Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Curso de Ciência da Computação

DANIEL DE SOUSA MORAES

LUA2NCL: *FRAMEWORK* PARA AUTORIA TEXTUAL DE APLICAÇÕES NCL USANDO LUA

São Luís
2016

DANIEL DE SOUSA MORAES

LUA2NCL: *FRAMEWORK* PARA AUTORIA TEXTUAL DE APLICAÇÕES NCL USANDO LUA

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profº Dr. Carlos de Salles Soares Neto

São Luís

2016

Moraes, Daniel de Sousa

Lua2NCL: *Framework* para autoria textual de aplicações NCL usando Lua/ Daniel de Sousa Moraes. – São Luís, 2016.

54 f.

Impresso por computador (Fotocópia).

Orientador: Profº Dr. Carlos de Salles Soares Neto

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2016.

1. Autoria Hipermídia 2. NCL 3. Lua 4. TV Digital

CDU 004.4:621.397.2

DANIEL DE SOUSA MORAES

LUA2NCL: *FRAMEWORK* PARA AUTORIA TEXTUAL DE APLICAÇÕES NCL USANDO LUA

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 07 de Abril de 2016

BANCA EXAMINADORA

Profº Dr. Carlos de Salles Soares Neto
(Orientador)
Universidade Federal do Maranhão

Profº Dr. Alexandre César Muniz de Oliveira
Universidade Federal do Maranhão

Profº Dr. Francisco José da Silva e Silva
Universidade Federal do Maranhão

MSc. Antonio José Grandson Busson
MediaBox Technologies

São Luís
2016

À minha família,
em especial aos meus pais,
e em memória de vovó Antonia.

Agradecimentos

Agradeço primeiramente a Deus por toda graça e misericórdia;

Aos meus pais, Francisco Ernane e Elienai, por todo o apoio e suporte, sem os quais não conseguiria esta realização;

Ao meu irmão Ernane, minha irmã Ana Beatriz e à todos os meus familiares, que sempre me ajudaram e apoiaram muito;

Ao meu orientador, Carlos de Salles, pela grande oportunidade concedida e por todo o apoio dado;

À todos os meus amigos que sempre me apoiaram e contribuiram direta ou indiretamente;

Aos meus companheiros e amigos do Laws que sempre me inspiraram a continuar e melhorar a cada dia;

Aos amigos e colegas de curso e à todos os professores que fizeram parte de toda essa trajetória.

“Gear up, We aren’t going on a windy walk here.”
(Contra-Terrorista, Counter-Strike: Global Offensive - Valve Corporation)

RESUMO

O desenvolvimento de aplicações para TV Digital no Brasil vem usando prioritariamente a linguagem declarativa NCL. Uma tecnologia brasileira bastante eficiente e poderosa que tem como uma de suas características um alto nível de abstração. Assim como outras linguagens XML, a NCL é de fácil aprendizado e leitura, o que facilita seu uso principalmente por pessoas ligadas diretamente à produção de conteúdos e não a programadores de aplicações. No entanto, o tamanho dos códigos descritos em NCL podem alcançar níveis que afetam até mesmo a legibilidade, o que também não atrai os programadores. O objetivo deste trabalho é propor o *framework* Lua2NCL para fornecer uma conjunto de recursos textuais que permitem a criação de aplicações para TV Digital sem o mesmo esforço dispensado à autoria em NCL e com a facilidade semelhante. Esses recursos oferecidos pelo Lua2NCL tem como foco central a redução na verbosidade dos documentos.

Palavras-chaves: Autoria Hipermídia. NCL. Lua. TV Digital.

ABSTRACT

The development of applications for Digital TV in Brazil is primarily using declarative language NCL. A very efficient and powerful Brazilian technology that has as one of its features a high level of abstraction. Like other XML languages, NCL is easy to learn and reading, which facilitates its use mainly by people linked directly to the production of content and not to application developers. However, the size of the codes described in NCL can reach levels which affect the even the readability, which also does not attract developers. The aim of this paper is to propose the Lua2NCL framework to provide a set of textual features that enable the creation of applications for Digital TV without the same effort dispensed to the authorship in NCL and with similar ease. These features offered by Lua2NCL has as its central focus the reduction in the verbosity of the documents.

Keywords: Hypermedia Authoring. NCL. Lua. Digital TV.

Lista de ilustrações

Lista de listagens

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
BDTV	Basic Digital Television - perfil NCL Básico para TV Digital.
DTD	Document Type Definition
DTV	Digital Television
EDTV	Enhanced Digital Television - perfil NCL Avançado para TV Digital.
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JRE	Java Runtime Environment
NCL	Nested Context Language
NCM	Nested Context Model
SBTVD-T	Sistema Brasileiro de Televisão Digital Terrestre
SMIL	Synchronized Multimedia Integration Language
URL	Universal Resource Locator
XML	Extensible Markup Language
WYSIWYG	What You See Is What You Get
W3C	World-Wide Web Consortium

Sumário

1 Introdução

Aplicações hipermídia para TV Digital abrangem desde aplicações simples, sem relação semântica com o conteúdo em exibição pelo programa principal até programas não-lineares, em que todas as mídias estão sincronizadas entre si, inclusive as mídias do programa principal (SOARES; BARBOSA, 2009). Tais aplicações são usualmente desenvolvidas usando dois paradigmas de programação: o declarativo e não-declarativo.

Linguagens de programação declarativas são linguagens de alto nível de abstração, em geral, ligadas a um domínio específico. Nelas o autor só precisa especificar a tarefa a ser executada sem se preocupar como essa tarefa será executada pelo interpretador ou compilador da máquina de execução. As linguagens declarativas são geralmente baseadas em uma metalinguagem que as impõe regras para a criação de um documento.

Por outro lado, em linguagens não-declarativas, o programador possui maior poder sobre o código, devendo estabelecer todo o fluxo de controle e execução de seu programa informando cada passo a ser executado. Evidentemente, o paradigma imperativo tem maior popularidade entre os programadores, pois é usado por linguagens já consagradas como: C, C++, Java, etc.

Além disso, pode-se destacar também o público-alvo desses paradigmas na autoria de aplicações hipermídia e a forma que essas aplicações podem ser desenvolvidas. Tratando-se especificamente de TV Digital é possível distinguir dois públicos-alvo distintos: os programadores e os produtores de conteúdo (produtores de TV, *designers*, jornalistas). Os programadores são ligados à programação imperativa, textual. Já os produtores de conteúdo são alvo da programação declarativa e autoria visual.

No Sistema Brasileiro de TV Digital Terrestre (SBTVD-T), o Ginga (ABNT, 2007) é o *middleware* responsável por viabilizar a execução de aplicações hipermídia. Inicialmente, o Ginga permitia o uso dos dois paradigmas de programação já mencionados, através dos ambientes Ginga-NCL (SOARES; RODRIGUES; MORENO, 2007) e Ginga-J (FILHO; LEITE; BATISTA, 2007). O Ginga-NCL usa a linguagem NCL para dar suporte declarativo à criação de aplicações e é obrigatório no padrão. Já o Ginga-J é uma extensão desenvolvida para permitir que aplicações pudessem ser desenvolvidas proceduralmente por meio da linguagem Java, no entanto este módulo foi descontinuado por motivos técnicos relacionados aos custos de licenciamento da tecnologia Java.

A NCL é uma linguagem declarativa, baseada em XML. Ela foi pensada para ser de fácil entendimento e aprendizado, assim pessoas com pouco conhecimento em

linguagens de programação podem usá-la sem tanto esforço. Ela consegue atender a esses requisitos, porém como é uma aplicação XML, um código NCL pode ficar bastante extenso por causa dos padrões definidos pelo XML. Outro fator relevante para a grande verbosidade de NCL é o fato dela ser baseada no reúso de elementos. Isto é, NCL possui vários elementos que são usados apenas para facilitar e estruturar o reúso de certos elementos e não possuem influência direta na semântica da aplicação (SOARES; LIMA; NETO, 2010).

Visando prover uma solução voltada a programadores e que supra o problema da verbosidade apresentado, essa monografia propõe um *framework*, chamado Lua2NCL, para autoria textual de aplicações hipermídia usando a linguagem Lua. Não é o objetivo do Lua2NCL substituir a linguagem NCL, mas apenas permitir a criação de documentos NCL de forma textual e com a verbosidade reduzida.

1.1 Objetivos

O objetivo geral desta monografia é apresentar o *framework* Lua2NCL, criado para auxiliar programadores no desenvolvimento de aplicações para TV Digital com a verbosidade reduzida, quando comparada a de documentos NCL equivalentes. Os objetivos específicos são:

- Reduzir a verbosidade em documentos de aplicações para TV Digital e possivelmente com isso diminuir o tempo de desenvolvimento;
- Prover uma estrutura sintática alternativa ao NCL, não baseada em XML e que garanta a equivalência semântica da aplicação;
- Garantir a validação do código final NCL gerado integrando o *framework* com a ferramenta NCLValidator.

1.2 Organização do Trabalho

Este trabalho é organizado da seguinte maneira: o Capítulo 2 apresenta conceitos e tecnologias, como os conceitos de Hipermídia e de autoria hipermídia e visões gerais sobre as tecnologias utilizadas como base teórica e prática desta monografia. O Capítulo 3 apresenta o *framework* desenvolvido com suas principais características e uma comparação entre documentos escritos com a abordagem definida pelo *framework* e documentos NCL equivalentes. O Capítulo 4 expõe as conclusões sobre o trabalho apresentado e os trabalhos futuros.

2 Tecnologias Relacionadas

Neste capitulo são discutidos conceitos e tecnologias importantes para esta monografia, como os conceitos de hipermídia e autoria hipermídia.

Apresenta-se sucintamente a linguagem XML, utilizada para especificação da Linguagem NCL. A linguagem NCL, que é a tecnologia padrão no desenvolvimento de aplicações para o sistema brasileiro de TV Digital e, portanto, a linguagem alvo do *framework* Lua2NCL. A ferramenta NCLValidator utilizada para validação do código NCL gerado pelo *framework* e por fim, a Linguagem Lua que é a tecnologia usada no desenvolvimento do *framework* e como a linguagem de programação da abordagem apresentada.

2.1 Autoria Hipermídia

O conceito de hipermídia é algo definido há bastante tempo em (NELSON, 1965). Segundo ele, hipermídia é uma extensão lógica do termo hipertexto, onde gráficos, áudio, vídeo, texto simples e *hyperlinks* se entrelaçam para criar um meio geralmente não-linear de informações.

Sabendo disto, uma aplicação hipermídia pode ser vista como em (COSTA, 2010; LOWE, 1999), que a definem como um conjunto de informações distribuídas no tempo e espaço. Onde, cada aplicação, além do seu conteúdo (vídeo, áudio, texto, imagem etc.), contém a especificação da sincronização espaço-temporal das suas mídias. Logo, na autoria de aplicações desse tipo é necessário que haja, entre outras coisas, a possibilidade de especificar os relacionamentos de sincronização espacial e temporal entre seus componentes, assim como uma forma de representação estruturada que garanta interoperabilidade entre essas aplicações.

A concepção de aplicações hipermídia concentra-se no preenchimento dessas necessidades por meio de linguagens cujo foco se encontra na descrição de relacionamentos entre mídias, podendo assim dar origem a programas não-lineares (SANTOS; NETO, 2010). Essa não-linearidade é possível graças ao suporte que essas linguagens dão à sincronização espaço-temporal do texto, do áudio, do vídeo e de qualquer outro objeto de mídia que possa compor as aplicações.

As linguagens, ditas declarativas, usadas na autoria de documentos hipermídia, utilizam-se de meta linguagens padrões como o XML, que é detalhado em seção 2.2, para conseguirem oferecer esse suporte à especificação de documentos que contenham além dos relacionamentos convencionais de navegação (*hyperlinks*), alguns tipo

de relacionamentos de sincronização. Um exemplo de linguagem declarativa é dado na seção 2.3.

Segundo Santos e Neto (2010), por apresentarem uma natureza diferente das linguagens imperativas, as linguagens declarativas possibilitam o uso de diferentes técnicas de suporte à autoria, dentre as quais destacam-se duas: as com suporte a autoria visual e as que enfatizam a autoria textual.

2.1.1 Autoria Visual

A autoria visual utiliza-se de objetos gráficos para abstrair o uso das linguagens de programação visando simplificar o desenvolvimento de aplicações. O uso de tais abstrações influencia diretamente no tempo de desenvolvimento já que consegue reduzir etapas do raciocínio inferencial (STENNING; OBERLANDER, 1995). O usuário constroi uma aplicação sem ter contato algum com código fonte do documento, usando apenas as abstrações gráficas adotadas pela ferramenta. O paradigma WYSIWYG (*What You See Is What You Get*) (ADOBE, 2007) endossa esse tipo de abordagem, já que ele define que o conteúdo exibido durante o processo de desenvolvimento deve ser semelhante ao produto final gerado.

