Specifying a Domain Specific Language for Simplifying the Authoring of Digital TV Applications

Lucas de Macedo Terças[1,2], Daniel de S. Moraes[1,2], Carlos de Salles Soares Neto[1,2]
[1]Telemídia - UFMA
[2]Laboratory of Advanced Web Systems - UFMA
Av. dos Portugueses, Campus do Bacanga
São Luís/MA, Brazil
(lucastercas, daniel)@laws.deinf.ufma.br
csalles@deinf.ufma.br

ABSTRACT

The Brazilian Terrestrial Digital TV System (SBTVD-T) uses the NCL language for multimedia application's authoring. Although such language has been designed to be easy to understand by producers of such applications, the fact of being XML based introduces a reasonable level of verbosity, which hinders or delays more experienced software developers. This paper proposes sNCL (simpler NCL) a domain specific language (DSL) focused on the reduction of verbosity in the construction of NCL documents. The sNCL approach does not act as a replacement to the use of NCL, but facilitates its use. Therefore, this scientific initiation research also proposes the development of a compiler, which generates the final NCL document from a sNCL document. This compiler will be implemented in Lua with LPeg library, generating a symbol table, where each table identifier corresponds to a different identifier of the XML elements to be generated in the final document. The compiler development process will be guided by Test Driven Development (TDD), which ensures a lower probability of errors.

Keywords

Digital TV; NCL; sNCL; DSL

1. INTRODUÇÃO

Sistemas de autoria multimídia têm sido discutidos na área acadêmica há anos. Um desafio têm sido prover formas de tornar o processo de criação de apresentações com diferentes tipos de conteúdo e seus relacionamentos o mais simples e eficaz possível e acessível a usuários *experts* ou não na área [5].

NCL [18] é uma linguagem declarativa de alto nível de abstração, baseada em XML e no modelo NCM (*Nested Context Model*) [17], usada na autoria de aplicações multimídia para TV Digital no Brasil [2] e também recomendação ITU-T para serviços IPTV [12] e padrão ISDB-TB do Sistema Nipo-Brasileiro de TV Digital Terrestre. Esse tipo de aplicação tem como um dos seus autores alvo os produtores de conteúdo e profissionais das emissoras de TV. Por isso, há a necessidade que a linguagem usada para autoria seja de fácil leitura e entendimento por esses profissionais, que não necessariamente têm conhecimento em desenvolvimento de aplicações.

A linguagem atualmente usada, NCL, é baseada em XML, que foi pensada para facilitar a abstração e a especificação de novas linguagens, o que a torna muito verbosa, e embora isso facilite a criação de aplicações por quem não tem conhecimento de programação, como programadores-visuais e desenvolvedores de animações, à medida que o documento da aplicação cresce, ele fica muito confuso e difícil de ler. Além disso, há o fato que programadores experientes produzindo aplicações para TV Digital não precisam de tal verbosidade e isso até atrapalha sua produtividade, geralmente preferindo uma linguagem mais compacta e simples.

O reúso em NCL foi implementado para possibilitar a reutilização de código dentro da aplicação, por aplicações diferentes, e até por bibliotecas externas, o que diminui a probabilidade de erros se o código reusado for bem testado e diminui também o tempo de desenvolvimento da aplicação. Embora a técnica de reúso seja geralmente implementada em linguagens imperativas, ela foi usada em NCL, uma linguagem declarativa, pelo fato de que na criação de aplicações para TV Digital existe a possibilidade de mídias diferentes terem características de apresentação iguais. Exemplos disso são como ou onde as mídias serão mostradas.

Tendo como objetivo criar uma linguagem mais simples e menos verbosa para desenvolvedores, este trabalho sugere a criação de uma linguagem declarativa de domínio específico, chamada sNCL (*simpler* NCL), baseando-se na sintaxe da linguagem Lua[11], devido ao uso dela como linguagem de extensão e *script* para NCL, com o NCLua[14].

2. TRABALHOS RELACIONADOS

Existem vários trabalhos comprometidos com a redução da verbosidade da NCL, como *scripts*[7, 15] ou que usam *templates*[6, 4] para a autoria, há também a sugestão de um novo perfil NCL, chamado NCL Raw, que visa ser uma linguagem intermediária entre o usuário e o *middleware*, removendo açucares sintáticos e entidades do NCL. Cada abordagem sugere meios de tornar a autoria de aplicações multimídia mais suave aos autores alvo.

