídia que deve ser um dos da tabela A.4. O *refer* contém o *id* de uma mídia que esta usa como base herdando todos os atributos da mesma. O *instance* define se este elemento será executado com a mesma instância do nó que referencia ("*instSame*" ou "*gradSame*") ou se será tratado como um novo nó ("*new*").

Nos exemplos de mídias da Listagem A.20, são mostrados três tipos de mídias. A primeira é uma mídia de configurações, esta contém variáveis globais que serão usadas dentro do documento. A segunda é uma mídia que contém um arquivo "mp4" e declara a própria região, mesmo não sendo uma mídia de configurações ela tem uma propriedade, que serve para marcar o volume da mídia, além disso, possui um elemento *area* que marca um trecho da mídia. A terceira mídia, contém uma imagem "png" e referencia um descritor que deve conter a sua região.

Listagem A.20: Exemplos de mídias

```
1           {media:{id:"variaveis", type:"application/x-ginga-settings",
2                   anchors:[{property:{"idioma":null}}]}},
3           {media:{id:"video",src:"videoSemSom.mp4",
4                   region:{height:"100%",width:"100%",zIndex:1},
5                   anchors:[{property:{"soundLevel":"1"}},
6                           {area:{id:"ponto",begin:"1s",end:"3s"}}]}},
7           {media:{id:"SelIngles",src:"imIngles.png",descriptor:"dSelecao1"}},
```

Dentro da mídia existem elementos filhos chamados âncoras, que são as propriedades e áreas. Estes elementos são declarados dentro do elemento *anchors*, que é original do JNS, pois não existe filiação no JSON como no XML. Estes elementos serão explicados adiante em A.2.2 e A.2.3.

A.2.2 Property

As propriedades têm uma especificação similar ao da meta, tendo a *string* do nome da propriedade, seguido do seu valor. As propriedades não são restritas a mídia, podendo ser encontradas dentro de contextos e do corpo do documento.

Listagem A.21: Especificação da propriedade

```
1           {property:{
2                   "stringDoNome":"stringDovalor"
```

```
3         }}
```

Os valores possíveis a uma propriedade de uma mídia são: *top, left, bottom, right, width, height, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife, location, size* e *bounds*. Essas propriedades seguem as mesmas especificações das propriedades do descritor.

Listagem A.22: Exemplo de propriedade

```
1         {media:{id:"botao1",src:"botaoVemelo.png",region:"regiaoBotao",
2                 anchors:[{property:{"location":"0,0"}}]}},
3         {media:{id:"botao2",src:"botaoAzul.png",region:"regiaoBotao",
4                 anchors:[{property:{"location":"90%,0"}}]}},
5         {media:{id:"botao3",src:"botaoAmarelo.png",region:"regiaoBotao",
6                 anchors:[{property:{"location":"0,90%"}}]}},
7         {media:{id:"botao4",src:"botaoVerde.png",region:"regiaoBotao",
8                 anchors:[{property:{"location":"90%,90%"}}]}},
```

No exemplo da Listagem A.22 é possível ver uma propriedade sendo usada para especificar a localização de uma imagem na tela.

A.2.3 Area

Um elemento *area* marca um espaço dentro de uma mídia. Este espaço pode ser um trecho com início e fim, marcado pelo *begin* e *end* (marcando o tempo) ou *first* e *last* (marcando o número de amostras). Pode ser uma área da imagem, delimitada pelas coordenadas em *coords*, um texto (*text*) ou uma posição no texto (*position*).

Listagem A.23: Especificação da *area*

```
1         {area:{
2                 id:"string",
3                 coords:"coordenadas que definem poligono x1,y1,..xn,yn",
4                 begin:"float(s) segundos ou hh:mm:ss",
5                 end:"float(s) segundos ou hh:mm:ss",
6                 text:"string",
7                 position:inteiro, //PosicaoDoTexto
8                 first:"segundos(s) ou frames(f) ou NPT",
9                 last:"segundos(s) ou frames(f) ou NPT",
10                label:"stringDaLabel",
11                clip:"stringDoClip"
12        }}
```

NPT significa “normal play time”, que é o tempo normal de exibição. O “label” identifica uma cadeia de caracteres, como uma região marcada pela âncora, e *clip*, identifica

um trecho. A âncora deve ser referenciada pelo *id* para fins de eventos de sincronismo, que é a sua principal utilização.

