NCLua - Objetos Imperativos Lua na Linguagem Declarativa NCL

Francisco Sant'Anna Renato Cerqueira Luiz Fernando Gomes Soares

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro – 22453-900 – Brasil

francisco@telemidia.puc-rio.br, rcerq@inf.puc-rio.br, lfgs@inf.puc-rio.br

RESUMO

Linguagens declarativas são de mais fácil aprendizado por profissionais sem formação em programação. No entanto, pecam pela flexibilidade, sendo difícil a realização de tarefas que fujam do escopo da linguagem.

O poder de uma linguagem declarativa é elevado quando integrada com uma linguagem imperativa, passando a ter acesso a recursos computacionais genéricos. Essa integração deve seguir critérios que não afetem os princípios da linguagem declarativa, mantendo uma separação bem definida entre os dois ambientes.

Este trabalho apresenta a integração entre as linguagens NCL declarativa e Lua imperativa, especificada e desenvolvida para o middleware Ginga do padrão brasileiro de TV digital.

ABSTRACT

Declarative languages are easier to learn by non-programmer professionals. On the other hand, they lack flexibility, being hard to perform tasks out of the language's scope.

The power of a declarative language is leveraged when integrated with an imperative language, bringing generic computation to the language. This integration should not conflict with the principles of the declarative language, keeping a clear boundary between the two environments.

This work presents the integration between the declarative NCL and imperative Lua languages, specified and developed for the middleware Ginga, part of the brazilian digital TV standard.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classifications—*Extensible languages, Multiparadigm languages, Specialized application languages*

General Terms

Design, Languages, Standardization

Keywords

NCL, Lua, Digital TV, Ginga, middlewares, declarative languages, multimedia systems

1. INTRODUÇÃO

O padrão brasileiro de TV digital tem como *middleware* o sistema Ginga [12], camada de software que permite o desenvolvimento de aplicações interativas portáveis para a TV Digital utilizando a linguagem NCL (Nested Contex Language).

NCL [14] é uma aplicação XML baseada no NCM [13] (Nested Context Model), modelo conceitual para especificação de documentos hipermídia com sincronização temporal e espacial entre seus objetos de mídia. NCL permite ao autor descrever o comportamento espacial e temporal de uma apresentação multimídia, associar *hyperlinks* (interação do usuário) a objetos de mídia, definir alternativas para apresentação (adaptação) e descrever o leiaute da apresentação em múltiplos dispositivos.

A autoria de aplicações utilizando linguagens declarativas como NCL é vantajosa quando o seu desenvolvimento depende apenas de recursos previstos no projeto da linguagem. No entanto, quando uma aplicação necessita de funcionalidades não previstas pela linguagem declarativa, a solução pode se tornar complicada ou até mesmo impossível.

Em NCL, a realização de muitas tarefas é complicada sem auxílio imperativo, tal como processamento matemático, manipulação sobre textos, uso do canal de interatividade, controle fino do teclado, animações e colisões para objetos gráficos e, de maneira geral, tarefas que necessitem da especificação de algoritmos e estruturas de dados.

Por outro lado, linguagens imperativas, apesar de genéricas, introduzem uma maior complexidade de programação e dependem de uma base lógica que autores de conteúdo áudio-visual nem sempre possuem.

Uma solução para esse impasse consiste em adicionar à linguagem declarativa algum suporte imperativo, assim, o autor da aplicação usa a forma declarativa sempre que possível e lança mão da forma imperativa somente quando necessário.

A criação da nova classe de objetos de mídia Lua, os quais são chamados de NCLua, é a principal via de integração de NCL a um ambiente imperativo, conforme definido em seu

perfil para TV Digital e apresentada neste artigo. Por meio de elementos de mídia, scripts NCLua podem ser inseridos em documentos NCL, trazendo poder computacional adicional às aplicações declarativas. [1]

Lua é uma linguagem de programação poderosa, rápida e leve, projetada para estender aplicações. Lua combina sintaxe simples para programação procedural com poderosas construções para descrição de dados baseadas em tabelas associativas e semântica extensível. Lua é tipada dinamicamente, é interpretada a partir de bytecodes para uma máquina virtual baseada em registradores, e tem gerenciamento automático de memória com coleta de lixo incremental. Essas características fazem de Lua uma linguagem ideal para configuração, automação (scripting) e prototipagem rápida [8].