Em geral esse tipo de ferramenta é utilizada por profissionais que não têm interesse ou não possuem experiência em programação. Um exemplo de ferramenta que utiliza o recurso de autoria visual é o Cacuriá (DAMASCENO; GALABO; NETO, 2014). O Cacuriá é uma ferramenta para criação de conteúdos multimídia e hipermídia, chamados de objetos de aprendizagem, isto é, conteúdos educacionais interativos lineares ou não-lineares, por meio de abstrações chamadas de cenas, que podem conter mídias como vídeos, textos e imagens com sincronização espaço-temporais. O Cacuriá é destinado principalmente a professores que não possuem conhecimento ou prática em programação. A Figura 1 mostra a interface da ferramenta, com as abstrações usadas para auxiliar a autoria.

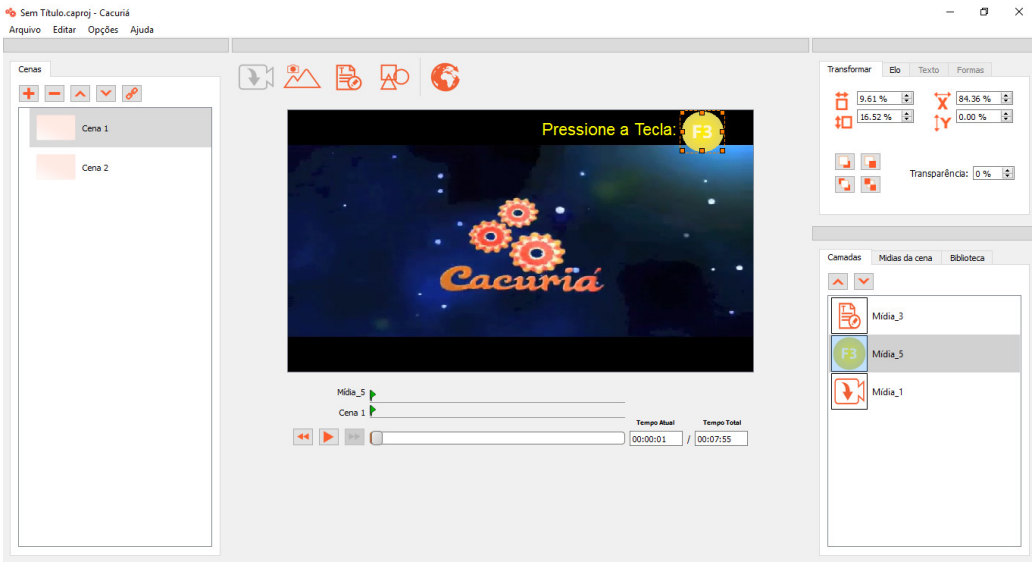


Figura 1 – Interface Cacuriá

No entanto, esse tipo de abordagem pode inserir alguns incovenientes à autoria. Há limitações no que as abstrações podem representar das linguagens de programção, exigindo muitas vezes que as ferramentas disponibilizem mais de uma abstração ao desenvolvedor, o que pode introduzir diferentes interpretações de suas abstrações em diferentes perfis de usuário.

2.1.2 Autoria Textual

A autoria textual pode ser vista como uma abordagem mais complexa, pois exige que o usuário tenha certo conhecimento da linguagem de programação. Por outro lado, ela também pode ser considerada mais fácil por manter um contato direto entre o código fonte e o desenvolvedor (AZEVEDO, 2008). Essa avaliação de quão fácil é a utilização de uma linguagem de programação deve, além de outros fatores, levar em consideração o público a quem esta é destinada.

Para programadores ainda iniciantes em uma linguagem de programação, a especificação de programas de forma textual pode ser um processo díficil. No entanto, muitas ferramentas voltadas a essa abordagem trazem funcionalidade que podem facilitar esse processo. Por exemplo, editores de texto com coloração de sintaxe, auto-completação de codigo, validação de código-fonte e indicação de erros sintáticos e semânticos (SANTOS; NETO, 2010).

Abordagens textuais são tipicamente utilizadas por pessoas com experiência e prática em programação e possuem conhecimento da linguagem e que se sentem limitados quando usando ferramentas visuais. Um exemplo de ferramenta que apoia a autoria textual de documentos hipermídia é o NCL Eclipse (AZEVEDO, 2008), um plugin

para a IDE Eclipse[1] projetado para auxiliar o desenvolvimento textual de aplicações NCL (AZEVEDO; TEIXEIRA; NETO, 2009). O NCL Eclipse disponibiliza funcionalidades, como as citadas acima, por meio do ambiente Eclipse e do NCL Validator, detalhado na seção 2.4. Ele provê também alguns recursos além da visão textual, como uma pré-visualização de mídias e regiões, que permite ao usuário ter uma idéia das dimensões da região declarada. A figura a seguir mostra a visão textual com uma pré-visualização de região ativa.

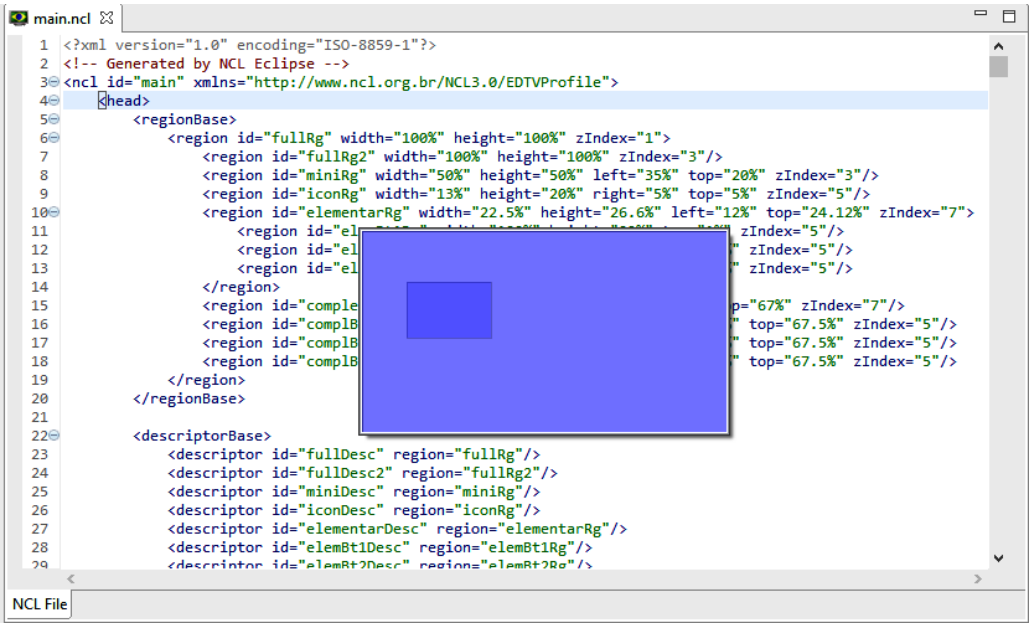


Figura 2 – Visão textual do plugin NCL Eclipse com pré-visualização de região

A principal desvantagem nessa abordagem é justamente a necessidade de um certo nível de conhecimento e prática, adquiridos geralmente apenas por programadores.

2.2 XML

O XML (*eXtensible Markup Language*) é um conjunto de regras usado para a definição de marcadores semânticos que dividem um documento em diferentes partes identificáveis. É uma metalinguagem de marcação que define a sintaxe usada para a definição de outras linguagens de marcação de domínio, estrutura e semântica específicos (HAROLD, 1999).

XML foi desenvolvido pelo XML Working Group, originalmente conhecido como SGML Editorial Review Board, formado com o patrocínio do World Wide Web Consortium (W3C) em 1996. Em 1998 tornou-se uma tecnologia recomendada pelo W3C

[1] Disponível em http://www.eclipse.org/

(BRAY et al., 1998; W3C et al., 2006). É uma versão simplificada do SGML (ISO, 1986). Segundo seus autores, o XML deve alcançar o poder e a generalidade oferecidos pelo SGML, mantendo a simplicidade do HTML (W3C et al., 1999) e seu principal objetivo é representar, genérica e estruturadamente, documentos que podem ser servidos, recebidos e processados na Web.

Ela possibilita o armazenamento e recuperação de dados em arquivos com certa facilidade por conta da indicação do conteúdo e da estruturação dos dados promovidos pelos marcadores XML. Porém, XML não fornece instruções de como o conteúdo deve ser apresentado. Os marcadores XML são bastante flexíveis quanto ao seu significado devendo seguir apenas alguns principios de organização (HAROLD, 1999). Se uma aplicação entende a meta sintaxe definida por XML para uma linguagem de marcação, ela automaticamente entende todas as linguagens construídas a partir dessa metalinguagem.

A Listagem 1 ilustra um exemplo de documento XML com informações sobre um canal de televisão e sua programação. Note que tanto o conteúdo da informação quanto os marcadores são dados em texto simples e claro. Na linha 1 tem-se o cabeçalho padrão com informações sobre a versão usada de XML e a codificação de caracteres do arquivo. A linha 2 inicia-se a especificação do elemento raiz `<canal>` que possui os atributos: `nome` com o valor `TV UFMA` e `numero` com o valor `54.1`. Na linha 3 está o elemento filho de `<canal>`, o elemento `<programa>`, que possui três atributos: o atributo `titulo` com seu valor e os atributos `inicio` e `fim`, também com valores atruibuidos a eles. Na linha 4 encontra-se o elemento filho de `<programa>`, o elemento `<descricao>`, que não possui atributos mas possui um conteúdo relacionado a si.

Listagem 1 – Exemplo documento XML.