Listagem A.24: Exemplo de *area*

```
{media:{id:"video",src:"videoSemSom.mp4",region:{height:"100%",width:"100%"},
        anchors:[
                {area:{id:"ponto",begin:"2s",end:"5.1s"}}
        ]
}}
```

Um exemplo de área pode ser visto na Listagem A.24. Neste exemplo no instante de 2 segundos, um evento chamado *ponto* é iniciado e no instante de 5.1 segundos, ele é terminado. Isso pode delimitar, por exemplo, a chamada com o nome do filme que está passando.

A.2.4 Port

A porta serve para indicar um mapeamento do contexto da porta para os contextos externos a ele (este modelo de contextos aninhados vem do "Nested Context Model"[37]). Ou seja, esta permite que elos referenciem eventos de mídias que estão dentro de um contexto diferente do contexto do elo. Não é possível acessar nenhum elemento interno a um contexto, sem que se use uma porta. Caso um contexto seja iniciado sem se especificar uma porta, todas as portas internas ao mesmo serão ativadas. Por isso, a porta quando colocada no contexto principal (*body*) indica as mídias que serão iniciadas no inicio do documento.

Listagem A.25: Especificação da porta

```
{media:{id:"video",src:"videoSemSom.mp4",region:{height:"100%",width:"100%"},
        anchors:[
                {area:{id:"ponto",begin:"2s",end:"5.1s"}}
        ]
}}
```

Uma porta é um item que pode ser repetido, dentro de um corpo ou contexto, porém, é possível declarar múltiplas portas através de somente um elemento *port*. Para se pegar eventos vindos de um nó dentro de um contexto, e para ativar ações dentro de nós contidos em um contexto, através de um elo, é necessário especificar a porta que ligue estes nós, dessa forma, deve ser referenciar no elo o "contexto.porta" e na porta deve se referenciar o "nó.interface".

Listagem A.26: Exemplo de portas

```
body:[
        {port: {"inicio1":"c1.p1"}},
        {context:[
                {id:"c1"},
                {port:{"p1":"no1.i1","p2":"no2"}},
                ...
        ]}
]
```

No exemplo de portas da Listagem A.26, as portas "inicio1" e "inicio2" estão contidas dentro do corpo do documento e, por isso, marcam o nós que irá começar no início do documento: "c1.p1" e "n1". Dessa forma, o contexto "c1" é iniciado na porta "p1", fazendo com que somente o "no1" na interface "i1" seja iniciado e a porta "p2" não seja ativada. Não é possível se referenciar uma porta para o inicio do documento, pois no início do mesmo todas as portas serão ativadas, porém, se o documento for importado para um contexto de um novo documento, essas portas podem ser referenciadas.

A.2.5 Link

O elo especifica relações de sincronismo entre as mídias. Ele pode tanto referenciar um *conector*, como especificar a expressão que representa a relação. No caso de especificar a expressão, é necessário que a expressão tenha após cada ação e condição, a referência do nó que se deseja ativar, caso seja mais de um nó, eles devem ser separados por vírgula. Na Listagem A.27 estão exemplos elos, primeiro usando *conectores* e depois usando expressões. O primeiro inicia os nós "v2" e "sAi" após o inicio do nó "bVe". O segundo para os nós "cPrincipal" e "cPropaganda", ao se selecionar o nó "botaoV" com a tecla vermelha. O terceiro elo testa se a propriedade "languale" do nó "settings" contém o valor "ingles" após receber o evento de atribuição, se sim, ele para o nó "aPortugues" e começa o nó "aIngles". É interessante notar que no terceiro caso ao contrário do que ocorre em um *switch* o nó "settings" não precisa ser do tipo -"application/x-ginga-settings".