Este artigo apresenta o uso de Lua como linguagem de script NCL e traz como principal contribuição a forma não intrusiva como a integração entre as duas linguagens é realizada. Apesar de tratar da integração entre NCL e Lua, tanto os requisitos quanto a solução adotada são suficientemente genéricas para servirem de base para outras integrações entre ambientes declarativos e imperativos.

O artigo está organizado como se segue. A Seção 2 descreve os objetivos almejados em uma integração minimamente intrusiva entre os ambientes declarativo e imperativo por meio da ponte NCL-Lua. A Seção 3 apresenta como outras linguagens declarativas tratam objetos imperativos embutidos, salientando o fato de não satisfazerem os requisitos de integração levantados na seção anterior. A Seção 4 descreve a abstração para objetos de mídia em NCL, o que são e como se relacionam em documentos multimídia. A Seção 5 trata especificamente dos objetos de mídia NCLua, como diferenciam-se de scripts Lua comum, suas bibliotecas, seu ciclo de vida dentro de um documento NCL e sua onipresente API de eventos. Um exemplo ilustrativo, apresentado na Seção 6, explora as características da integração apresentadas no artigo. Por fim, a Seção 7 conclui e aponta alguns caminhos a serem percorridos no futuro.

2. OBJETIVOS NA INTEGRAÇÃO NCL-LUA

Na integração entre linguagens já especificadas e implementadas, como é o caso de NCL e Lua, deve-se mexer o mínimo nas linguagens para evitar o surgimento de dependências mútuas que comprometam a evolução independente de cada uma delas. Como se verá, no caso em questão, esse objetivo foi alcançado, uma vez que não foi necessário adicionar novos elementos ou atributos à linguagem NCL. Tampouco houve mudanças no compilador ou máquina virtual de Lua, linguagem que foi originalmente concebida para ser embarcada em outras linguagens.

Linguagens declarativas como NCL são acessíveis a autores de conteúdo áudio-visual que não possuem base de programação. Sendo assim, também é desejado que haja o mínimo de intercalação entre código declarativo e imperativo de modo a simplificar a divisão de tarefas entre equipes de profissionais técnicos e não técnicos. Esse objetivo também foi alcançado. Um NCLua deve ser escrito em um arquivo separado do documento NCL, que apenas o referencia. Os scripts NCLua usam a mesma abstração para objetos de mídia usada para imagens, vídeos e outras mídias (ver Seção 4). Em outras palavras, um documento NCL apenas relaciona objetos de mídia, não importando qual o tipo de seu conteúdo.

Por fim, em apresentações multimídia, o documento NCL deve ser o componente mestre onde todos os relacionamentos entre objetos de mídia devem ser definidos explicitamente (ver Seção 4). De fato, como se verá, a especificação desenvolvida impede que objetos de mídia (em particular os NCLua) sobrepujam essa hierarquia por meio de acessos diretos à estrutura do documento.

Resumindo, a motivação de mínima intrusão destaca três requisitos na integração NCL-Lua:

1. As linguagens devem ser alteradas o mínimo possível.

2. Do ponto de vista do desenvolvedor, deve ser mantida uma fronteira bem delineada entre os dois modos de programação.

3. Em termos de design, a relação entre os dois ambientes deve ser ortogonal, em outras palavras, operações em um ambiente não devem produzir efeitos imprevistos no outro.

3. TRABALHOS RELACIONADOS

Outros sistemas de TV Digital também possuem *middlewares* com suporte declarativo e imperativo. Os padrões ARIB japonês [1], ATSC americano [2] e DVB europeu [7] utilizam como ambiente declarativo variações do XHTML (BML, DVB-HTML) com suporte a DOM, CSS e ECMAScript. Com a exceção do japonês, também adotam Java como ambiente imperativo de desenvolvimento.