Em TAL, os autores propõem uma linguagem para autoria de *templates* de aplicações hipermídia, onde a criação de uma aplicação passa a ter duas etapas: a autoria dos

In: Workshop de Trabalhos em Iniciação Científica (WTIC), 13., 2016, Teresina. Anais do XXII Simpósio Brasileiro de Sistemas Multimídia e Web. Porto Alegre: Sociedade Brasileira de Computação, 2016. v. 2.
ISBN: 978-85-7669-332-1
©SBC – Sociedade Brasileira de Computação

templates, que são documentos incompletos que descrevem aplicações deixando lacunas; e a criação da aplicação final, onde tais lacunas são preenchidas. Apresenta-se a separação dos papéis envolvidos na autoria, onde tem-se o autor de *templates* e o autor final, que produz uma aplicação a partir dos *templates* disponíveis. Com essa divisão, a aplicação em desenvolvimento possui diferentes arquivos com tamanhos reduzidos.

O trabalho em Luar introduz um sistema mais robusto para processamento de *templates*, onde tem-se dois processadores: um responsável pelo processamento dos *templates* e geração de uma biblioteca dos mesmos (documentos luar), que é compartilhada entre os desenvolvedores; e um segundo, chamado processador de aplicação, responsável por interpretar os *templates* e os enviar ao primeiro processador. A biblioteca também permite a inserção de trechos de código Lua no documento NCL, transformando-os em um documento luar.

Esses dois trabalhos se relacionam ao proposto neste artigo por aplicar técnicas de reúso.

No trabalho Lua2NCL, usa-se a linguagem Lua para criar um documento que passa por rotinas responsáveis pela geração do arquivo NCL final. Nesse projeto são usadas tabelas Lua para representar os elementos do documento NCL, onde cada elemento é representado como uma classe.

Ao instanciar um objeto de umas das classes, não há a necessidade de repetição dos termos, como acontece com as *tags* XML. Embora haja uma redução nos documentos de aplicações, o uso de tabelas Lua ainda possui uma verbosidade considerável quando se tem muitos aninhamentos de tabelas. Isso porque a sintaxe de Lua para tabelas exige o uso de muitos símbolos, como chaves e colchetes, o que pode ainda não ser uma boa opção para simplificar o processo de criação de aplicações.

Já em [15] o trabalho propõe uma forma de autoria alternativa, chamada JNS, *JSON NCL Script*, em que o desenvolvedor cria um documento JSON e o compila para NCL.

O JNS também tem como objetivo reduzir a verbosidade. Tendo como alvo o usuário final, sejam estes desenvolvedores ou produtores de conteúdo. Oferece-se uma sintaxe mais limpa, tornando o desenvolvimento de aplicações mais rápido e fácil, provendo uma alternativa a desenvolvedores que não gostam do estilo de um documento XML. No entanto, essa alternativa requer o conhecimento de JSON.

Em ambos os trabalhos, Lua2NCL e JNS, utiliza-se de estruturas de linguagens já existentes, Lua e JSON respectivamente, para a construção dos documentos, o que exige o conhecimento prévio das linguagens base.

No entanto, são linguagens para diferentes perfis, já que JSON é bastante usada para aplicações web, e Lua, além de outras utilizações, vem sendo muito usada nas aplicações DTV. A linguagem proposta tem sintaxe própria e reduz o uso de símbolos como chaves e dois pontos, que são bastantes usados nas abordagens citadas.

É proposto em [13] um novo perfil para NCL, *NCL Raw*, que remove as estruturas de reúso, deixando somente *links*, *context*, *port*, *media* e *properties*.

O projeto NCL Raw foi proposto a fim de melhorar o processo de conversão do documento NCL para o modelo do player Ginga-NCL [16], e, para isso, o perfil remove os açúcares sintáticos e as possibilidades de reúso do NCL, para implementar uma linguagem intermediária, mas que ainda assim é fácil de entender para pessoas que não têm experiência com programação.

O trabalho apresentado neste artigo se baseia no citado logo acima para um uso mínimo de estruturas sem afetar a semântica das aplicações e promovendo o reúso.