Listagem A.27: Exemplos de elos

```
{link:{ connector:"onBegin1StartN",
                        binds:[{"onBegin":"bVe"},{"start":"v2,sAi"}]
}}
...
{link:{
        expression:"onBegin bVe then start v2 , sAi"
}}
...
{link:{ connector:"onKeySelectionStopNStartN",
                        ,params:{key:"RED"},
                        binds:[
```

```
12                                      {"onSelection":"botaoV"},
13                                      {"stop":"cPrincipal"},
14                                      {"start":"cPropaganda"}
15                                  ]
16          }}
17          ...
18          {link:{
19                  expression:"onSelection botaoV with key='RED' then stop cPrincipal ||
                        start cPropaganda"
20          }}
21          ...
22          {link:{ connector:"onEndAttributionTestStopNStartN",
23
24                                  binds:[
25                                      {"onEndAttribution":"settings.languale"},
26                                      {"test":"setings.languale",
27                                              params:{"value":"ingles"}},
28                                      {"stop":"aPortugues"},{"start":"aIngles"}
29                                  ]
30          }}
31          ...
32          {link:{
33                  expression:"onEndAttribution setings.languale and
34                          settings.languale == 'ingles' then stop aPortugues ||
35                          start aIngles"
36          }}
```

Ao usar um *assessment* de atribuição a propriedade, não é necessário especificar os atributos *eventType* e *attributeType* como *attribution* e *nodeProperty*. Pois já são automaticamente colocados pelo compilador (visto que só há esta possibilidade). Um exemplo disso pode ser visto no último elo da Listagem A.27 onde se compara a propriedade *languale* do nó *settings* ao valor *ingles* (os valores precisam estar entre '). Repare que o elemento *params* pode ser usado tanto no elo, substituindo a variável em todo elo, quanto no *bind*, substituindo à variável somente naquele papel. A expressão usada nos elos segue o mesmo padrão das dos *conectores*, no entanto, precisam referenciar os nós que irão ser afetados pelos papeis(*roles*). Os detalhes desta expressão são explicados na Listagem A.28.

Listagem A.28: BNF da expressão do elo

```
1         <expression>::=<condList>"then"<actList>
2         <condList> ::= <assessment>|<condition>|
              "("<condList>")"<comparador>"("<condList>")"
3         <assessment>::= <attr><operador><attr>|<attr><operador><valor>
4         <attr> ::= "id" "with" <params> | "id.inteface"
5         <noRef> :=      "id.inteface"   |       "id"
6         <params> ::= "idref" "=" "string"| <params>" , "<params>
7         <operador> ::= "=="|"<"|">"|"!="|"<="|">="|"eq"|"lt"| "gt"|"ne"|"gte"|"lte"
8         <valor> ::= "string"|number
9         <condition> ::= <condRole> <target> | <condRole> <target> "with" <params>
```

```
10          <condRole> ::= "onAbort" | "onBegin"|"onBeginAttribution"|"onEndAttribution" |
                "onEnd"|"onPause"|"onResume" | "onSelection"|"String"
11          <target>        :=      <noRef> |       <target>        ,       <target>
12          <actList> :: = <action>| "("<action>")"<sincronis>"("<action>")"
13          <sincronis> ::= "||" | ";"
14          <action>        ::=     <actionRole>    <target>|<actionRole>   <target>
                "with"  <params>
15          <actionRole> ::= "abort" | "pause" | "resume" | "start"| "stop"| "string"
16          <comparador> ::= "and"|"or"
```

Para aproveitar a funcionalidade de *reuso* é necessário se utilizar de um *conector* declarado no cabeçalho(*head*). Deve se referenciar o *conector* com o atributo *connector* e preencher os campos *binds* e *params*. Com os nós e parâmetros, que serão usados pelo *conector*. É possível referenciar uma interface usando o formato "idDono.Interface" tanto na expressão como dentro do *bind*.

Listagem A.29: Especificação do elo

```
{link:{
        id:"stringLivre",
        connector: idDoConector ,
        binds:[objetoBind0,..,objetoBindN],
        params:[objetoParam0,...,objetoParamN] ,
        expression: "expressãoBNF"
}
```

No interior do *bind* é necessário especificar o *objetoBind*, que é composto pela tupla *string* do role e referencia do nó. A referência do nó pode ser: uma *string* referenciando um nó pelo seu *id*; ou "umNoId.interface"; ou ainda um conjunto destas estruturas separadas por uma vírgula ",". O parâmetro pode ser colocado no elo no elemento *params* ou dentro do *objetoBind*. A estrutura do parâmetro é especificada pelo *objetoParam*. O elemento *descriptor* referencia um descritor que será associado ao nó e deve ser colocado dentro do *objetoBind*. A estrutura dos objetos *Bind* e *Param* pode ser vista na Listagem A.30.