Linguagens declarativas baseadas no tempo para a Web também oferecem algum suporte imperativo, tais como SMIL e SVG. SVG (Scalable Vector Graphics) [3] é um padrão W3C para descrever gráficos bi-dimensionais. Através do uso de SMIL ou ECMAScript é possível criar animações gráficas. A sua integração com ECMAScript possui as mesmas características encontradas na integração XHTML- ECMAScript. SMIL, também padrão W3C, é discutida à parte.

3.1 XHTML - ECMAScript

O padrão ECMAScript (através do seu dialeto JavaScript) está hoje disseminado na *Web*, tornando possível que autores adicionem funcionalidades imperativas aos seus documentos XHTML.

No entanto, a abordagem utilizada na integração XHTML-ECMAScript, encontrada nos sistemas de TV Digital e na Web, é bastante intrusiva e, portanto, vai contra os requisitos apontados na Seção 2:

1. Em XHTML, scripts nem sempre são entidades especiais (elementos `<object>` ou `<script>`), não existindo, obrigatoriamente, uma abstração mais geral que os englobe (tal como os objetos de mídia da Seção 4).

2. O código em ECMAScript pode ser escrito dentro de documentos XHTML, ou mais intrusivamente, dentro de atributos de elementos XHTML.

[1] A especificação Ginga também prevê a integração de NCL com o ambiente Ginga-J, que permite o desenvolvimento de aplicações escritas em Java [6]. O desenvolvimento em Java, puramente imperativo, no entanto, foge ao escopo deste artigo.

3. ECMAScript tem acesso e pode alterar a árvore DOM do documento XHTML.

Além das mudanças que XHTML sofreu para incorporar uma linguagem de script, ECMAScript também está fortemente atrelada à cultura *web*, sendo incomum encontrá-la em uso fora desse ambiente.

O exemplo abaixo mostra a comum mistura de código XHTML e ECMAScript:

```
<input type="button" onclick="myFunc(...)"/>
```

A chamada à função `myFunc` está inserida em um atributo de um elemento XHTML. O autor XHTML deve, portanto, estar familiarizado com conceitos de programação tais como função, chamada e argumentos, tendo inclusive que conhecer suas respectivas sintaxes em ECMAScript.

Esse exemplo ilustra que a fronteira entre os dois modos de programação não é rígida, o que dificulta a separação total de código e, conseqüentemente, de tarefas entre equipes.

Por fim, o acesso irrestrito à árvore DOM XHTML por scripts ECMA compromete a ortogonalidade da integração, principalmente por depender dela para a realização de tarefas corriqueiras. A forma habitual de acesso a valores de atributos de elementos do documento é com acesso direto à sua estrutura, por exemplo:

```
document.getElementByID('myInput').value
```

3.2 SMIL

SMIL ([5]) é uma linguagem com propósitos similares aos de NCL (linguagens XML baseadas no tempo - *XML time-based languages*), no entanto, até sua versão corrente 2.1, ainda não possui suporte imperativo. A versão 3.0 de SMIL, em desenvolvimento, através do módulo `State` ([4]) visa dar ao autor um controle maior do fluxo das apresentações através da manipulação explícita de estado (variáveis ou propriedades) de um documento. Em particular o atributo `expr` poderá ser utilizado dentro de um elemento de mídia e aceitará expressões escritas em XPath, ou outra linguagem imperativa (Python é citada como exemplo):

```
<audio src="background.mp3"
       expr="smil-bitrate()>1000000"/>
```

No exemplo acima, o áudio só é tocado caso a velocidade de conexão com a internet seja superior a 1Mbps.

Essa abordagem é menos poderosa que a integração com uma linguagem imperativa completa e também não atende aos requisitos almejados, pois não cria uma fronteira rígida entre os dois modos de programação.