3. SNCL

Este trabalho propõe a criação da sNCL (*simpler* NCL), uma linguagem de domínio específico declarativa, voltada para o desenvolvimento de aplicações multimídia para o Sistema Brasileiro de TV Digital Terrestre (SBTVD-T), visando diminuir a verbosidade do documento final, e fazer a autoria de tais aplicações por desenvolvedores mais fácil.

A sNCL planeja trazer um modo mais simples na autoria de aplicações para o Ginga-NCL. As aplicações escritas usando a *sNCL* serão compiladas para a linguagem final usada no *middleware*, a NCL, usando como parser a biblioteca *Lua Parsing Expression Grammar* (LPeg) [10, 1] e assim fornecer aos desenvolvedores que não querem usar o estilo de um documento XML, preferindo uma sintaxe mais parecida com as línguas a que estão familiarizados.

3.1 Proposta de Sintaxe

A linguagem proposta neste trabalho sugere o uso de uma sintaxe parecida com a da linguagem imperativa Lua, já que Lua também é usada como uma linguagem de extensão e *script* para aplicações de TV Digital, o NCLua. Portanto, o uso da sintaxe semelhante proporciona aos desenvolvedores uma familiaridade com a nova linguagem.

Espera-se que essa nova sintaxe propicie uma redução na verbosidade da aplicação, uma vez que ela não faz uso de XML, e devido a semelhança dela com a sintaxe de linguagens imperativas, que ela facilite o desenvolvimento para programadores que estão mais familiarizados com tal sintaxe.

A sNCL usa os elementos de NCL, como *media, area, context*, e seus atributos como palavras reservadas da língua, que simulam um tipo de dado, como *int, float.* A listagem 1 representa a declaração de uma *region*, de uma *media* e de um *descriptor*, e dentro da *media* há também a declaração de uma *area*.

```
1  region bgReg
2     width = "100%"
3     height = "100%"
4     zIndex = "10"
5  end
6  media foto01
7     right = "10%"
8     descriptor = bgDesc
9     source = "../media/foto01"
10    area aFoto01
11       begin = "5s"
12       end = "10s"
13    end
14 end
15 descriptor bgDesc
16    region = bgReg
17 end
```

Listagem 1: Exemplo da declaração de uma mídia em sNCL

A Listagem 2 apresenta a comparação da declaração de um elo em NCL (linha 1 á 6) e em sNCL (linha 9 á 13), nota-se que em sNCL, os elos perdem o atributo id, assim, não podendo referenciá-lo em um momento futuro, e não há

a necessidade de explicitar uma base de conectores, pois o compilador infere e os cria a medida que o *parsing* é feito, sendo que, se o programador criar um elo diferente dos pré-definidos pela linguagem NCL, o processo de compilação é interrompido, não gerando o documento sNCL final.

```
1  <link id="mediaGol" xconnector="conEx#
      onBeginStartNStop">
2    <bind role="onBegin" component="
      animation" interface="momentoGol"/>
3    <bind role="start" component="foto2" />
4    <bind role="start" component="img1" />
5    <bind role="stop" component="img2" />
6  </link>
7
8
9  onBegin animation.momentoGol do
10    start foto2
11    start img1
12    stop img2
13 end
```

Listagem 2: Exemplo de um elo em NCL (linha 1-6) e do mesmo elo em sNCL (linha 9-13).

3.2 Compilador

Em 2004, introduziu-se a Gramática de Análise Sintática de Expressão (GASE), ou em inglês, *Parsing Expression Grammar* (PEG), por Bryan Ford [8]. Diferentemente das Gramáticas Livres de Contexto, onde uma expressão pode resultar em mais de uma árvore, a PEG não possui ambiguidade, cada cadeia analisada possui apenas uma árvore que a representa. O LPeg é uma biblioteca Lua que baseia-se nesse conceito para auxiliar na análise de PEGs, por isso será usada para fazer a construção do compilador da sNCL.

O compilador sNCL usará uma tabela de símbolos, onde cada elemento NCL é um identificador para uma tabela Lua que o representa (na Listagem 1, os identificadores são 'foto1' para a mídia, e 'aFoto1' para a area).