Listagem A.30: Especificação dos objetos:*objetoParam* e *objetoBind*

```
objetoBind
{ "roleString":"idNo" }
{ "roleString":"idNo.inteface" }
{ "roleString":"idNo",params:objetoParam],descriptor:"idDescritor" }
{ "roleString":"idNo0,idNo1,idNo2.interface,...,idNoN"] }

objetoParam
{ "StringVariavel" : "valor" }
{ "StringVariavel0" : "valor0", "StringVariavel1" : "valor1", ...,"StringVariavelN" :
    "valorN" }
```

A.2.6 Context

Um contexto é um elemento que agrega outros elementos, nele está contido praticamente um novo corpo interno ao corpo do documento NCL. Todos os elementos internos a um contexto são inacessíveis fora dele, a não ser que exista uma porta no contexto referenciando este elemento. Igualmente os elos dentro de um contexto não têm acesso a elementos externos a ele, necessitando ativar portas do contexto para que as mesmas ativem relações externas ao contexto.

Listagem A.31: Especificação do contexto

```
1   {context:[
2           {id:"stringLivre"},
3           {refer:"idDoNó"},
4           {media:{?}},
5           {link:{?}},
6           {context:{?}},
7           {switch:{?}},
8           {meta:{?}},
9           {property:{?}}
10          {metadata: RDF tree } ,
11          {port: ?}
12  ]}
```

A estrutura do contexto pode ser vista na Listagem, que tem uma forma semelhante ao corpo do documento NCL(*Body*), contendo um *array* interno que contém os elementos internos. O elemento *refer* serve para o contexto referenciar a outro contexto, herdando do mesmo toda a estrutura interna, o conteúdo do *refer* é o *id* de outro contexto. O *refer* também pode ser usado para herdar um corpo de um documento NCL externo.

Listagem A.32: Exemplo de contexto

```
1   {context:[
2           {media:{id:"fundo",src:"fundo.png",
3                   region:{"height":"100%",width:"100%",zIndex:1}}},
4           {port:{"pFundo":"fundo"}},
5           {media:{id:"selecao1",src:"switch.png",descriptor:"dSelecao1"}},
6           {port:{"pS1":"selecao1"}},
7           {media:{id:"selecao2",src:"conector.png",descriptor:"dSelecao2"}},
8           {port:{"pS2":"selecao2"}},
9           {media:{id:"selecao3",src:"descritor.png",descriptor:"dSelecao3"}},
10          {port:{"pS3":"selecao3"}},
11  ]}
```

O exemplo da Listagem A.32 mostra um contexto que guarda um menu de seleção, verifique que existe uma porta para cada elemento do contexto, de modo que se o mesmo for executado todos os elementos serão executados. Entretanto, se ao receber a instrução

de execução, for referenciada uma porta específica do contexto, somente o nó referenciado pela porta será executado.

A.2.7 Switch

O *switch* é um elemento responsável por selecionar elementos para exibição, baseado em regras previamente declaradas no cabeçalho. Em JNS é possível declarar as regras diretamente dentro do *switch*. As regras declaradas, dentro do *switch*, seguem o mesmo padrão das regras do *descriptorSwitch* e da *rule*. Usando as variáveis declaradas dentro do documento NCL, por uma propriedade mídia do tipo "application/x-ginga-settings".

Listagem A.33: Especificação do switch

```
1        {switch:{
2                id:"stringLivre",
3                refer:'idDeReferencia',
4                switchPort:[{"idPorta0":
5                        ["componente0.interface0",...]},...,
6                        {"idPortaN":["componente0.interface0",...]}
7                ],
8                switchPort:{"idPorta0":
9                        ["componente0.interface0",...,"componenteN.interfaceN"]
10               },
11               vars:[{"stringDaVarNaRule":"idDaVariavel"}  ,..],
12               "expressao":{media:{?}}|{context{?}| "idDeUmNó"  ,
13               "idDaRule":{media:{?}}|{context{?}| "idDeUmNó"  ,
14               default: {media:{?}}|{context{?}| "idDeUmNó"
15       }}
```

O elemento *vars* será usado para ligar as variáveis temporárias usadas em uma regra a uma variável existente no documento. O *default* é a regra que deverá ser executada caso nenhuma das outras seja válida. Qualquer regra pode conter um nó (mídia ou contexto) ou um id de um nó já usado na regra. O *switchPort* define que determinados nós irão ser executados através de certas interfaces, ele pode ser um *arrays* de objetos ou somente um objeto. Em ambos os casos, as portas definidas irão conter *arrays* de mapeamento do *switch* com *strings* na forma "ipComponente.IdInterface" ou somente "IdComponenete". Na Listagem A.34, segue um exemplo de *switch* para um áudio em diferentes idiomas.