Como NCL, SMIL não prescreve e nem restringe qualquer tipo de objeto de mídia. Assim, também em SMIL pode-se ter código imperativo como conteúdo de um objeto de mídia. No entanto, SMIL não define um modelo de comportamento e nem de ciclo de vida de tais objetos, deixando a cargo da implementação a integração dos dois mundos (imperativo e declarativo) que pode ser totalmente intrusiva.

4. OBJETOS DE MÍDIA NCL

A linguagem NCL tem como característica uma separação acurada entre o conteúdo das mídias e a estrutura do documento (aplicativo NCL). NCL apenas referencia conteúdos de mídia e não é capaz de exibi-los, papel este que cabe aos adaptadores de mídia (*plug-ins*).

Um documento NCL define apenas como os objetos de mídia são estruturados e relacionados, no tempo e espaço. Como uma linguagem de cola, ela não restringe ou prescreve os tipos de conteúdo dos objetos de mídia. Nesse sentido, pode-se ter como objetos de mídia: imagens, vídeos, áudios, textos e também scripts NCLua.

4.1 Relacionamentos entre Objetos de Mídia

Os relacionamentos entre mídias de um documento NCL são tratados em separado das próprias mídias e com sintaxe própria, através do elemento `<link>`. Mais do que isso, os relacionamentos são independentes do tipo de mídia, ou seja, a sintaxe para relacionar vídeos, textos, imagens ou scripts NCLua é idêntica [9].

Segue um exemplo ilustrativo de um relacionamento entre um vídeo e um NCLua:

```
<media id="myvideo" src="video.mpg"/>
<media id="mynclua" src="script.lua"/>
<link>
    <bind role="onBegin" component="myvideo"/>
    <bind role="start"   component="mylua"/>
</link>
```

Esse relacionamento cria um elo entre `myvideo` e `mynclua`: quando o vídeo inicia (`role="onBegin"`), a ação do elo é disparada, iniciando a execução do NCLua (`role= "start"`).

Deve-se interpretar o elo como uma relação entre eventos: no caso, o evento de início do vídeo dispara o evento de início do NCLua. O uso da nomenclatura de eventos é bastante utilizada no decorrer do texto, principalmente na discussão do modelo de execução de um objeto NCLua.

Repare, no exemplo, que o conteúdo das mídias não é definido no documento, mas apenas referenciado pelo atributo `src` dos elementos `<media>`. Também é fácil notar que o objeto NCLua não possui nenhum privilégio, tanto a sua definição quanto seu uso em um elo são exatamente iguais ao do vídeo ou de outro objeto de mídia qualquer.

4.2 Âncoras de Objetos de Mídia

Os objetos se relacionam através de suas âncoras em um documento NCL. Para objetos de mídia, há dois tipos de âncoras: de conteúdo e de propriedade, chamadas de áreas e propriedades, respectivamente.

Áreas definem porções do conteúdo que são exportadas como âncoras para o documento NCL. É responsabilidade de quem especificou os adaptadores da mídia interpretar a semântica dada pelo perfil da linguagem a elas. Por exemplo, para uma imagem, pode ser uma região de seu quadro completo; para um vídeo, um trecho temporal.

No caso do NCLua, a semântica não é definida, mas postergada para que o autor de cada script possa dar um comportamento próprio para suas áreas. A Seção 5 discute como isso é feito. Um possível comportamento é o de executar uma função em Lua de mesmo nome da área especificada no documento NCL.

O outro tipo de âncora, de propriedade, define pares *"nome = valor"* e serve para parametrizar o comportamento dos adaptadores. Também é de responsabilidade do adaptador de mídia interpretar a semântica dada pelo perfil da linguagem a cada propriedade. Por exemplo, um adaptador de texto deve ser capaz de interpretar as propriedades *fontStyle* e *fontSize*; já um adaptador de som, as propriedades *soundLevel* e *bassLevel*. Propriedades globais podem ser criadas pelo autor para guardar variáveis de estado da aplicação,