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <canal nome="TV UFMA" numero="54.1">
3   <programa titulo="CAMINHOS DA REPORTAGEM" inicio="6:30" fim="7:25">
4     <descricao>O Caminhos da Reportagem leva o telespectador para uma viagem pelo
      mundo atrás de grandes histórias com uma visão diferente, instigante e complexa
      de cada um dos assuntos escolhidos. Temas atuais e polêmicos são tratados com
      profundidade e seriedade.</descricao>
5   </programa>
6   (...)
7 </canal>
```

Algumas linguagens Hipermídia que usam o XML como padrão e que possibilitam a especificação de documentos hipermídia com relacionamentos convencionais de navegação e relacionamentos de sincronização. Dentre elas estão o SMIL (HOSCHKA et al., 1998; W3C et al., 2005), também adotada como recomendação pelo W3C, e a NCL, que é detalhada na seção 2.3.

2.3 NCL

A linguagem NCL (*Nested Context Language*) é uma linguagem declarativa para autoria de documentos hipermídia, baseada em XML, desenvolvida no laboratório Telemídia. Ela usa como base de definição o modelo conceitual de hipermídia NCM (*Nested Context Model*) (SOARES; RODRIGUES, 2005), um modelo que permite a criação de documentos com sincronização espacial e temporal.

Em sua primeira versão NCL foi projetada para ser uma linguagem para descrição de documentos hipermídia na Web (ANTONACCI et al., 2000). Ela foi especificada por um DTD (*Document Type Definition*) XML e tinha como objetivo suprir a necessidade de uma linguagem padronizada sem as limitações identificadas em linguagens como o SMIL (HOSCHKA et al., 1998; W3C et al., 2005).

A partir de sua segunda versão, NCL passou a ter uma especificação modular em XML Schema e novas facilidades como: o tratamento de relações hipermídia como entidades de primeira classe, através da definição de conectores hipermídia e de bases de conectores e o uso de conectores hipermídia para a autoria de elos (SAADE, 2003; SAADE; SILVA; SOARES, 2003).

Na versão 3.0, a linguagem passa a ser utilizada pelo Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) (ABNT, 2011) como linguagem declarativa padrão do *middleware* Ginga (SOARES; RODRIGUES; MORENO, 2007; ABNT, 2007). E também como recomendação para sistemas de IPTV pela União Internacional de Telecomunicações (ITU) (ITU-T, 2009). NCL é usada por esses sistemas como linguagem para descrição de aplicações hipermídia para TV Digital. Nessa versão, além de profundas modificações no template de nós de composição e da reestruturação dos conectores hipermídia, são introduzidas novas funcionalidades, como: navegação por meio de teclas e funcionalidades de animação (SOARES; RODRIGUES, 2006).

Como uma aplicação XML, NCL faz uma clara separação entre os conteúdos multimídias e a estrutura de uma aplicação. Ela não restringe ou especifica os tipos de conteúdo dos objetos de mídia de uma aplicação multimídia, mas sim, apenas define como os objetos são estruturados e relacionam-se no espaço e tempo (SOARES; BARBOSA, 2009). O que é útil pois permite que mídias de diversos tipos possam ser relacionadas com um mesmo documento NCL (ABNT, 2007).

As próximas seções trazem uma visão de como a linguagem é estruturada e detalham alguns conceitos e elementos mais importantes para esse trabalho.

2.3.1 Módulos e Perfis de NCL

A NCL foi especificada em módulos, o que permite a combinação destes em perfis de linguagem. Segundo Soares e Barbosa (2009), módulos são conjuntos de elementos, atributos e valores de atributos XML que possuem relação semântica e representam alguma funcionalidade. E um perfil de linguagem é a combinação de módulos. Cada perfil pode agrupar um subconjunto de módulos NCL, o que permite a criação de linguagens restritas a domínios específicos, como é o caso dos perfis da linguagem para TV Digital.

A versão NCL 3.0 possui 15 áreas funcionais divididas em módulos. Das quais, 14 áreas são utilizadas na definição dos perfis TVD básico e TVD Avançado. Esses perfis foram definidos para ajustar a linguagem às características do ambiente de televisão digital (SOARES; RODRIGUES, 2006; ABNT, 2011; ITU-T, 2009). A Tabela 1 apresenta as 14 áreas e seus módulos usados nos perfis de TV Digital.

Tabela 1 – Áreas Funcionais da NCL 3.0

Fonte: (SOARES; BARBOSA, 2009)

Áreas Funcionais	**Módulos**	**Elementos**
Structure	Structure	ncl
		head
		body
Layout	Layout	regionBase
		region
Components	media	media
	Context	context
Interfaces	MediaContentAnchor	area
	CompositeNodeInterface	port
	PropertyAnchor	property
	SwitchInterface	switchPort
		mapping
Presentation Specification	Descriptor	descriptor
		descriptorParam
		descriptorBase
Linking	Linking	bind
		bindParam
		linkParam
		link

Áreas Funcionais	**Módulos**	**Elementos**
Connectors	CausalConectorFunctionality agrupa funcionalidades dos módulos: ConnectorCausalExpression; ConnectorCommonPart; ConnectorAssessmentExpression; CausalConnector)	causalConnector
		connectorParam
		simpleCondition
		compoundCondition
		simpleAction
		compoundAction
		assessmentStatement
		attributeAssessment
		valueAssessment
		compoundStatement
	ConnectorBase	connectorBase
Presentation Control	TestRule	ruleBase
		rule
		compositeRule
	TestRuleUse	bindRule
	ContentControl	switch
		defaultComponent
	DescriptorControl	descriptorSwitch
		defaultDescriptor
Timing	Timing	
Reuse	Import	importBase
		importDocumentBase
		importNCL
	EntityReuse	
	ExtendedEntityReuse	
Navigational Key	KeyNavigation	
Animation	Animation	
Transition Effects	TransitionBase	transitionBase
	Transition	transition
Meta-Information	Metainformation	meta
		metadata

O detalhamento de todos os módulos e elementos não faz parte do escopo deste trabalho. No entanto, alguns elementos são abordados nas próximas seções.

Como já citado, um perfil NCL agrupa um subconjunto de módulos de acordo com a necessidade do usuário. O perfil NCL 3.0 é o perfil completo, ele compreende todos os módulos NCL fornecendo todas as facilidades para a autoria declarativa.

Há também um perfil da linguagem, com a mesma expressividade do perfil completo, definido com as facilidades mínimas de reuso da linguagem (LIMA et al., 2010). Esse perfil, denominado “Raw”, evita usar módulos que definem elementos apenas para facilitar o reuso. O perfil NCL 3.0 Raw inclui os seguintes módulos: *Structure, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, Linking, CausalConnectorFunctionality, ConstraintConnectorFunctionality, ConnectorBase, EntityReuse, ExtendedEntityReuse e Meta-Information*.

Os perfis definidos para Sistemas de TV Digital (SOARES; RODRIGUES, 2006) são:

- **NCL 3.0 DTV Avançado - EDTV**

  Inclui os módulos: *Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse, KeyNavigation, Animation, TransitionBase, Transition e Meta-Information*.

- **NCL 3.0 DTV Básico - BDTV**

  Inclui os módulos: *Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse e KeyNavigation*.

- **NCL 3.0 CausalConnector**

  Inclui os módulos: *Structure, CausaConnectorFunctionality, ConnectorBase*.

A única diferença entre os perfis EDTV e BDTV é que os módulos de animação e metadados não são incluidos no BDTV. Porém, ambos são usados para desenvolver aplicações declarativas. O perfil CausalConnector permite a criação de conectores causais.

2.3.2 Conceitos Básicos

NCL usa como base o modelo NCM (SOARES; RODRIGUES, 2005), que define um conjunto de entidades para descrever os nós de mídias e seus relacionamentos de forma separada, abstraindo-se do conteúdo dessas mídias. Assim, os dois principais conceitos de NCL são os nós e os elos. Nós são os elementos que podem se relacionar e interagir entre si. Eles podem ser nó de mídia (`<media>`) ou de composição (`<context>`, `<switch>`).

Os elos são os elementos que definem os relacionamentos entre os nós usando conectores. Um conector define a semântica de um relacionamento. Definindo papéis a serem exercidos pelos nós que serão associados por meio do elo. O elo por sua vez, funciona como uma cola que liga o nó de mídia desejado ao papel definido pelo conector. Um elo define o relacionamento de sincronismo propriamente dito.

Para fazer a associação das interfaces dos nós aos papéis definidos pelo conector, o elo usa o elemento `<bind>` (ligação). A Figura 3 abaixo ilustra o esquema de um elo básico.

```
<link xconnector="id_do_conector">
   <bind role="id_de_papel_de_condicao"
                          component="id_de_um_objeto"
                          interface="id_de_uma_interface" />
   <bind role="id_de_papel_de_acao" component="id_de_um_objeto"
                          interface="id_de_uma_interface" />
</link>
```

Figura 3 – Esquema de um elo.
Fonte: (SOARES; BARBOSA, 2009)

2.3.3 Estrutura Básica de um Documento NCL

A estrutura básica de um documento NCL é formada pelo elemento `<ncl>` e por seus filhos `<head>` (cabeçalho) e `<body>` (corpo).

O elemento `<head>` contém bases de elementos referenciados pelo corpo da aplicação, como as regiões, descritores, os conectores. No `<head>` também define-se documentos que serão utilizados pelo documento atual.

O elemento `<body>` contém os elementos relacionados ao conteúdo da aplicação, tais como objetos de mídia, contextos e elos. A Figura 4 mostra uma tabela dos elementos com seus atributos e elementos filhos.

Como convenção, para os elementos filhos utiliza-se uma “interrogação” para indicar que o elemento é opcional (pode não existir ou ter uma ocorrência), um “asterisco” indica que o elemento pode ocorrer zero ou mais vezes. Os atributos de um elemento que são obrigatórios são sublinhados.

Elementos	Atributos	Conteúdo
ncl	*id*, *title*, *xmlns*	(head?, body?)
head		(importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*)
body	*id*	(port | property | media | context | switch | link | meta | metadata)*

Figura 4 – Tabela de Elementos, Atributos e Conteúdo (elementos filhos) de um documento NCL no perfil EDTV.
Fonte: (SOARES; BARBOSA, 2009)

A recomendação é que os elementos de `<head>` sigam a ordem mostrada pela Figura 4, já os elementos do `<body>` podem ser definidos em qualquer ordem.

A listagem abaixo traz o exemplo de um documento NCL definindo uma aplicação com duas imagens como mídias que possuem uma propriedade chamada `explicitDur`, que define o tempo de duração de cada imagem com valor atribuído. A aplicação inicia exibindo uma imagem e ao final do tempo de duração desta, um elo será ativado iniciando a exibição da outra mídia imagem. E assim a aplicação estará em um loop entre as duas mídias, pois sempre que uma termina a exibição o elo inicia a exibição da outra. Um exemplo semelhante é usado também no Capítulo 3.

Listagem 2 – Exemplo de Estrutura de uma Aplicação NCL.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <ncl id="exemplo1" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
3    <head>
4      <regionBase>
5        <region height="100%" width="100%" id="reg1" />
6      </regionBase>
7      <descriptorBase>
8        <descriptor region="reg1" id="desc1" />
9      </descriptorBase>
10     <connectorBase>
11       <importBase documentURI="ConnectorBase.ncl" alias="con"/>
12     </connectorBase>
13   </head>
14   <body>
15     <media src="medias/img1.jpg" descriptor="desc1" id="img1" >
16       <property value="3s" name="explicitDur" />
17     </media>
18     <media src="medias/img2.jpg" descriptor="desc1" id="img2" >
19       <property value="3s" name="explicitDur" />
```

```
20      </media>
21      <port component="img1" id="p1" />
22      <link xconnector="con#onEndStartN">
23        <bind role="onEnd" component="img1"/>
24        <bind role="start" component="img2"/>
25      </link>
26      <link xconnector="con#onEndStartN">
27        <bind role="onEnd" component="img2"/>
28        <bind role="start" component="img1"/>
29      </link>
30    </body>
31  </ncl>
```

Como já mencionado, o documento inicia com o elemento `<ncl>` que contém os filhos `<head>` e `<body>`. No `<head>`, da linha 3 a 13, estão os elementos que serão referenciados pelos elementos do corpo. Na linha 5 uma região, na 8 um descritor e na 11 a importação de um documento com a base de conectores.

No elemento `<body>`, a partir da linha 14, tem-se o conteúdo da aplicação em si. Da linha 15 a 17 a definição da primeira mídia e da 18 a 20 a segunda mídia. Note que ambas possuem o atributo `<property name="explicitDur»` com o valor de 3 segundos atribuído. O que significa que a exibição delas dura exatamente 3 segundos. Na linha 21, o elemento `<port>` define a mídia img1 como a porta da aplicação, isto é, quando a aplicação é iniciada, a mídia img1 é iniciada instantaneamente. Logo a seguir, vê-se a definição dos elos, linhas 22 a 25 e 26 a 29. O que cada elo faz é, quando o tempo de execução de uma mídia termina, ele inicia a outra mídia imediatamente.

Esse exemplo mostra apenas alguns elementos básicos de uma aplicação NCL, mas é suficiente para o entendimento da estrutura de uma aplicação.

2.4 NCL Validator

O NCL Validator (ARAÚJO; AZEVEDO; NETO, 2008; AZEVEDO, 2008) é uma ferramenta desenvolvida especialmente para a validação de documentos hipermídia escritos em NCL. Os autores apresentam razões pelas quais apesar da linguagem ser baseada em XML, a validação de documentos que a utilizam não pode ser realizada com a abordagem XML Schema (W3C et al., 2003). Uma das razões é o fato de NCL possuir certas especificidades que não poderiam ser validadas com a abordagem citada. Por exemplo, segundo a terminologia XML, por meio do XML Schema é possível definir a cardinalidade dos elementos e seus filhos e os tipos de atributos. No entanto, não é possível validar semanticamente atributos que fazem referência a identificadores de tipos específicos de elementos, ou que referenciem elementos filhos do mesmo contexto em que se encontra um elo.

Segundo Araújo, Azevedo e Neto (2008), o NCL Validator possui um processo de validação sintática e semântica exclusivo para documentos NCL. Esse processo é dividido em quatro fases:

- *Validação Léxica e Sintática*: Como NCL é baseada em XML, o primeiro passo da validação é a estrutura léxica e sintática. No entanto, devido a particularidades da linguagem, já citadas, requere-se o uso de um esquema de validação particular ao requisitos específicos da linguagem.
- *Validação Estrutural*: Nessa etapa é verificado se todos os atributos presentes em um elemento são válidos, se todos os atributos obrigatórios estão presentes, se os filhos de um determinado elemento estão definidos na norma e se estão conforme a cardinalidade especificada.
- *Validação Contextual*: Essa fase consiste em verificar se um elemento referenciado existe, se é do tipo que o atributo "requer" e se está no mesmo contexto do elemento que o referencia.
- *Validação Semântica e de Referências*: Este passo envolve uma simulação da apresentação do documento e tem por função gerar alertas ao usuário caso sejam detectadas inconsistências do ponto-de-vista da apresentação do documento. Por exemplo, a existência de trechos do documento que não são alcançáveis por um elo.

A figura abaixo ilustra esse processo de validação que gera mensagens e alertas durante cada fase e disponibiliza-as ao final.

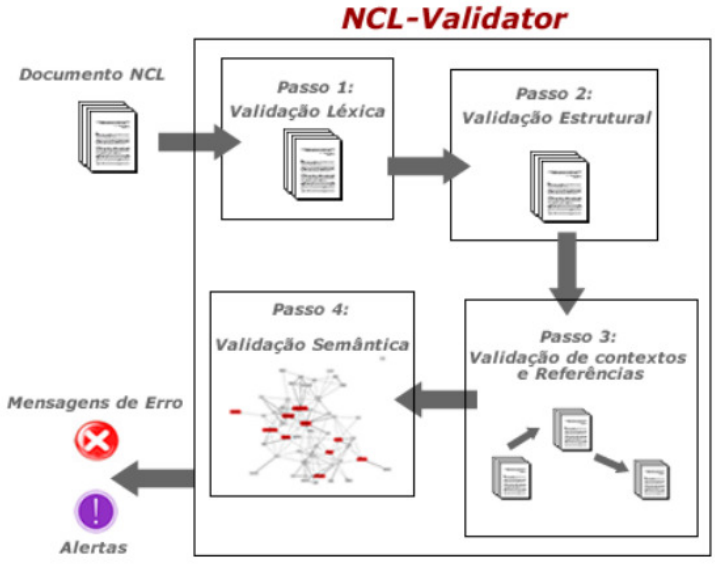


Figura 5 – Processo do NCL Validator
Fonte: (ARAÚJO; AZEVEDO; NETO, 2008)

O NCL Validator foi desenvolvida em Java (ORACLE, 2016). Usando a biblioteca XML padrão da linguagem para fazer o parser dos documentos. A implementação do NCL Validator também pode ser reutilizada por outros softwares. Com este objetivo foi disponibilizada uma API simples que permite chamar o processo de validação (passando um arquivo) e recuperar as mensagens de erros no final no processo. O trabalho apresentado nessa monografia utiliza a ferramenta NCL Validator para validar o código NCL final gerado, com o propósito de auxiliar os desenvolvedores.