Listagem A.34: Exemplo de switch

```
1 {ncl:{
2        head[
3                {rule:{id:"rEn", vars:["val"], expression:"val == en" }},
4        ],
5        body:[
6                {media:{id:"variaveis" type:"application/x-ginga-settings" ,
```

```
7                   anchors:[property:{"i":null}]}},
8                 {switch:{
9                         id:"sAi",
10                        vars:[{"val":"i"}],
11                        "rEn":{media:{id:"aE", src:"En.mp3""}},
12                        "i == pt":{media:{id:"aP", src:"Pt.mp3"}}
13                        default:"aP"
14                }}
15        ]
16}}
```

O exemplo da Listagem A.34 usa uma regra declarada no cabeçalho do documento e uma regra interna. A regra declarada no cabeçalho, “rEn”, contém uma variável temporária, “vaI”, que é ligada a uma variável real, “idioma”, pelo atributo *vars* do *switch*. A regra “rEn”, testa se a variável é igual ao valor “en”. O *switch* também cria uma regra própria para checar se o valor de “idioma” é “pt”. Logo, o *switch* está checando se a variável “idioma” é “en” ou “pt”. Caso “idioma” seja “en” (ou seja a regra “rEn” é verdadeira) a mídia “audioEn” será executada. Caso “idioma” seja “pt”, a mídia “audioPt” será executada. A regra padrão é executada quando nenhuma das regras declaradas no *switch* é válida, e ela define que o “audioPt” será executado.

Tabela A.1: Parâmetros do descritor

Parâmetro	**Valor**
top, left, bottom, right, width, height	Número real na faixa 0 a 100 terminando com o caractere "%" (por exemplo, 30 %), ou um valor inteiro especificando o atributo em *pixels* (no caso de *weight* e *height*, um inteiro não negativo).
Location	Dois números separados por vírgula, cada um seguindo as regras de valor especificadas para parâmetros *left* e *top*, respectivamente.
Size	Dois valores separados por vírgula. Cada valor deve obrigatoriamente seguir as mesmas regras especificadas para parâmetros de *width* e *height*, respectivamente.
Bounds	Quatro valores separados por vírgula. Cada valor deve obrigatoriamente seguir as mesmas regras especificadas para parâmetros *left, top, width* e *height*, respectivamente.
Background	Nomes de cores reservadas: "white", "black", "silver", "gray", "red", "maroon", "fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua" ou "teal". O valor da cor de fundo pode também ter valor reservado "transparent", que é o valor padrão.
Visible	"true" ou "false". Quando não especificado, o atributo assume o valor "true".
Transparency	Um número real na faixa 0 a 1, ou um número real na faixa 0 a 100 terminando com o caractere "%" (por exemplo, 30%), especificando o grau de transparência de uma apresentação de objeto ("1" ou "100%" significa transparência total e "0" ou "0%" significa opaco).
Fit	"fill", "hidden", "meet", "meetBest", "slice".
Scroll	"none", "horizontal", "vertical", "both", ou "automatic".
Style	Localizador de um arquivo de folha de estilo.
soundLevel, balanceLevel, trebleLevel, bassLevel	Um número real na faixa [0, 1], ou um número real na faixa [0,100] terminando com o caractere "%" (por exemplo, 30%).
zIndex	Um número inteiro na faixa 0 a 255, sendo que regiões com maior valor de zIndex são posicionadas sobre regiões com menor valor de zIndex.
fontFamily	Uma lista priorizada de nomes de família de fontes e/ou nomes genéricos de famílias.
fontStyle	Estilo da fonte ("normal" ou "italic").
fontSize	Uma lista priorizada de nomes de família de fontes e/ou nomes genéricos de famílias.
fontVariant	Tamanho da fonte (inteiro).
fontWeight	Forma de exibição do texto: fonte em "small-caps" ou "normal".
fontColor	Cor da fonte ("white", "black", "silver", "gray", "red", "maroon", "fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua", ou "teal").
reusePlayer	Valor booleano: "false", "true". Valor default = "false".
playerLife	"keep", "close". Valor default = "close".