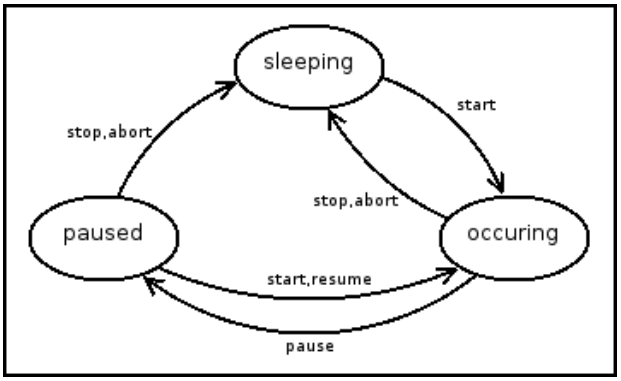

Figura 1: Máquina de Estados NCL.

por exemplo, o idioma utilizado em uma mídia de texto.

No caso do NCLua, a semântica dada a uma propriedade é também controlada pelo autor da aplicação. Uma possível abordagem é a de mapear cada propriedade a variáveis de mesmo nome nos scripts NCLua, ou então, a funções Lua que tratem de forma especial o valor sendo atribuido à propriedade.

No exemplo da seção anterior, nenhuma âncora é utilizada na definição do elo. Quando isso acontece, o formatador NCL (agente do usuário NCL) deve assumir que o relacionamento se faz com o objeto de mídia inteiro (com a âncora especificando todo o conteúdo).

Um exemplo mais complexo pode ser construído tendo por base o exemplo anterior. Imagine que quando um determinado personagem aparecer no vídeo, deseja-se incrementar um contador no script NCLua.

```
<media id="videoId" src="video.mpg">
    <area id="personagem" begin="2s" end="10s"/>
</media>

<media id="scriptId" src="script.lua"/>
    <property name="contador" value="0"/>
</media>

<link>
    <bind role="onBegin" component="videoId"
          interface="personagem"/>
    <bind role="set" component="scriptId"
          interface="contador">
        <bindParam var="1"/>
    </bind>
</link>
```

O código NCL acima cria um relacionamento entre o aparecimento do personagem (aos 2 segundos do vídeo) e a chamada `set` sobre a propriedade `contador` do NCLua (apresentada na próxima seção).

Em NCL cada âncora (área ou propriedade), incluindo a área que representa o objeto de mídia como um todo, possui uma máquina de estados associada, conforme apresentado na Figura 1.

Os elos indiretamente relacionam transições nas máquinas de estado das âncoras. No exemplo, a transição `start` da máquina associada à área `personagem` aciona a transição `start` da máquina associada à propriedade `contador`. O formatador identifica as transições pelo atributo `role` do elemento `<bind>`.

Para informações mais detalhadas sobre as máquinas de estado NCL, a norma ABNT referente ao padrão brasileiro de TV Digital [12] deve ser consultada.

5. NCLUA - OBJETOS IMPERATIVOS LUA

A Seção 4 apresentou a abstração criada por NCL para seus objetos de mídia, tratando-os de forma unívoca. Esta seção volta-se para o conteúdo específico de um objeto NCLua, ou seja, os arquivos de extensão `.lua` utilizados no parâmetro `src` dos elementos `<media>`.

Scripts NCLua seguem sintaxe idêntica a de Lua e possuem um conjunto de bibliotecas similar. Ou seja, a linguagem utilizada é Lua, que apenas foi adaptada para funcionar embutida em documentos NCL.

5.1 Bibliotecas NCLua

Além do conjunto padrão de bibliotecas (com algumas exceções), quatro novos módulos estão disponíveis para os NCLua:

1. módulo `event`: permite que aplicações NCLua se comuniquem com o ambiente através de eventos;
2. módulo `canvas`: oferece uma API para desenhar primitivas gráficas e imagens;
3. módulo `settings`: exporta uma tabela com variáveis definidas pelo autor do documento NCL e variáveis de ambiente reservadas;
4. módulo `persistent`: exporta uma tabela com variáveis persistentes, que estão disponíveis para manipulação apenas por objetos imperativos.

Um fato comum entre esses novos módulos é o de serem extremamente simples. Eles não tentam de forma alguma definir um modelo de programação ou esgotar em APIs todo possível tipo de uso. O que eles oferecem, de maneira geral, são acessos a primitivas disponíveis em qualquer sistema computacional.