2.5 Lua

Lua é uma linguagem procedural com poderosas facilidades para descrição de dados baseadas em tabelas associativas, projetada para ser usada como uma linguagem de extensão de propósito geral. Ela surgiu como a fusão de duas linguagens descritivas, projetadas para a configuração de duas aplicações específicas: uma para a entrada de dados científicos, o outro para a visualização de perfis de litologia obtidos a partir de sondas geológicas (IERUSALIMSCHY; FIGUEIREDO; FILHO, 1996). Lua nasceu no Tecgraf da PUC-Rio e atualmente é desenvolvida e mantida no laboratório LabLua também no Departamento de Informática da PUC-Rio.

Lua é uma linguagem dinâmica. Ideal para configuração, automação (*scripting*) e prototipagem rápida, justamente por oferecer características que facilitam essas atividades (LUA, 2016). Algumas dessas características (IERUSALIMSCHY, 2009) são:

- *Interpretação dinâmica*: significa que a linguagem pode executar trechos de código criados dinamicamente, no mesmo ambiente de execução do programa. Como exemplos dessa facilidade têm-se a função `loadstring` em Lua.
- *Tipagem dinâmica forte*: significa que a linguagem faz verificação de tipos em tempo de execução do programa e que jamais aplica uma operação a um tipo incorreto. Linguagens com tipagem dinâmica em geral não possuem declarações de tipos no código e não fazem verificação de tipos em tempo de compilação.
- *Gerência automática de memória dinâmica (coleta de lixo)*: significa que o desenvolvedor não precisa gerenciar memoria explicitamente na sua aplicação. Não há necessidade de um comando para liberar memória após seu uso.

Estas características são recorrentes a maioria das linguagens dinâmicas, porém Lua se destaca das demais por ser uma linguagem de *script*. Segundo, Ierusalimschy (2009), ela baseia-se em Tcl (OUSTERHOUT, 1990), uma linguagem estruturada como uma biblioteca C com uma API que permite que funções na linguagem possam chamar funções escritas em C e códigos em C chamem funções da linguagem. Lua ainda se

destaca entre as linguagens de *script* por sua simplicidade, portabilidade, economia de recursos e desempenho (IERUSALIMSCHY; FIGUEIREDO; CELES, 2007).

Lua pode ser usada não somente como uma linguagem completa, mas também como uma linguagem *framework* por ser capaz de abstrair estruturas específicas para qualquer domínio particular, além de incorporar as estruturas mais comuns das linguagens procedurais, como, estruturas de controle (*whiles, ifs, etc*), tarefas, sub-rotinas e operadores infixos (IERUSALIMSCHY; FIGUEIREDO; FILHO, 1996).

2.5.1 Tabelas Lua

Em Lua há apenas um método para a estruturação de dados, chamado tabela. Tabelas são estruturas que associam chaves com valores e permitem um rápido acesso ao valor associado a uma dada chave. Este conceito é conhecido como *array associativo* (IERUSALIMSCHY, 2009).

Tabelas podem ser usadas para representar estruturas de dados como *arrays*, listas, conjuntos e registros e até para implementação de conceitos mais abstratos como classes, objetos e módulos. Contudo, a tabela é muito mais poderosa que *arrays* e listas, simplificando muitos algoritmos à trivialidade com seu uso. Como exemplo, em Lua raramente é necessário escrever-se uma pesquisa já que as tabelas oferecem acesso direto a qualquer tipo de dado (IERUSALIMSCHY, 2006).

As tabelas possuem semântica bastante simples. Como mostra a listagem abaixo:

Listagem 3 – Exemplo de uso de tabela Lua.

```
1 t = {}
2 t[x] = 5
3 t[x]
```

Na linha 1, a expressão `t ={ }` cria uma tabela vazia e retorna a referência desta. Na expressão da linha 2, há um exemplo de atribuição, onde o valor 5 é associado à chave `x` na tabela referenciada por `t`. E a expressão `t[x]`, na última linha, retorna o valor associado à chave `x` na tabela `t`, ou o valor `nil`, caso a tabela não possua a chave dada. As chaves de uma tabela são as que possuem um valor associado diferente de `nil`.

As tabelas também apresentam sintaxe simples. Por exemplo, as notações `t.x` e `t[‘‘x’’]`, onde a string `x` é um campo indexado da tabela `t`, são equivalentes na manipulação de uma tabela como uma estrutura. No entanto, as expressões `t.end` e `t[‘‘end’’]` não são equivalentes, uma vez que a primeira expressão é invalida já que a palavra `end` é uma palavra reservada em Lua, o que não inválida a segunda expressão

já que `"end"` é uma string válida em Lua. A Listagem 4 apresenta o uso de algumas notações válidas:

Listagem 4 – Notações válidas em Lua.

```
1 t = {}
2 t["x"] = 5
3 t.y = 10
4 print(t.x, t["y"])    -- --> 5 10
```

2.5.2 Orientação a Objetos em Lua

A programação orientada a objetos em Lua tira vantagem da versatilidade das tabelas. Como os objetos, as tabelas possuem estados, possuem uma identidade (*self*) que independe dos seus valores. Por exemplo, dois objetos (tabelas) com o mesmo valor são objetos diferentes, enquanto que um objeto poder ter diferentes valores em momentos diferentes (IERUSALIMSCHY, 2006).

Lua implementa também o conceito de funções de primeira classe, isto é, as funções são como objetos e podem ser atribuídas à variáveis, passadas como parâmetros para outras funções, além de outras operações. E um objeto nada mais é que uma tabela, contendo campos com seus dados e operações. (IERUSALIMSCHY; FIGUEIREDO; FILHO, 1996)

O conceito de classes da orientação a objetos não existe em Lua, cada objeto define seu próprio comportamento e possui seu próprio formato. Portanto, não é difícil simular tal abstração usando tabelas como protótipos para os objetos. Um protótipo, chamado de metatabela em Lua, é um objeto comum criado exclusivamente para representar uma determinada classe, ou seja, é onde os demais objetos (suas instâncias) encontram qualquer operação que eles não conhecem, é o protótipo que controla o comportamento das instâncias. Protótipos são bastante simples de implementar em Lua. Por exemplo, para fazer que um objeto `b` se torne o protótipo de um objeto `a` basta usar comando ilustrado na Listagem 5

Listagem 5 – Comando para criação de protótipo.

```
1 b = {attr1 = "value", attr2 = "value"}
2 a = {}
3 setmetatable(a, {__index = b })
```

O que acontece após este comando é que o objeto `a` passará a buscar em `b` por qualquer operação ou atributo que ele desconheça. Isto é, o objeto `a` que foi declarado como uma tabela vazia, passa herda os atributos declarados no seu protótipo, que é a tabela `b`.

Nesse trabalho a terminologia Classe refere-se a um protótipo e objeto a uma tabela ligada àquela Classe.

2.5.3 NCLua

Lua é a linguagem adotada como linguagem de script do *middleware* Ginga (SOARES; RODRIGUES; MORENO, 2007; ABNT, 2007), por ser uma linguagem de fácil aprendizado, que combina sintaxe procedural com declarativa, com poucos comandos primitivos (SANT'ANNA et al., 2009). E principalmente por ter um alto grau de portabilidade, podendo ser executada com todas as suas funcionalidades em diversas plataformas, como computadores pessoais, celulares, etc.

A principal razão para incorporação de Lua ao padrão de TV Digital brasileiro é que quando uma aplicação necessita de funcionalidades não previstas pela linguagem declarativa, tais como processamento matemático, manipulação sobre textos, uso do canal de interatividade, animações, etc, a solução pode se tornar complicada ou até mesmo impossível (SANT'ANNA; CERQUEIRA; SOARES, 2008). Porém, com o suporte imperativo provido por Lua, o autor sempre que necessário pode usar seus recursos na criação de sua aplicação.

Como já visto, a linguagem NCL tem como característica a separação entre o conteúdo das mídias e como elas são estruturadas no documento NCL. Isto é, ela apenas referencia conteúdos de mídia sem ser capaz de exibi-los. Com isso, os *scripts* NCLua usam a mesma abstração para objetos de mídia usada para imagens, vídeos e demais mídias. Um NCLua deve ser escrito em um arquivo separado do documento NCL, que apenas o referencia (SANT'ANNA; CERQUEIRA; SOARES, 2008).

Para se adequar ao ambiente de TV Digital e se integrar à NCL, a linguagem Lua foi estendida com novas funcionalidades que permitem ao *script*, por exemplo, se comunicar com o documento NCL e saber quando seu elemento `<media>` correspondente foi iniciado por um elo. Permitem também que um NCLua responda a teclas do controle remoto, ou desenhe livremente dentro da região NCL a ele destinada. A principal diferença entre um NCLua e um Lua puro, é que o NCLua é controlado pelo documento NCL que está inserido (SANT'ANNA et al., 2009). Além da biblioteca padrão de Lua, o NCLua possui alguns módulos exclusivos, que são responsáveis pelas funcionalidades descritas acima. Estes são:

- *Módulo event*: permite a comunição entre o objeto NCLua o documento NCL e outras entidades externas, tais como controle remoto e canal de interatividade.
- *Módulo canvas*: oferece funcionalidades para desenhar objetos gráficos na região NCL referenciado pelo objeto NCLua.

- *Módulo settings*: oferece acesso às variáveis definidas no objeto settings do documento NCL (objeto do tipo “application/x-ncl-settings”).
- *Módulo persistent*: exporta uma tabela com variáveis persistentes entre execuções de objetos imperativos.

A especificação completa do módulos de NCLua foge ao escopo dessa monografia. A norma ABNT NBR 15606-2:2007 (ABNT, 2007) e H.761 (ITU-T, 2009) lista detalhadamente todas as funções suportadas por cada módulo.

3 Lua2NCL

Este capítulo apresenta a abordagem desenvolvida e detalha as principais contribuições mostrando também exemplos de utilização da proposta na autoria de documentos que expressam aplicações interativas de TV Digital.

3.1 Apresentação

Pensando-se em reduzir o máximo possível de esforço na autoria de documentos hipermídia em NCL por programadores não familiarizados com o paradigma declarativo, esse trabalho sugere uma abordagem com um *framework* de desenvolvimento, chamado de Lua2NCL (Lê-se *Lua to NCL*), em que o programador usa a linguagem imperativa (procedural) Lua na autoria de aplicações hipermídia. A linguagem Lua se mostra uma boa alternativa na construção de aplicações NCL, quando se trata de reduzir a verbosidade do documento, por ser uma linguagem poderosa, rápida, leve e bastante flexível (SANT'ANNA; CERQUEIRA; SOARES, 2008). E por já ser a linguagem de *script* usada para estender aplicações do padrão brasileiro de TV Digital.

NCL oferece um vasto conjunto de opções para a criação de um documento, tendo como uma de suas principais vantagens a possibilidade de reúso de alguns de seus elementos, como as regiões (SOARES; RODRIGUES, 2006). No entanto, mesmo permitindo reúso, a verbosidade implicada pelo XML penaliza aplicações mais simples, pois requer a declaração de muitos elementos para que a aplicação esteja de acordo com os padrões da linguagem, deixando o código NCL final bastante extenso (SOARES; LIMA; NETO, 2010). Por exemplo, para a declaração de regiões ou descritores é necessário que primeiro o desenvolvedor adicione os elementos `<regionBase>` e `<descriptorBase>` apenas para facilitar o reúso, mas que não influenciam diretamente na semântica da aplicação. Então, como no perfil NCL 3.0 Raw (LIMA et al., 2010), o Lua2NCL evita a declaração de tais elementos e obriga apenas a declaração dos elementos filhos, como as regiões e descritores.

Ao usar a abordagem do Lua2NCL como alternativa para a criação de um documento de aplicação NCL, certos elementos ficam implícitos para o desenvolvedor na autoria, reduzindo consideravelmente a verbosidade do documento. Isso porque o processador utilizado para a conversão do documento produzido em Lua para um documento final NCL é o responsável pela a inserção destes elementos no documento final de maneira correta. Por exemplo, a estrutura básica de um documento NCL é formada pelo elemento `<ncl>` e seus filhos `<head>` (cabeçalho) e `<body>` (corpo) como já dito na seção 2.3. Porém, o elemento `<ncl>` não é referenciado por outros elementos.