Tabela A.2: Tipos e Subtipos de transições possíveis

Tipo de transição	**Subtipo padrão**
barWipe	leftToRight
irisWipe	Rectangle
clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal
Fade	Crossfade

Tabela A.4: Tipos possíveis a uma mídia

Tipos de mídias MIME para formatadores Ginga-NCL[35]	**Extensão de arquivo**
text/HTML	html,html
text/plain	txt
text/css	css
text/XML	xml
image/bmp	bmp
image/png	png
image/gif	gif
image/jpeg	jpg, jpeg
audio/basic	wav
audio/mp3	mp3
audio/mp2	mp2
audio/mpeg	mpeg, mpg
audio/mpeg4	mp4, mpg4
video/mpeg	mpeg, mpg
application/x-ginga-NCL	ncl
application/x-ginga-NCLua	lua
application/x-ginga-NCLet	class, jar
application/x-ginga-settings	sem fonte (src)
application/x-ginga-time	sem fonte (src)

ANEXO A – Linguagem NCL

NCL é baseado no Modelo de Contextos Aninhados(NCL)[37] e oferece dois tipos de nós: nós de conteúdo e nós de contextos. Um nó de conteudo representa um ojeto mídia, por exemplo um audio, um vídeo, um texto ou mesmo um aplicação procedural, como um script Lua[17] ou um Java Xlet[20]. Um nó de conteudo, porem, não contém a mídia em si, ele somente representa a mídia dentro do documento. Um nó de contexto representam um grupo de nós, que podem ser nós de conteúdo ou nós de contexto, e um grupo de elos que representam relações entre os nós de componente.

Um documento NCL, assim comum uma pagina HTML, tem duas partes principais: o cabeçalho do documento (*head*) e o corpo do documento (*body*). O cabeçalho define, ou importa, regiões, descritores, regras, transições e conectores. O corpo usa essas definições pare especificar objetos de mídia e relações de sincronismo entre eles.

As regiões em NCL definem áreas da tela em que os objetos de mídia serão apresentados. Os descritores NCL são elementos que descrevem como o objeto de mídia será apresentado. As transições são usadas pelos descritores para indicar a animação que será feita quando um objeto inicia e/ou termina a apresentação. O descritor pode definir, por exemplo, o volume de um objeto áudio ou vídeo, a transparência da imagem ou mesmo a duração da mídia apresentada. O descritor também define a região onde o objeto mídia, usando aquele descritor, será apresentada

Note que a definição das regiões, transições e descritores são independentes do objeto de mídia que será apresentado. A definição destes elementos é feita em bases, definidas no cabeçalho do documento NCL, que são os a base de regiões (representada pelo elemento *regionBase*), base de transições (representada pelo elemento *transitionBase*) e base de descritores (representada pelo elemento *descriptorBase*). A listagem A.1 apresenta um exemplo.

Listagem A.1: Exemplo de base de regiões, transição e descritores

```
1 <regionBase>
```

```
2     <region id="reg1" top="10%" left="10%" width="80%" height="60%"/>
3  </regionBase>
4
5  <transitionBase>
6     <transition id="tr1" type="fade"/>
7  </transitionBase>
8
9  <descriptorBase>
10    <descriptor id="desc1" transIn="tr1" region="reg1" explicityDur="20s"/>
11 </descriptorBase>
```

A listagem A.1 apresenta uma região identificada por *reg1*, que define uma área de largura de 80% da largura da tela e 60% da algura da tela. A região é posicionada a 10% da largura da tela a esquerda e a 10% da altura da tela ao topo. A transição *tr1* define uma animação de desvanecimento. O descritor *desc1* usa a região *reg1* e usa a transição*tr1* quando o nó inicia a apresentação. O descritor também define uma duração fixa de 20 segundos ao objeto de mídia que o usa.

NCL permite a definição de relações de hipermídia genéricas, representada por conectores, que serão usados na definição dos elos representando relações especificas entre um conjunto de nós. Um conector NCL representa uma relação causal, onde uma condição, quadno satisfeita, dispara uma ação. Condições e ações definidas num conector são representadas por papeis. Conectores são definidos em uma base, o elemento *connectorBase*, também contida no cabeçalho do documento NCL.