Algumas vantagens dessa abordagem são:

1. Não define uma forma amarrada de desenvolvimento de aplicações, que poderia não fazer sentido com a evolução do padrão e do mercado.
2. A definição é pequena, sucinta e de fácil entendimento. Atualmente são aproximadamente 30 novas funções ou tipos (uma ordem de grandeza abaixo das APIs do MHP/GEM ou ECMAScript).
3. A implementação do adaptador é bastante direta, com um mapeamento próximo de 1:1 com o sistema de implementação do *middleware*, portanto pequena e com pouca margem a comportamentos não padronizados.

5.2 Ciclo de Vida do NCLua

A principal peculiaridade do NCLua está no fato de que o seu ciclo de vida é controlado pelo documento NCL que o referencia. Isso acarreta num modelo de execução bem diferente de quando um script Lua é executado pela linha de comando em um computador pessoal, por exemplo.

Como fica mais claro a seguir, o modelo de execução de um NCLua é orientado a eventos, isto é, o fluxo da aplicação é guiado por eventos externos oriundos do formatador NCL. Um script NCLua nada mais é que um **tratador de eventos**.

Isto quer dizer que, para todos os usos de controle remoto, transmissões pelo canal de interatividade, sincronismos com o documento NCL, etc., existem eventos associados e é através deles que toda dinâmica de um NCLua se realiza.

O acionamento de eventos ora é feito pelo formatador NCL que comanda o NCLua, ora é feito pelo NCLua que sinaliza ao formatador uma mudança interna. Esse comportamento caracteriza uma ponte de comunicação bidirecional entre o formatador e o NCLua.

A seqüência abaixo é válida para qualquer objeto de mídia, mas está particularizada para objetos NCLua. Ela é determinante para o entendimento do modelo de execução:

1. O formatador NCL identifica nos elementos `<media>` o objeto NCLua e os relacionamentos em que participa.
2. O formatador infere o momento da aplicação em que o NCLua irá participar e o carrega (a chamada *fase de pré-carregamento* no plano de escalonamento do formatador [10]). Nesse momento o NCLua entre em modo orientado a eventos.
3. O formatador envia ao NCLua os eventos correspondentes aos relacionamentos em que participa, por exemplo, o `start` definido no elo do exemplo na Seção 4.1.
4. Enquanto houver alguma âncora do NCLua que não esteja no estado `sleeping`, o NCLua permanece vivo respondendo a eventos.
5. Quando todas as âncoras estiverem no estado `sleeping`, o formatador pode destruir o NCLua, mas não é obrigado a fazê-lo.

O script é inteiramente carregado no segundo passo. Sendo um tratador de eventos, o código do NCLua deve apenas definir variáveis e funções, além de chamar uma função especial para registrar um tratador de eventos para comunicação posterior com o formatador. A partir de então, qualquer ação tomada pela aplicação será somente em resposta a um evento enviado pelo formatador a essa função tratadora.

Durante o período do quarto passo, o NCLua pode responder a outros eventos, tanto do documento NCL, quanto de outras fontes, tais como rede e teclado. Nesse período, o NCLua, em execução, pode também sinalizar eventos internos ao formatador, tais como a mudança em uma de suas propriedades, ou até mesmo o seu fim natural.

O último passo não necessariamente destrói (remove da memória) o NCLua [2]. Por exemplo, pode-se deixar o objeto residente, para evitar a sua recriação posterior ao receber um novo `start`.

É essencial que tanto a iniciação do NCLua quanto os tratadores de eventos sempre executem rapidamente, pois nesse período nenhum evento é processado pelo NCLua. Para tanto, as aplicações NCLua fazem uso intenso da API de eventos, que possui apenas chamadas assíncronas (não bloqueantes).

5.3 O Módulo event

Dentre os módulos que foram introduzidos no ambiente dos NCLua, o mais importante é o módulo `event` uma vez que está diretamente ligado com a ponte entre o documento NCL e o script NCLua.