Ele é usado basicamente para indicar que o documento é um NCL. Portanto, com o Lua2NCL, uma vez que o tradutor sabe que o código desenvolvido deve ser traduzido para NCL, ele adiciona esse elemento automaticamente no código NCL final sem que o autor o tenha adicionado no documento escrito usando o Lua2NCL.

Ainda almejando a redução da verbosidade, elementos como `<head>` e `<body>` também podem ser abstraídos, já que suas utilizações são mais ligadas à organização do documento e não são referenciados por outros elementos. Portanto, com o Lua2NCL não é necessario explicitar o elemento `<body>`, já que sua função é apenas organizacional e não há nenhuma regra que restrinja a declaração dos seus elementos filhos. Por isso, o *framework* adiciona automaticamente esse elemento ao código NCL final. Já o elemento `<head>` possui uma ordem de declaração de seus elementos filhos que deve ser seguida, como citado na seção 2.3, e para que manter uma divisão entre os elementos referenciáveis e os de conteúdo, ele deve ser declarado explicitamente.

Além da redução de verbosidade promovida pela diferente representação dos elementos e da eliminação de alguns, por utilizar a linguagem Lua, o Lua2NCL possibilita a introdução de estruturas comuns às linguagen procedurais no processo de autoria, como os as estruturas de controle (*ifs, whiles, for*), o que pode reduzir ainda mais o número de elementos que o usuário declararia na aplicação.

As seções a seguir trazem mais detalhes sobre as mudanças promovidas, como usar o Lua2NCL e sobre como ele foi implementado.

3.2 Representação dos elementos NCL como tabelas Lua

Na abordagem do Lua2NCL, além da redução dos elementos já citados, as principais mudanças estão relacionadas a forma de representação dos elementos. A primeira mudança está relacionada diretamente ao uso da linguagem Lua na autoria de uma aplicação. Em Lua2NCL os elementos de NCL são representados como tabelas Lua. Estas tabelas desempenham o papel de classes, onde cada classe possui os atributos referentes ao elemento NCL que representa e alguns metódos usados pelo tradutor para a conversão de Lua para NCL. Por exemplo, para declarar uma região com o Lua2NCL, o desenvolvedor apenas instancia um novo objeto da classe região com os campos necessários. Os campos do objeto são os mesmos atributos que o elemento possui em NCL. A representação de um elemento NCL em forma de classe em Lua é ilustrada na Listagem 6 abaixo:

Listagem 6 – Definição da classe Region e do seu método construtor.

```
1 region = {  id = "",
2       left = "",
3       top = "",
```

```
4         right = "",
5         bottom = "",
6         width = "",
7         height = "",
8         zIndex = "",
9         title = ""
10       }
11 function region:new(o)
12   o = o or {}
13   setmetatable(o,self)
14   self.__index = self
15   if (o:analyse()) then
16     return o
17   end
18 end
```

Nesta listagem, a tabela `region`, linhas 1 a 10, tem o papel de protótipo da classe chamada de *region*. Ela estabelece todos os atributos pertencentes à classe *region*, que podem ou não ser instanciados em um objeto criado pelo desenvolvedor. Da linha 11 a 18 é definido o construtor da classe. O construtor entra em ação criando um novo objeto dessa classe utilizando a tabela `region`, declarada anteriormente, como o protótipo do novo objeto. O que significa que o novo objeto possui, por herança, os mesmos atributos definidos no protótipo *region*.

Utilizando essa metodologia é possível representar todos os elementos de NCL como classes contendo seus respectivos atributos e valores, bem como criar métodos que os validem. Para a instanciação de um objeto dessas classes, basta que o desenvolvedor utilize o método `new()` seguindo a notação da linguagem Lua, como no exemplo da Listagem 7 logo a seguir, onde uma nova região é instanciada utilizando alguns atributos necessários a uma região.

Listagem 7 – Exemplo de instancia da classe Region.

```
1 region:new{ id= "reg1", width = "100%", height = "100%", top="10%",left=
    "15%", zIndex="2"}
```

A representação da maioria dos elementos de NCL como classes em Lua segue o exemplo dado na Listagem 6. Como visto, um elemento que antes era declarado como uma *tag* em NCL agora é representado como um objeto (uma tabela), onde cada par (atributo=valor) do elemento corresponde a um campo da tabela que o representa. A única exceção se dá na representação do elemento elo que em Lua2NCL ganhou uma estrutura simplificada, a seção 3.3.

No entanto, o Lua2NCL não inclui os elementos da área funcional *Connectors*, com excessão do módulo *ConnectorBase*. Os módulos dessa área que não são inclusos

na Lua2NCL são usados na criação de conectores. Porém, o Lua2NCL sugere a utilização de uma base de conectores padrão que contempla um grande número de conectores.

3.2.1 Cobertura da abordagem Lua2NCL

Lua2NCL omite a declaração explícita de alguns elementos no intuito de reduzir a verbosidade do documento em edição e simplificar a autoria. Para melhor detalhar quais elementos de NCL podem ser omitidos ao usar a abordagem proposta pelo Lua2NCL, a tabela a seguir mostra os elementos que precisam ser representados explicitamente como tabelas Lua e os que já são inseridos automaticamente pelo gerador de código NCL contido no *framework*, podendo assim, ser omitidos.

Tabela 2 – Declaração dos elementos NCL no *framework* Lua2NCL

Elementos	**Tipo de declaração em Lua2NCL**
ncl	implícita
head	explícita
body	implícita
regionBase	implícita
region	explícita
media	explícita
context	explícita
area	explícita
port	explícita
property	explícita
switchPort	explícita*
mapping	explícita*
descriptor	explícita
descriptorParam	explícita
descriptorBase	implícita
bind	implícita
bindParam	explícita
linkParam	explícita
link	explícita
causalConnector	não possui
connectorParam	não possui
simpleCondition	não possui

Elementos	**Correspondente em Lua2NCL**
compoundCondition	não possui
simpleAction	não possui
compoundAction	não possui
assessmentStatement	não possui
attributeAssessment	não possui
valueAssessment	não possui
compoundStatement	não possui
connectorBase	explícita
ruleBase	implícita
rule	explícita*
compositeRule	explícita*
bindRule	explícita*
switch	explícita*
defaultComponent	explícita*
descriptorSwitch	explícita*
defaultDescriptor	explícita*
importBase	implícita
importDocumentBase	explícita
importNCL	explícita
transitionBase	implícita
transition	explícita*
meta	explícita*
metadata	explícita*

Os elementos com correspondência implícita em Lua2NCL são os que são adicionados automaticamente pelo gerador de código do *framework*. Os correspondentes explícitos são os que necessitam de uma declaração explícita no código. E os que não possuem correspondencia são os elementos não inclusos na abordagem.

Os elementos marcados com “asterisco” são os elementos não implementados pela versão atual do *framework*. Contudo, isso tem pouco impacto na utilização do mesmo, pois verificou-se que estes elementos não possuem um grau elevado de utilização em aplicações publicadas.

3.3 O elemento <link> como tabela Lua

Como visto na seção 2.3, que detalha a linguagem NCL, um elo é uma "cola" entre os papéis definidos pelos conectores e os elementos que desempenharão estes papéis. A Figura 3 mostra a estrutura do elemento <link> com seus elementos filhos `<bind>`, responsáveis por ligarem os papéis (*role*) aos elementos (*component*) desejados.

Em (SOARES; LIMA; NETO, 2010), os autores definem um nova sintaxe para os elos onde não é necessário referenciar um conector. O elo é escrito de uma forma mais simples, sem os elementos filhos `<bind>`. A representação de elos em Lua2NCL baseia-se nessa definição, com algumas modificações para ajustar à linguagem Lua.

A relação entre objetos em NCL definida pelos conectores é de causalidade, pode-se então dividir os papéis de um conector em condição e ação. Logo, o elo também pode ser dividido em duas partes: a que associa o objeto à condição e a que associa objeto à ação. Por isso, para a simplificação, a classe *link* segue essa divisão para definição dos seus atributos. A classe define apenas dois atributos: as tabelas `when` e `[''do'']` que representam a condição e ação dos conectores, respectivamente. A Listagem 8 ilustra a instanciação de um objeto da classe *link*.

Listagem 8 – Objeto da classe Link.

```
1 link:new{ when = { onEnd = {"img3"} }, ["do"] = { start = {"img1"} } }
```

Esse objeto da classe *link* evidencia o uso dos campos `when` e `do` para o mapeamento do relacionamento com os objetos. A leitura deste objeto pode ser feita da seguinte forma: *"when onEnd img3 do start img1"* (quando o elemento img3 terminar sua execução inicie o elemento img1). Neste caso, a condição é o evento de término (*onEnd*) do elemento `img3` e a ação a ser disparada é o inicio (*start*) do elemento `img1`.

Considerando a divisão do elemento elo em duas partes, um objeto pode ser visto da seguinte forma: a primeira parte de um objeto é composta da tabela `when`, que é a responsável pelo mapeamento da condição usada no *link*. Ela deve conter uma ou mais, subtabelas (caso a condição seja composta) nomeadas com os mesmos *id* do papéis de condição definido pelo conector usado. Por exemplo: `onBegin` ou `onEnd` como mostrado na listagem acima. Essa subtabela, que representa o papel de condição, deve conter o *id* do objeto associado a esse papel, equivalente ao atributo component da tag `<bind>` de NCL.

Na Listagem 8 a tabela `onEnd` contém apenas o *id* `img3`, logo a condição deste elo é: *"when onEnd img3..."* (quando o elemento 3 encerrar a execução...). A subtabela contida em `when` pode possuir outros elementos relacionados ao elemento que exercerá o papel e/ou ao próprio papel definido, como o par (interface=valor) e/ou o par

(parâmetro=valor), como ilustrado na Listagem 9 logo abaixo, onde a condição do *link* definido na linha 1 é: *"when onEnd interface explicitDur of img1..."* (quando a interface explicitDur do elemento img1 encerrar a execução...). Semelhantemente, a condição do *link* na linha 3 pode ser lida como: *"when onSelection interface area1 of img1 with keyCode Enter..."* (quando selecionara a interface area1 do elemento img1 com a tecla Enter...).

Listagem 9 – Objetos da classe Link com parâmetros.

```
1 link:new{ when = { onEnd = {"img1", interface="explicitDur"} }, ["do"] =
      { start = {"img2"} } }
2
3 link:new{ when = { onSelection = {"img1", interface="area1", keyCode="
     ENTER"} }, ["do"] = { start = {"img2"} } }
```

A segunda parte de um objeto *link* representa a ação ou ações (caso as ação seja composta) disparadas pela condição contida na primeira parte do objeto. A tabela `["do"]` ("do" deve ser usado entre colchetes e aspas por ser uma palavra reservada de Lua) é a responsável pelas ações com os devidos objetos. Similarmente à tabela `when`, a tabela `["do"]` deve conter uma ou mais subtabelas identificadas pelo *id* do papel de ação do conector usado. Por exemplo: `start`, `stop`. Essas subtabelas farão a associação do elemento desejado com o papel de ação, logo elas devem conter o *id* do elemento desejado, podendo também conter os campos de interface e/ou de parâmetro (que pode ser um *keyCode* ou *var*). A Listagem 10 a seguir mostra alguns exemplos que evidenciam como a tabela `["do"]` pode ser constituída.

Listagem 10 – Exemplos de uso da tabela de ação ["do"]

```
1 link:new{ when = { onEnd = {"img1"} }, ["do"] = { start = {"img3"} } }
2
3 link:new{ when = { onSelection = {"img1"} }, ["do"] = { start = {"img2"
     }, stop = {"img3"} } }
4
5 link:new{ when = { onSelection = {"img1"} }, ["do"] = { set = {"img2",
     interface="explicitDur", var="5s"} } }
```

A linha 1 mostra um exemplo simples onde apenas uma ação é disparada quando a condição é satisfeita. A tabela `["do"]` nesse *link* pode ser lida da seguinte forma: *"... do start img3"* (... inicie o elemento 3). E o *link* completo então ficaria: *"when onEnd img1 do start img3"* (quando o elemento img1 terminar a execução inicie o elemento img3). O segundo exemplo, na linha 3, mostra como seria o mesmo *link* disparando mais de uma ação. Note que a tabela `["do"]` agora contém duas subtabelas associando dois objetos a diferentes ações. Esse exemplo pode ser lido então da seguinte maneira: *"when onEnd img1 do start img2 and stop img3"* (quando o elemento img1 terminar a execução inicie o elemento img2 e o elemento img3). Na

linha 5, o exemplo mostra como os campos interface e de parâmetro, nesse caso o *var*, podem ser incorporados no elo. A leitura do elo completo pode ser: *"when onEnd img2 do set interface explicitDur of img3 as 5s"* (quando o elemento img2 terminar a execução atribua o valor 5s à interface explicitDur de img3).