Ao carregar o arquivo com o código sNCL, o compilador permite a declaração prévia de elementos para depois específica-los, como na Listagem 1, em que o *descriptor* bgDesc é citado na linha 8, para na linha 15 ser especificado. Assim, problemas como uma *media* depender de um *descriptor* que ainda não foi especificado são resolvidos, contanto que tal *descriptor* tenha sido previamente declarado, porém, se isso não tiver acontecido, o compilador tenta continuar, mas não produz um documento sNCL final, o mesmo acontece quando um elemento não tem identificador, ou quando é encontrado o mesmo identificador sendo especificado novamente.

Porém, ainda assim tem-se uma fase de varredura da tabela de símbolos para checagem de tais dependências. No caso de aninhamento de elementos, o índice do elemento só aponta para o local em que as informações sobre ele estão na tabela de símbolos, eliminando assim a ocorrência de redundância de memória, como ilustrado na Figura 1.

Depois da construção da tabela de símbolos e verificação do cumprimento das dependências entre elementos, faz-se uma separação entre identificadores dos elementos que compõem o elemento *head* do NCL e dos elementos filhos do *body*. A seguir, tais tabelas são devidamente escritas no documento NCL que é gerado.

Há também o aspecto de importação em NCL, que é resolvido em sNCL com a inclusão de outros arquivos sNCL

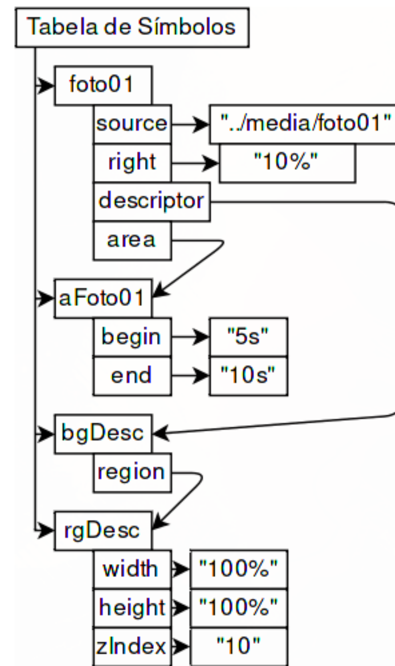

Figura 1: Exemplo da Tabela de Símbolos após a leitura da Listagem 1.

como bibliotecas no começo do arquivo principal, como mostra a Listagem 4. No exemplo, o arquivo bases.sncl contém as definições para o *descritor* bgDesc que a *media* foto2 usa.

```
1 #include bases
2 media foto2
3    descriptor = bgDesc
4 end
```

Listagem 3: Exemplo de um trecho de reuso e importação em sNCL.

O ambiente de testes consiste na compilação de tais documentos sNCL por um protótipo do compilador específico para um pacote de testes. O arquivo de saída gerado pelo compilador será então comparado com um arquivo com a saída esperada, previamente escrito pelos desenvolvedores. Isso é feito para a análise de todos os elementos, assegurando completude, ou seja, que cada elemento seja analisado corretamente e que o documento final NCL seja gerado de maneira congruente. A Figura 2 ilustra o processo de desenvolvimento do compilador.

Embora artigos [9] sugiram que essa abordagem gaste mais tempo (16% a mais), ela também assegura que o código fique mais limpo, com maior qualidade e menos propenso a erros inesperados por parte do compilador, que não seriam detectados caso fosse adotado outro modelo, por exemplo, o modelo de desenvolvimento em cascata, em que a verificação e validação do software só é realizada no final do processo.

4. DISCUSSÃO

Como o compilador da sNCL ainda está em fase de implementação, há funcionalidades de NCL que ainda não são resolvidas. Como os conectores, que em NCL não são pré-definidos, e sim criados pelo autor da aplicação, o que gera

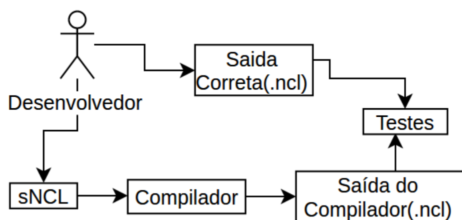

Figura 2: Exemplo do processo de Desenvolvimento Orientado a Testes

a possibilidade de uma gama de conectores diferentes. Para resolver esse problema, sugere-se que o compilador consiga inferir e gerar automaticamente os conectores a partir dos elos usados em um documento, tornando a criação dos conectores invísivel ao autor. Porém, o compilador também permitirá a declaração prévia dos conectores que irão ser utilizados no elos, para depois defini-los.