[2]O comportamento desejado pode ser especificado pelo atributo *playerLife* do descritor.

Como já mencionado, a comunicação entre o formatador e um NCLua é bidirecional, ou seja, tanto o NCLua quanto o formatador enviam e recebem eventos.

Para que um NCLua receba eventos oriundos do formatador é necessário que ele registre uma função tratadora através de uma chamada à `event.register`:

```
function handler (evt)
    -- codigo para tratar os eventos
end
event.register(handler)
```

A função a ser registrada deve possuir um único parâmetro, o evento sendo recebido do formatador (`evt` na assinatura de `handler`).

No sentido contrário, um NCLua deve fazer uma chamada à função `event.post` para enviar um evento ao formatador NCL, passando como parâmetro o evento criado:

```
evt = { ... } -- definicao do evento
event.post(evt)
```

Os eventos são tabelas Lua simples cujo campo `class` identifica a classe do evento. As seguintes classes são definidas: ncl, key (acesso ao controle remoto), tcp (conexão à internet), sms (envio e recebimento e mensagens de texto) e user (eventos customizados).

A classe de eventos `ncl` trata dos eventos relacionados à ponte entre o NCL e o NCLua. Para essa classe, os seguintes campos também estão presentes:

1. `type`: Assume `presentation`, para âncoras de conteúdo (áreas) ou `attribution`, para âncoras de propriedades.
2. `action`: Pode assumir um dos valores das transições da máquina de estado da Figura 1: `start`, `stop`, `pause`, `resume`, `abort`.
3. `area` ou `property`: O nome da área ou propriedade dependendo do valor em `type`.
4. `value`: Caso `type` seja `attribution`, assume o valor da propriedade no campo `property`.

As informações acima são suficientes para um NCLua identificar e dar uma semântica apropriada para suas âncoras.

Abaixo um exemplo simples de um NCLua que sinaliza o seu fim assim que é iniciado:

```
function handler (evt)
    if (evt.class == 'ncl') and
       (evt.type == 'presentation) and
       (evt.action == 'start') then
        evt.action = 'stop'
        event.post(evt)
    end
end
event.register(handler)
```

O complemento para o exemplo na Seção 4.2, um NCLua que controla a sua propriedade `contador`, é como a seguir:

```
local cur = 0
function handler (evt)
    if (evt.property == 'contador') and
       (evt.action == 'start') then
        cur = cur + evt.value -- atualiza o valor
        evt.action = 'stop'   -- fim da atribuicao
        evt.value  = cur      -- com novo valor
        event.post(evt)       -- sinaliza o fim
    end
end
event.register(handler)
```

Para a listagem completa das funções e classes de eventos do módulo `event` a norma ABNT [12] deve ser consultada.

6. EXEMPLO

Como exemplo ilustrativo da integração NCL-Lua, a aplicação detalhada a seguir simula um corrida entre dois atletas. O exemplo completo está disponível em [11] e pode ser executado na atual implementação do Ginga.

Ao iniciar a aplicação NCL, é exibida uma imagem com o texto "Largada!" que, ao ser selecionada pelo controle remoto, inicia a simulação.

Cada corredor é representado por um NCLua que o anima da esquerda para a direita, até que o fim de sua região na tela seja alcançado. Nesse momento, uma imagem correspondendo ao corredor é exibida na tela. A velocidade do atleta varia aleatoriamente com o tempo.

O código NCL para as mídias do exemplo é o seguinte:

```
<media id="go" src="go.png"/>
<media id="runner1" src="runner.lua">
    <area id="arr1" label="arrival"/>
</media>
<media id="runner2" src="runner.lua">
    <area id="arr2" label="arrival"/>
</media>
<media id="but1" src="but1.png"/>
<media id="but2" src="but2.png"/>
```

São definidas a imagem que inicia a simulação, os dois corredores e os dois botões que indicam a chegada dos corredores. O conteúdo de cada uma das mídias é definido em um arquivo separado e referenciado pelo atributo `src`.