3.4 Exemplo de aplicação com Lua2NCL

Para exemplificar o uso do Lua2NCL, a Listagem 11 mostra uma aplicação simples para TV Digital, um carrossel de imagens, onde uma imagem é apresentada por certo período de tempo e ao término deste há uma transição automática para a próxima imagem. Para efeitos práticos esse exemplo é nomeado de Carrossel. A Figura 6 mostra a visão estrutural do exemplo, gerada na ferramenta de autoria NCL Composer (GUIMARãES; COSTA; SOARES, 2007).

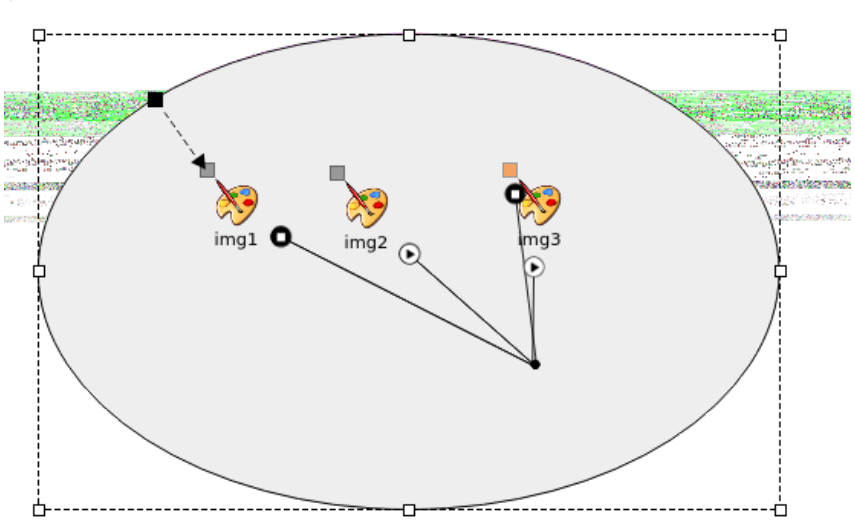


Figura 6 – Visão estrutural da Aplicação Carrossel

A aplicação inicia por uma porta ligada a mídia `img1`. As mídias `img1` e `img2` possuem, cada uma, uma propriedade explicitando sua duração e a mídia `img3` uma âncora definindo um certo intervalo de tempo. Essas propriedades e âncora são utilizadas como pontos de ativação e ação dos elos.

O código Lua utilizando a abordagem do Lua2NCL nessa aplicação Carrossel é ilustrado a seguir.

Listagem 11 – Exemplo de aplicação em Lua2NCL.

```
1 Lua2NCL = require("Lua2NCL/Lua2NCL")
2 head:new {
3     region:new{ id= "reg1", width = "100%", height = "100%"},
4     descriptor:new{ id ="desc1", region = "reg1"},
```

```
5     connectorBase:new{ documentURI= "ConnectorBase.ncl", alias= "con"}
6 }
7 media:new{ id ="img1", src = "medias/img1.jpg", descriptor="desc1",
      property:new{name="explicitDur", value= "3s"} }
8 media:new{ id ="img2", src = "medias/img2.jpg", descriptor="desc1",
      property:new{name="explicitDur", value= "3s"} }
9 media:new{ id ="img3", src = "medias/img3.png", descriptor="desc1", area
      :new{id="anchor1", begin="1s", ["end"]="3s"} }
10 port:new{ id = "p1", component = "img1"}
11 link:new{ when = { onEnd = {"img1"} }, ["do"] = { start = {"img2"} } }
12 link:new{ when = { onEnd = {"img2"} }, ["do"] = { start = {"img3"} } }
13 link:new{ when = { onEnd = {"img3"} }, ["do"] = { start = {"img1"} } }
14 Lua2NCL:Translate()
```

A primeira linha nesse código se remete à importação do processador responsável pela validação das tabelas e conversão para elementos NCL e pela construção do documento NCL correspondente. Mais detalhes sobre o processador são dados na seção 3.6. A função do Lua `require` é uma função para carga de módulos. Ela mantém uma lista de módulos já carregados e permite que o módulo carregado possa ser até mesmo renomeado pelo código que o carrega. Nesse caso, o módulo do processador é atribuído à variável Lua2NCL.

Da linha 2 a 6 na Listagem 11, tem-se a declaração da tabela de cabeçalho `head` representando o elemento `<head>`, que diferentemente do elemento `<body>` que não precisa ser declarado e é implicitamente inserido na conversão pelo tradutor, possui a declaração explícita em Lua2NCL. A tabela `head` deve conter os elementos que são reutilizados no corpo da aplicação, como regiões, descritores e switches, assim como em NCL. Porém, como dito anteriormente, alguns elementos do NCL são omitidos ao usar essa abordagem, por já serem incluídos automaticamente pelo tradutor. Por isso, ao declarar regiões e descritores não é necessário criar tabelas para os elementos `<regionBase>` e `<descriptorBase>`, basta declarar as regiões e descritores diretamente como nas linhas 3 e 4.

Na linha 5 a base de conectores utilizada como padrão em Lua2NCL é importada. A partir da linha 7 estão os elementos que em NCL são filhos do elemento `<body>`, porém em Lua2NCL podem ser declarados sem a criação do elemento raiz `<body>`. Mídias, portas, elos e contextos podem ser inseridos a partir desse ponto. No exemplo acima, vê-se nas linhas 7 a 9 a declaração de três mídias com alguns de seus campos (atributos) necessários. Cada campo de uma tabela, separados por uma “,” (vírgula), representa um atributo do elemento. Uma grande diferença se dá na declaração dos atributos *property* e *area* na tabela `media`. Em NCL, eles são declarados como *tags* separadas dos demais atributos como visto na seção 2.3, mas com a abordagem Lua2NCL, eles podem ser declarados como campos de tabelas.

A linha 10 da Listagem 11 mostra a declaração de um objeto porta, que deve obrigatoriamente possuir um campo *id* e referenciar um componente (mídia ou contexto), através do campo *component*, que será disparado no inicio da aplicação.

Logo a seguir, linhas 11 a 13 da Listagem 11, tem-se a declaração dos elos. Os três elos usam o mesmo conector da base, que pode ser inferido observando-se os campos das tabelas `when` e `[''do'']`. Neste caso, o campo da tabela `when` chama-se `onEnd` e recebe uma tabela com o *id* da mídia que desempenha o papel de condição. Já o índice da tabela `[''do'']` é chamado de `start` e também recebe uma tabela com o *id* da mídia escolhida. Logo, vê-se que se tratam de elos *onEndStart*, e genericamente podem ser lidos da seguinte forma: *"when onEnd objeto1 do start objeto2"* (quando objeto1 terminar a execução inicie o objeto2).

Note que na abordagem com tabelas Lua não é necessária a declaração do atributo *xconnector* como no NCL, pois o conector usado é selecionado automaticamente pelo *framework* a partir do *id* das subtabelas das tabelas de condição e ação (`when` e `[''do'']`).

Ainda na Listagem 11, na última linha, tem-se a chamada do método principal do *framework*. A partir do qual é feito o processo de conversão do código Lua para um documento NCL e a validação do mesmo.

3.5 Inclusão de Estruturas de Repetição

O uso da linguagem Lua no desenvolvimento de aplicações hipermídia não só torna o desenvolvimento mais próximo do paradigma imperativo como permite também a inclusão de recursos comuns a essas linguagens procedurais, como estruturas de repetição (*while, for, do*).

Em uma aplicação escrita com a linguagem NCL, que não possui essa estruturas de repetição, o programador é obrigado a declarar todos os nós de mídia mesmo que estas sejam do mesmo tipo e possuam apenas o atributo `src` diferentes. Isso poderia ser minimizado, usando uma estrutura como um `for`.

Utilizando o exemplo da seção anterior em um cenário onde o número de mídias da aplicação Carrossel, que no exemplo dado era apenas de 3 mídias, é um número N suficientemente grande, a definição de tais mídias torna-se um trabalho muito árduo, mesmo usando recursos de alguma ferramenta de autoria textual, como a completação automática. Uma solução para isso, é o uso de estruturas de repetição, tanto para criação das mídias como para descrição dos elos que definem o sincronismo entre essas mídias. A listagem abaixo ilustra como seria a definição das mídias e elos usando a estrutura `for`.

Listagem 12 – Exemplo do uso de estrutura de controle.

```
1 for i=1,N do
2   media:new{ id ="img"..i, src = "medias/img"..i..".jpg", descriptor="
      desc1", property:new{name="explicitDur", value= "3s"} }
3 end
4 for i=1,N-1 do
5   link:new{ when = { onEnd = {"img"..i} }, ["do"] = { start = {"img"..i
      +1} } }
6 end
7 link:new{ when = { onEnd = {"img"..N} }, ["do"] = { start = {"img1"} } }
```

Evidentemente, o uso de estruturas de controle não se limita a casos como o ilustrado acima, ficando a critério do desenvolvedores que no caso do Lua2NCL infere-se ser programadores com certo nível em linguagens procedurais.

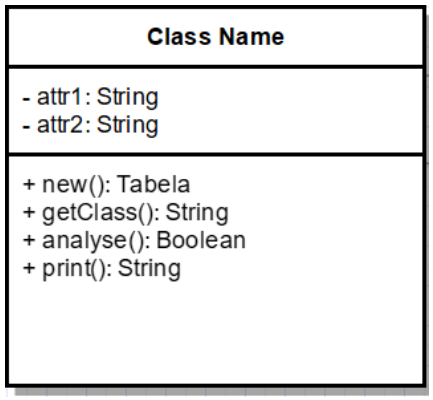
3.6 O Tradutor Lua2NCL

Para possibilitar a criação de aplicações para o middleware Ginga usando a abordagem de tabelas Lua, nomeada de Lua2NCL, desenvolveu-se um módulo responsável por traduzir códigos com tabelas Lua para NCL. O *framework* Lua2NCL foi desenvolvido usando a própria linguagem Lua, seguindo o paradigma de orientação a objetos.

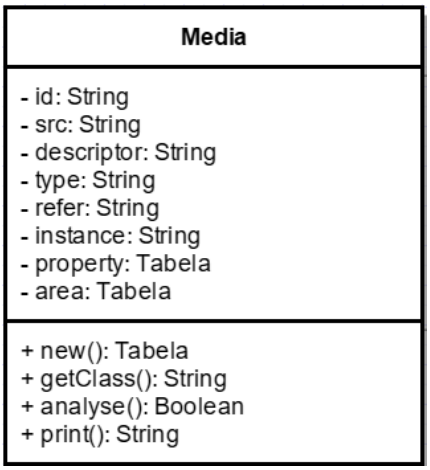
O Lua2NCL é constituído de classes representantes dos elementos de NCL e de uma classe principal chamada Lua2NCL, responsável pela criação, organização e validação do documento NCL gerado. O método mais importante da classe principal é o `Translate()`. Nele os objetos instanciados são transformados em elementos NCL e escritos em um documento NCL, que por fim é validado utilizando a ferramenta NCL Validator, apresentada na seção 2.4.

Ao inserir o *framework* em um código Lua, como um módulo e atribuí-lo a uma variável como na linha 1 do exemplo mostrado na Listagem 11, suas classes e métodos ficam armazenados e disponíveis para chamada pelo documento Lua a qualquer instante.

As classes que representam os elementos de NCL possuem métodos padronizados, como ilustrado na Figura 7a. Os atributos de cada classe variam de acordo com o elemento que a classe representa. A Figura 7b exemplifica uma classe do Lua2NCL, a classe *Media*.



(a) Modelo Genérico de Classe



(b) Exemplo de Classe: Classe Media

Figura 7 – Classes da Arquitetura Lua2NCL

As funções desempenhadas pelos métodos são as seguintes:

- `new()` – método construtor. É o responsável por criar uma nova estrutura de tabela. Porém essa estrutura só é criada e retornada se a chamada ao método `analyse()` retornar com valor verdadeiro.
- `getClass()` – método responsável por verificar a qual classe um objeto pertence. Retorna o nome da classe.
- `analyse()` – método responsável por verificar se os atributos declarados em um objeto pertencem à classe do objeto e se os valores desses atributos são válidos. Retorna um booleano, verdadeiro se os atributos e valores estiverem de acordo e falso caso contrário.
- `print()` – método para a impressão do objeto como um elemento NCL. Retorna uma string.

Ao instanciar um objeto, o desenvolvedor utiliza o método `new()` que cria uma nova estrutura de tabela para comportar o objeto. O construtor faz uma chamada ao método `analyse()` para verificar se os atributos do objeto declarado são válidos. Caso os atributos e valores estejam de acordo, o construtor retorna o objeto criado e o adiciona a uma lista de objetos instanciados. Porém, se houver algum atributo que não pertença àquele objeto ou se o valor de um atributo não está de acordo, mensagens de erros são mostradas e o objeto não é instanciado.