Pretende-se, durante a criação da linguagem, a execução de experimentos para verificar a efetividade da linguagem proposta em reduzir a verbosidade dos documentos e do tempo de desenvolvimento, fazendo testes com desenvolvedores, e usando como parâmetros o tamanho do documento final, o número de palavras, e o tempo comparando-se com o desenvolvimento de aplicações usando a linguagem NCL. Pretende-se avaliar a sintaxe da linguagem proposta, aplicando-se questionários tanto com desenvolvedores experientes, quanto com pouca experiência no desenvolvimento de aplicações para TV Digital.

O processo de desenvolvimento do compilador baseia-se na técnica de Desenvolvimento Orientado a Testes [3], utilizando exemplos de documentos sNCL escritos de maneira correta e documentos com erros de sintaxe e suas respectivas saídas em NCL. Por meio dos testes, verifica-se automaticamente se tais saídas estão de acordo com as saídas esperadas.

5. CONSIDERAÇÕES FINAIS

Considerando os trabalhos relacionados, a sNCL diferencia-se pela proposta de uma DSL que seja mais familiar para programadores que preferem uma abordagem imperativa. Embora o Lua2NCL [7] use a linguagem Lua, ele limita-se ao uso das tabelas para representar os elementos de NCL, criando um certo nível de complexidade na compreensão dos códigos. A sNCL propõe uma maior simplicidade na descrição das aplicações, utilizando uma linguagem de mais alto nível que não só reduza a verbosidade mas que seja também de fácil entendimento aos desenvolvedores.

A sNCL manterá as funcionalidades do NCL 3.0, como a possibilidade de reúso e importação de bases que ficam em outros documentos sNCL.

Levando em conta que os códigos de aplicações em sNCL serão transcritos para NCL, este trabalho não apresenta a sNCL como uma substituta da NCL, que já possui um *player* próprio, o Ginga-NCL, mas sim como uma alternativa para desenvolvedores mais familiarizados com linguagens imperativas.

6. REFERENCES

[1] Lpeg: Parsing expression grammars for lua. available at http://www.inf.puc-rio.br/˜roberto/lpeg.

[2] ABNT. 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, 2011.

[3] K. Beck. *Test Driven Development: By Example.* 2003.

[4] D. H. D. Bezerra, D. M. T. de Sousa, G. L. de S. Filho, A. M. F. Burlamaqui, and I. R. de M. Silva. Luar: A language for agile development of ncl templates and documents.

[5] D. C. Bulterman and L. Hardman. Structured multimedia authoring. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 1(1):89–109, 2005.

[6] C. de Salles Soares Neto, H. F. Pinto, and L. F. G. Soares. Tal processor for hypermedia applications.

[7] D. de Sousa Moraes, A. Damasceno, A. Busson, and C. S. Neto. Lua2ncl: Framework for textual authoring of ncl applications using lua. 2016.

[8] B. Ford. Parsing expression grammars: A recognition-based syntactic foundation. 2004.

[9] B. Georgea and L. Williamsb. A structured experiment of test-driven development.

[10] R. Ierusalimschy. A text pattern-matching tool based on parsing expression grammars.

[11] R. Ierusalimschy, L. H. de Figueiredo, , and W. Celes. Lua: an extensible extension language.

[12] ITU-T. H. 761, nested context language (ncl) and ginga-ncl for iptv services, geneva, apr. 2009, 2009.

[13] G. A. F. Lima. Eliminando redundâncias no perfil ncl edtv. 2011.

[14] F. Sant'Anna, R. Cerqueira, and L. F. G. Soares. Nclua: objetos imperativos lua na linguagem declarativa ncl, 2008.

[15] E. C. O. Silva, J. A. F. dos Santos, and D. C. Muchaluat-Saade. Jns: An alternative authoring language for specifying ncl multimedia documents. 2013.

[16] L. F. G. Soares, C. de Salles Soares Neto, and M. Moreno. Ginga-ncl: Declarative middleware for multimedia iptv services.

[17] L. F. G. Soares and R. F. Rodrigues. Nested context model 3.0: Part 1–ncm core. *Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio*, (18/05), 2005.

[18] L. F. G. Soares and R. F. Rodrigues. Nested context language 3.0 part 8–ncl digital tv profiles. *Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio*, page 06, 2006.