Um exemplo de conector NCL e o conector *onBeginStart*. Este conector define dois papeis: *onBegin* e *start*. Este conector especifica uma relação que é "o inicio da apresentação de um nó, relacionado ao papel *onBegin*, causa o inicio da apresentação de um outro nó, relacionado ao papel *start*". Repare que este conector não especifica quais nós serão anexados a estes papeis, eles apenas definem o tipo da relação.

As regras em NCL definem expressões booleanas que testam valores de variáveis globais do documento NCL. Uma regra é usada dentro do documento na definição de elementos de controle (*switch*). Ela também é usada na definição de descritores alternativos que um nó pode usar(*descriptorSwitch*). Neste caso, o descritor será escolhido no momento que o nó for iniciado, dependendo do resultado da expressão booleana daquela regra.

Região, descritor, transição, regra e conector estão contidos em bases que podem ser definidas no cabeçalho do documento ou importados de outro documento NCL.

O nós de conteúdo NCL são definidos pelos elementos *media*. Um nó de conteúdo, ou

um nó de mídia, indica a localização do conteúdo de mídia, o seu tipo e o descritor NCL que especifica as suas características de apresentação. Um nó de mídia também define pontos de interface, que podem ser ancoras ou propriedades. Ancoras representam um subgrupo do conteúdo da mídia, e propriedades representam atributos do nó. A listagem **??** apresenta exemplo de nó de mídia em NCL.

Listagem A.2: Exemplo de nó de mídia

```
1 <media id="media1" src="videoclip.mpg" type="video/mpeg" descriptor="desc1">
2    <area id="track1" begin="2s" end="10s"/>
3 </media>
```

A listagem A.2 apresenta uma nó de mídia que representa o vídeo "videoclip.mpg". Esta mídia também define uma ancora (elemento *area*) que corresponde ao conteúdo do vídeo entre dois e dez segundo do vídeo.

Como foi explicado anteriormente, um nó de contexto, ou contexto, representa um grupo de nós e elos entre eles. Apesar disso, um contexto tem pontos de interface, representado por portas, que mapeiam nós internos ao contexto ou mesmo interfaces internas aos nós internos ao contexto. Vale lembrar que quando um contexto inicia sua apresentação, se não for definido um ponto de interface especifico, todos os pontos de interface irão ser iniciados. Como consequência, todos os nós de componente mapeados por eles serão iniciados, e todas as ações feitas a uma interface do contexto serão refletidas ao nó mapeado.

Um tipo especial de nó de contexto é o elemento *switch*. Ele associa as regras a nós de componentes. Quando um switch inicia a sua apresentação, ele apresenta o primeiro nó associado cuja a regra é verdadeira. Vale mencionar que somente um componente do switch será apresentado por vez.

Relações são especificadas por elos em NCL. Elos são contidos dentro dos contextos e só podem conectar nós contidos no mesmo contexto que eles. Para criar elos que se relacionam com elementos diferentes contexto, é necessário especificar portas que mapeiam estes nós. Um elo usa uma relação definida por um conector e indica os nós que participam desta relação. Os nós que irão participar da relação são definidos por um grupo de vinculos (elemento *bind*), onde cada vinculo anexa um nó (ou ponto de interface do nó) a um papel do conector. A listagem A.3 apresenta um exemplo de elo NCL.

Listagem A.3: Exemplo de elo NCL

```
1 <link xconnector="onBeginStart">
2    <bind component="media1" interface="track1" role="onBegin"/>
3    <bind component="media2" role="start"/>
```

```
4 </link>
```

A listagem A.3 apresenta um elo NCL que usa o conector previamente mencionado *onBeginStart* e indica o nó *media1*, na ancora *track1*, e o nó *media2* como os participantes desta relação. Isso significa que quando a apresentação do *track1* começar, a *media2* será iniciada.

Note que o corpo do documento é um nó de contexto, contendo um grupo de portas, nós e elos. No inicio de um documento o corpo do mesmo é iniciado fazendo com que todos os nós mapeados por portas sejam iniciados. Por isso é possível importar um documento externo para um contexto interno a um documento NCL.