O diagrama na Figura 8, ilustra a sequência de execução na criação de um documento usando o *framework* Lua2NCL.

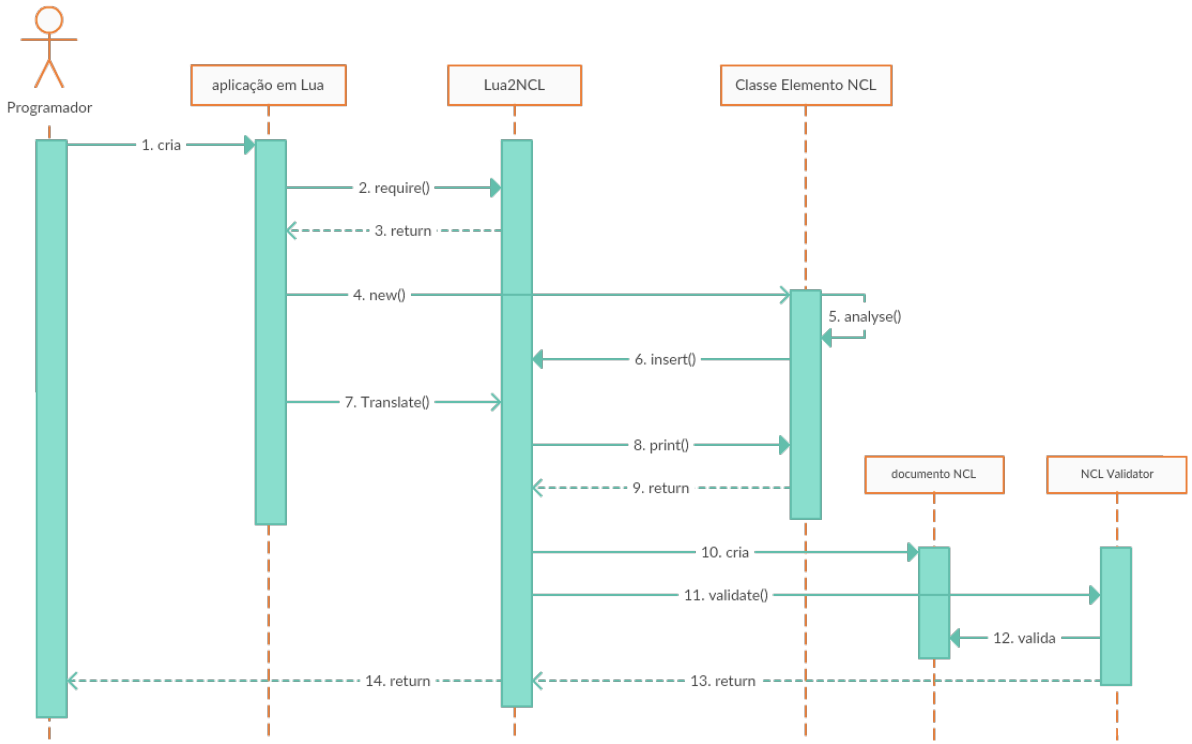


Figura 8 – Diagrama de Sequência do *framework* Lua2NCL

Ao inserir o Lua2NCL com a função `require()` do Lua, todas as funções e classes definidas pelo *framework* são retornadas e armazenadas em uma variável, como na linha 21 da Listagem 11 na seção seção 3.4. Isso é o que está descrito nos passos 1, 2 e 3 do diagrama.

Os passos 4, 5 e 6 ocorrem quando o desenvolvedor usa o método `new()`, que é o método construtor. Esse por sua vez, antes de retornar o objeto solicitado, faz uma chamada ao método `analyse()`, que faz uma verificação dos atributos do objeto declarado. Caso não haja erros na declaração desse objeto, o `analyse()` adiciona esse objeto a uma lista de objetos instanciados e retorna para o `new()`, que encerra sua execução retornando para a aplicação.

O passo 7 desencadeia a sequência de eventos que ocorrem para a conversão do documento em Lua para um documento NCL. Quando o método `Translate()` é acionado, ele percorre a lista de objetos instanciados, chamando para cada objeto o método `print()` correspondente. Esse retorna a estrutura daquele objeto em NCL. Ao final da lista, um documento NCL com as estruturas dos objetos é criado e validado com a ferramenta NCL Validator, que mostra mensagens de erro ou alertas caso existam. Essa é a sequências dos passos 7 à 14.

Como o Lua2NCL é baseado em tabelas Lua, a execução de um documento que especifica uma aplicação hipermídia continua usando o próprio interpretador padrão de Lua. A figura abaixo mostra a execução do exemplo dado na Listagem 11 e a

Listagem 13 mostra o código NCL equivalente da aplicação gerado pelo Lua2NCL.

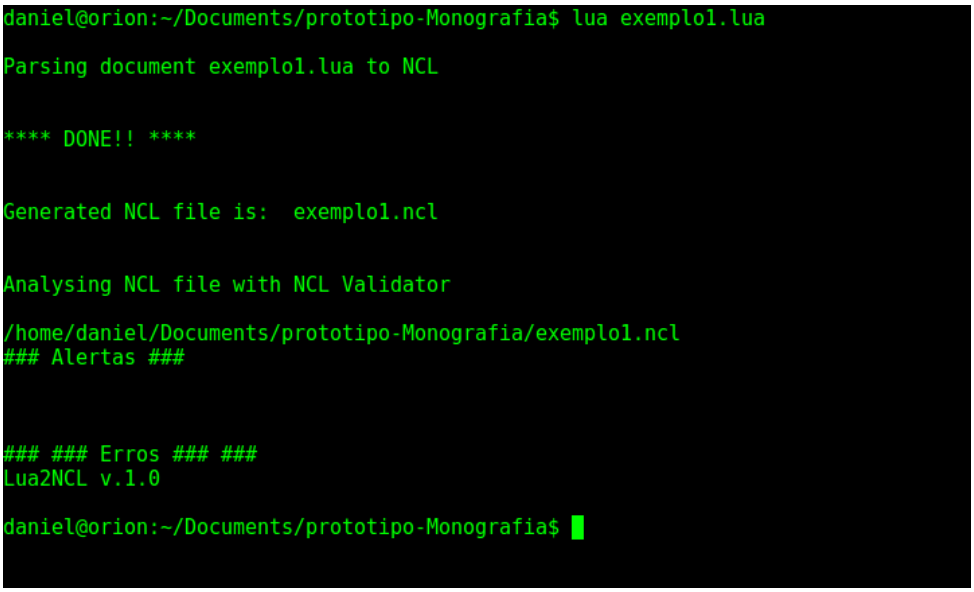


Figura 9 – Execução do exemplo Carrossel

Listagem 13 – Código NCL da aplicação Carrossel

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!--Generated by Lua2NCL v.1.0-->
3  <ncl id="exemplo1" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
4    <head>
5      <regionBase>
6        <region height="100%" width="100%" id="reg1" />
7      </regionBase>
8      <descriptorBase>
9        <descriptor region="reg1" id="desc1" />
10     </descriptorBase>
11     <connectorBase>
12       <importBase documentURI="ConnectorBase.ncl" alias="con"/>
13     </connectorBase>
14   </head>
15   <body>
16     <media src="medias/img1.jpg" descriptor="desc1" id="img1" >
17       <property value="3s" name="explicitDur" />
18     </media>
19     <media src="medias/img2.jpg" descriptor="desc1" id="img2" >
20       <property value="3s" name="explicitDur" />
21     </media>
22     <media src="medias/img3.png" descriptor="desc1" id="img3" >
23       <area begin="1s" end="3s" id="anchor1" />
24     </media>
25     <port component="img1" id="p1" />
26     <link xconnector="con#onEndStartN">
```

```
27        <bind role="onEnd" component="img1"/>
28        <bind role="start" component="img2"/>
29      </link>
30      <link xconnector="con#onEndStartN">
31        <bind role="onEnd" component="img2"/>
32        <bind role="start" component="img3"/>
33      </link>
34      <link xconnector="con#onEndStartN">
35        <bind role="onEnd" component="img3" interface="anchor1"/>
36        <bind role="start" component="img1"/>
37      </link>
38    </body>
39  </ncl>
```

Esse é o código NCL gerado pelo Lua2NCL equivalente ao código mostrado na Listagem 11. É possível notar uma diferença significativa no número de linhas de código. Uma comparação entre códigos NCL e código usando o Lua2NCL é feita na próxima seção.

O Lua2NCL pode ser usado em qualquer plataforma que possua a API da linguagem Lua e o ambiente de execução da tecnologia Java (ORACLE, 2016) instalados.

3.7 Comparação entre Lua2NCL e NCL

Como visto, o Lua2NCL promove algumas mudanças tanto na estrutura do código como na maneira de se programar uma aplicação hipermídia. Para mostrar o quanto essas mudanças afetam no código de uma aplicação, esta seção apresenta algumas comparações entre códigos usando a abordagem Lua2NCL e NCL puro. A aplicação escolhida é a aplicação utilizada como exemplo para explicar o uso da linguagem NCL no Capítulo 3 de (SOARES; BARBOSA, 2009). Essa aplicação chamada de "O Primeiro João" é construída em várias versões de forma iterativa, onde elementos e conceitos da linguagem são introduzidos à medida que são apresentados. Os códigos da aplicação podem ser encontrados no repositório ClubeNCL[1]. Escolheu-se então cinco das versões disponíveis e converteu-se para códigos que utilizam Lua2NCL. Os parâmetros usados nessa comparação são: número de linhas e número de elementos usados. A Tabela 3 mostra os valores desses parâmetros para as versões selecionadas da aplicação.

[1] Disponível em: http://clube.ncl.org.br/

Tabela 3 – Comparação do Primeiro João em NCL e Lua2NCL

	NCL		**Lua2NCL**	
Exemplos	**Nº de linhas**	**Nº de elementos**	**Nº de linhas**	**Nº de elementos**
1 - 01sync	49	33	23	20
2 - 02syncInt	84	61	41	34
3 - 07transition	108	79	66	49
4 - 10menu	218	167	151	101
5 - 11nclua	237	182	156	106

Observando a tabela, percebe-se que para os exemplos usados há uma redução de linhas miníma de 46% ocorrida no exemplo 1 e uma redução máxima de 69% no exemplo 4. O gráfico na Figura 10 ilustra melhor as diferenças.

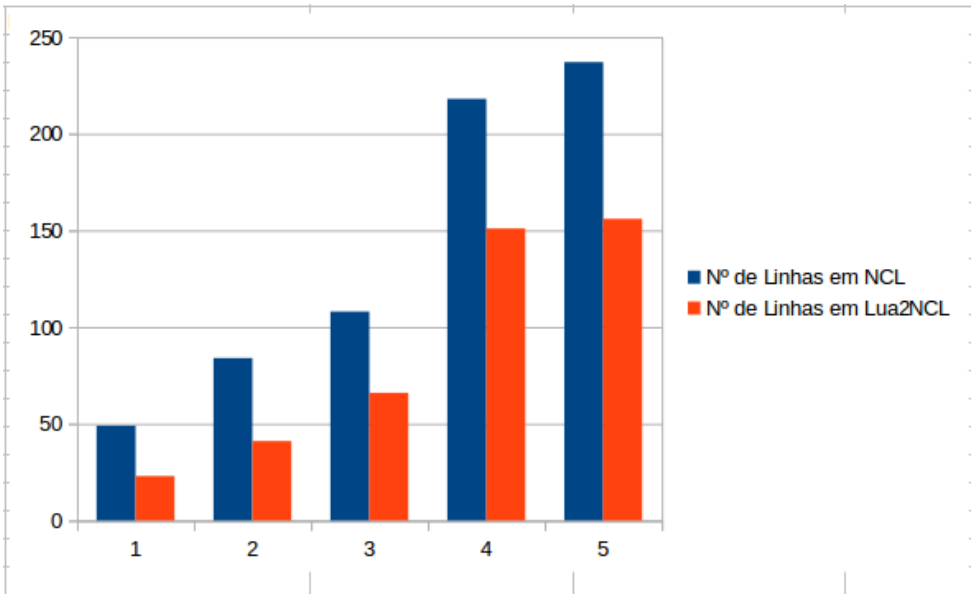


Figura 10 – Comparação do número de linhas em NCL e Lua2NCL

Em relação às estruturas usadas em cada documento, percebe-se uma redução miníma no exemplo 2 de 55% e a maior redução ocorre no exemplo 3, com 62%. A Figura 11 ilustra as diferenças no uso de elementos.

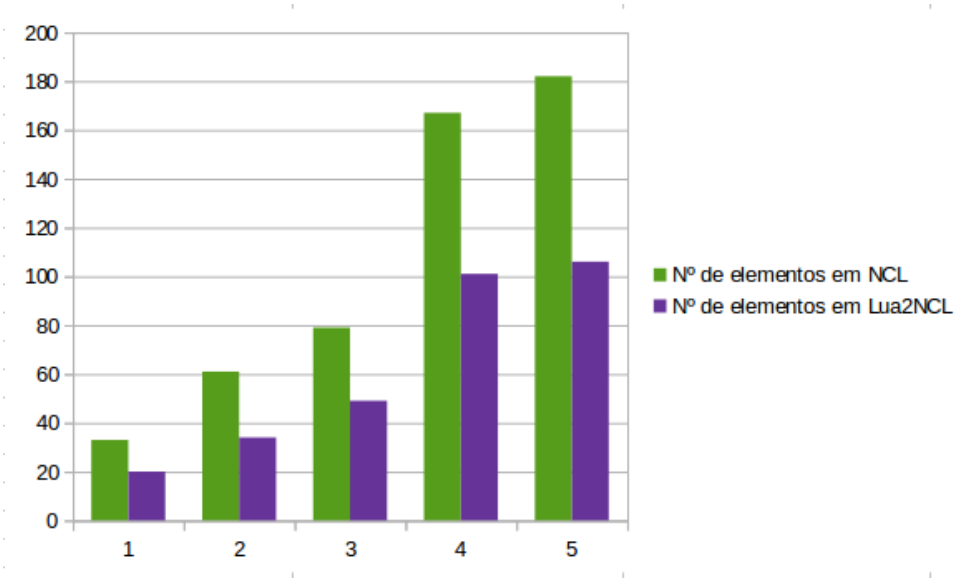


Figura 11 – Comparação do número de elementos em NCL e Lua2NCL

Nesse cenário de uso, observa-se que o Lua2NCL consegue reduzir a verbosidade de um documento NCL de forma significativa, no entanto não se pode afirmar que isso reduz o tempo de desenvolvimento da aplicação e nem mesmo que o Lua2NCL é uma abordagem mais fácil para autoria. Já que estes conceitos podem ser considerados subjetivos e inerentes ao perfil de cada desenvolvedor e deve ser avaliado em um trabalho futuro.

4 Conclusão

O Lua2NCL propõe-se a diminuir o esforço no desenvolvimento de aplicações hipermídia para TV Digital atacando diretamente a verbosidade da linguagem NCL. Essa verbosidade está ligada ao número de elementos que devem ser especificados em um documento para uma certa aplicação e consequentemente ao número de linhas do documento final.

O Lua2NCL consegue diminuir tanto a quantidade de elementos quanto o número de linhas de um documento sem alterar a semântica da aplicação, já que a aplicação final é o documento NCL equivalente gerado pelo próprio Lua2NCL. A função do Lua2NCL não é substituir a NCL mas sim servir como uma ferramenta de apoio ao desenvolvimento de aplicações para TV Digital.

É importante enfatizar que essa redução no número de elementos de um documento ocorre pelo fato do Lua2NCL abster-se da declaração de alguns elementos relacionados à organização e reúso do código, mas mantendo ainda uma nível adequado de organização. Outro fator importante é a mudança na representação dos elos, quem em NCL precisam definir explicitamente o conector que usará e as ligações dos papéis desse conector aos objetos de mídia por meio dos elementos filhos `<bind>`. Por outro lado, usando as tabelas o Lua2NCL consegue inferir o conector desejado pelo elo e ainda reduzir o número de elementos já que não utiliza os elementos de ligação.

O código Lua desenvolvido utilizando a abordagem do Lua2NCL é convertido em um documento NCL equivalente pelo tradutor do Lua2NCL. Assim, apesar do usuário descrever sua aplicação com a linguagem Lua, ao final ele tem um documento executável pelo *middleware* Ginga, gerado pelo *framework* e validado pelo NCL Validator.

Como trabalhos futuros pretende-se finalizar o desenvolvimento do *framework* estendendo aos elementos que ainda não são atendidos propriamente e após isso, aplicar alguns testes de usabilidade para buscar possíveis melhorias baseadas principalmente no *feedback* do público-alvo. Também é um objetivo futuro, a criação de uma versão da linguagem com sintaxe própria e que possa reduzir ainda mais a verbosidade de um documento hipermídia.

Referências

ABNT, N. 15606-2 (2007)–associação brasileira de normas técnicas,“. *Televisão digital terrestre–Codificação de dados e especificações de transmissão para radiodifusão digital–Parte*, v. 2, p. 15606–2, 2007. Citado 4 vezes nas páginas 14, 21, 32 e 33.

ABNT, N. *15606-2, 2011. Digital Terrestrial Television-Data Coding and Transmission Specification for Digital Broadcasting-Part 2: Ginga-NCL for fixed and mobile receivers-XML application language for application coding*. [S.l.]: São Paulo, Brazil, 2011. Citado 2 vezes nas páginas 21 e 22.

ADOBE. *Adobe Dreamweaver CS4 Professional.* 2007. Disponível em: <http://www.adobe.com/br/products/dreamweaver/>. Citado na página 17.

ANTONACCI, M. J.; MUCHALUAT-SAADE, D.; RODRIGUES, R.; SOARES, L. F. G. Ncl: Uma linguagem declarativa para especificação de documentos hipermídia na web. *VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia-SBMídia2000, Natal, Rio Grande do Norte*, 2000. Citado na página 21.

ARAÚJO, E. C.; AZEVEDO, R. G. d. A.; NETO, C. de S. Ncl-validator: um processo para validação sintática e semântica de documentos multimídia ncl. *II Jornada de Informática do Maranhão–São Luís, Brasil*, 2008. Citado 2 vezes nas páginas 27 e 28.

AZEVEDO, R. d. A.; TEIXEIRA, M. M.; NETO, C. d. S. S. Ncl eclipse: Ambiente integrado para o desenvolvimento de aplicações para tv digital interativa em nested context language. In: *SBRC: Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2009. Citado na página 19.

AZEVEDO, R. G. D. A. Ncl eclipse: editor textual para desenvolvimento de programas hipermıdia interativos em ncl. 2008. Citado 2 vezes nas páginas 18 e 27.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E.; YERGEAU, F. Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, v. 16, 1998. Citado na página 20.

COSTA, R. M. de R. *Controle do sincronismo temporal de aplicações hipermídia*. Tese (Doutorado) — Puc-Rio, 2010. Citado na página 16.

DAMASCENO, A. L.; GALABO, R. J.; NETO, C. S. S. Cacuriá: Authoring tool for multimedia learning objects. In: ACM. *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*. [S.l.], 2014. p. 59–66. Citado na página 17.

FILHO, G. L. d. S.; 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*, SciELO Brasil, v. 12, n. 4, p. 47–56, 2007. Citado na página 14.

GUIMARãES, R. L.; COSTA, R. M. R.; SOARES, L. F. G. Composer: Ambiente de autoria de aplicações declarativas para tv digital. In: ACM. *Proceedings of the 13th Brazilian Symposium on Multimedia and the Web*. [S.l.], 2007. Citado na página 41.

HAROLD, E. R. *XML Bible*. [S.l.]: IDG Books Worldwide, Inc., 1999. Citado 2 vezes nas páginas 19 e 20.

HOSCHKA, P.; BUGAJ, S.; BULTERMAN, D. et al. Synchronized multimedia integration language (smil) 1.0 specification, w3c recommendation 15-june-1998. *URL: http://www. w3. org/TR/REC-smil/: W3C*, 1998. Citado 2 vezes nas páginas 20 e 21.

IERUSALIMSCHY, R. *Programming in lua*. [S.l.]: Roberto Ierusalimschy, 2006. Citado 2 vezes nas páginas 30 e 31.

IERUSALIMSCHY, R. *Uma Introdução à Programação em Lua*. 2009. <http: //www.lua.org/doc/jai2009.pdf>. Citado 2 vezes nas páginas 29 e 30.

IERUSALIMSCHY, R.; FIGUEIREDO, L. H. D.; FILHO, W. C. Lua-an extensible extension language. *Softw., Pract. Exper.*, Citeseer, v. 26, n. 6, p. 635–652, 1996. Citado 3 vezes nas páginas 29, 30 e 31.

IERUSALIMSCHY, R.; FIGUEIREDO, L. H. de; CELES, W. The evolution of lua. In: ACM. *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. [S.l.], 2007. p. 2–1. Citado na página 30.

ISO, I. O. f. S. *Information Processing: Text and Office Systems: Standard Generalized Markup Language (SGML)*. [S.l.]: ISO 8879, 1986. Citado na página 20.

ITU-T. *H. 761, Nested Context Language (NCL) and Ginga-NCL for IPTV Services, Geneva, Apr. 2009*. 2009. Citado 3 vezes nas páginas 21, 22 e 33.

LIMA, G.; SOARES, L. F. G.; NETO, C. d. S. S.; MORENO, M. F.; COSTA, R. R.; MORENO, M. F. Towards the ncl raw profile. In: *II Workshop de TV Digital Interativa (WTVDI)-Colocated with ACM WebMedia*. [S.l.: s.n.], 2010. v. 10. Citado 2 vezes nas páginas 24 e 34.

LOWE, D. *Hypermedia and the Web: an engineering approach*. [S.l.]: John Wiley & Sons, Inc., 1999. Citado na página 16.

LUA. *Lua, the Programming Language*. 2016. Acessado em: 2015-03-20. Disponível em: <http://www.lua.org/>. Citado na página 29.

NELSON, T. H. Complex information processing: a file structure for the complex, the changing and the indeterminate. In: ACM. *Proceedings of the 1965 20th national conference*. [S.l.], 1965. p. 84–100. Citado na página 16.

ORACLE. *Java*. [S.l.]: Oracle, 2016. <http://www.oracle.com/br/java/>. Citado 2 vezes nas páginas 29 e 48.

OUSTERHOUT, J. K. Tcl: An embeddable command language. In: *Proceedings of the Winter 1990 USENIX Conference*. [S.l.: s.n.], 1990. Citado na página 29.

SAADE, D. C. M. *Relações em Linguagens de Autoria Hipermídia: Aumentando Reuso e Expressividade*. Tese (Doutorado) — Tese de Doutorado, Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil, 2003. Citado na página 21.

SAADE, D. C. M.; SILVA, H. V.; SOARES, L. F. G. Linguagem ncl versão 2.0 para autoria declarativa de documentos hipermídia. *IX Simpósio Brasileiro de Sistemas Multimídia e Web-WebMídia2003*, p. 1–17, 2003. Citado na página 21.

SANT'ANNA, F.; CERQUEIRA, R.; SOARES, L. F. G. Nclua: objetos imperativos lua na linguagem declarativa ncl. In: ACM. *Proceedings of the 14th Brazilian Symposium on Multimedia and the Web*. [S.l.], 2008. p. 83–90. Citado 2 vezes nas páginas 32 e 34.

SANTOS, R. C. M.; NETO, C. d. S. S. Contribuições para ferramentas de autoria desenvolvidas para a comunidade ginga. 2010. Citado 3 vezes nas páginas 16, 17 e 18.

SANT'ANNA, F.; NETO, C. de S. S.; BARBOSA, S. D. J.; SOARES, L. F. G. Nested context language 3.0 aplicações declarativas ncl com objetos nclua imperativos embutidos. *Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio*, n. 17, p. 07, 2009. Citado na página 32.

SOARES, L.; LIMA, G.; NETO, C. S. Ncl 3.1 enhanced dtv profile. In: *Workshop De Tv Digital Interativa em WebMedia*. [S.l.: s.n.], 2010. v. 1, n. 2, p. 44. Citado 3 vezes nas páginas 15, 34 e 39.

SOARES, L. F. G.; RODRIGUES, R. F. Nested context model 3.0: Part 1–ncm core. *Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio*, n. 18/05, 2005. Citado 2 vezes nas páginas 21 e 24.

SOARES, L. F. G.; RODRIGUES, R. F. Nested context language 3.0 part 8–ncl digital tv profiles. *Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio*, n. 35, p. 06, 2006. Citado 4 vezes nas páginas 21, 22, 24 e 34.

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*, SciELO Brasil, v. 12, n. 4, p. 37–46, 2007. Citado 3 vezes nas páginas 14, 21 e 32.

SOARES, L. F. G. S.; BARBOSA, S. D. J. *Programando em NCL 3.0: desenvolvimento de aplicaçoes para middleware Ginga: TV digital e Web*. [S.l.]: Elsevier, 2009. Citado 6 vezes nas páginas 14, 21, 22, 25, 26 e 48.

STENNING, K.; OBERLANDER, J. A cognitive theory of graphical and linguistic reasoning: Logic and implementation. *Cognitive science*, Wiley Online Library, v. 19, n. 1, p. 97–140, 1995. Citado na página 17.

W3C, W. W. W. C. et al. Html 4.01 specification. World Wide Web Consortium, 1999. Citado na página 20.

W3C, W. W. W. C. et al. *XML Schema*. [S.l.]: World Wide Web Consortium, 2003. Citado na página 27.

W3C, W. W. W. C. et al. Synchronized multimedia integration language–smil 2.1 specification. *W3C Recommendation*, v. 15, 2005. Citado 2 vezes nas páginas 20 e 21.

W3C, W. W. W. C. et al. Extensible markup language (xml) 1.1. World Wide Web Consortium, 2006. Citado na página 20.