Os atributos `descriptor` de cada mídia foram omitidos para poupar espaço, no entanto, são obrigatórios e definem como cada mídia é exibida (por exemplo, em que região da tela)[11].

Cada corredor é uma instância do script `runner.lua`, o que mostra a facilidade de definir novos componentes NCL totalmente escritos em Lua. Os corredores possuem uma área com `label="arrival"` para representar o trecho da animação em que o corredor alcança a linha de chegada. Nesse ponto, ainda não se conhece o código Lua, e nem é necessário. O programador NCL só precisa conhecer a interface exportada pelo NCLua (nesse caso, a área *arrival*).

Os relacionamentos entre os cinco objetos de mídia também são especificados em NCL, no código a seguir:

```
<link xconnector="onSelectionStopStart">
    <bind role="onSelection" component="go"/>
    <bind role="start"       component="runner1"/>
    <bind role="start"       component="runner2"/>
    <bind role="stop"        component="go"/>
</link>

<link xconnector="onBeginStart">
    <bind role="onBegin" component="runner1"
                         interface="arr1"/>
    <bind role="start"   component="but1"/>
</link>
<link xconnector="onBeginStart">
    <bind role="onBegin" component="runner2"
                         interface="arr2"/>
    <bind role="start"   component="but2"/>
</link>
```

Os atributos `xconnector` definem os conectores [9] usados nos relacionamentos e não são mostrados aqui, por problema de espaço [11].

O primeiro elo define que quando a imagem `go` é selecionada pelo controle remoto, os dois corredores são iniciados, além de esconder a própria imagem `go`. Os dois elos seguintes relacionam a chegada dos atletas (*arr1* e *arr2*) à exibição de seus botões correspondentes.

Esse exemplo caracteriza NCL como uma linguagem de cola, definindo as mídias da aplicação e seus relacionamentos no tempo.

O código NCLua dos corredores, escritos de forma totalmente independente da codificação em NCL, é descrito a seguir. Primeiramente, o script inicializa variáveis que serão usadas durante a sua execução, em particular, o objeto `runner` que representa o atleta durante a corrida:

```
-- dimensoes da regiao do NCLua
local DX, DY = canvas:attrSize()

-- objeto runner: guarda sua imagem,
--   frame, posicao e tamanho
local img = canvas:new('runner.png')
local dx, dy = img:attrSize()
local runner = { img=img, frame=0,
                 x=0, y=(DY-dy)/2,
                 dx=dx/2, dy=dy }
```

O objeto guarda a metade da dimensão (`dx=dx/2`) pois a imagem é, na verdade, uma tira com dois quadros do corredor (Figura 2).

Figura 2: Quadros do corredor.

A cada momento é exibido um quadro diferente, de modo a transmitir uma sensação de realidade na animação.

Em seguida é definida a função responsável por redesenhar a tela a cada passo da animação:

```
-- funcao de redesenho de cada ciclo de animacao
function redraw ()
  -- desenha o fundo
  canvas:attrColor('black')
  canvas:drawRect('fill', 0,0, DX,DY)

  -- desenha o corredor
  local dx = runner.dx
  canvas:compose(runner.x, runner.y, runner.img,
                 runner.frame*dx,0, dx,runner.dy)
  canvas:flush()
end
```

A função utiliza o objeto `runner`, que representa o corredor, inicializado anteriormente e cujos campos são atualizados nas definições a seguir.

O módulo `canvas` não foi detalhado nas seções anteriores, mas uma breve descrição dos métodos utilizados é apresentada:

canvas:attrSize(): Retorna as dimensões do canvas.

canvas:new(): Retorna um novo canvas com o conteúdo da imagem passada como parâmetro.

canvas:attrColor(): Define a cor utilizada em operações gráficas subseqüentes.

canvas:drawRect(): Desenha um retângulo de acordo com os parâmetros passados.

canvas:compose(): Desenha um canvas sobre outro de acordo com os parâmetros passados.

canvas:flush(): Atualiza o canvas após uma série de operações.

A cada passo da animação, a tela é pintada de preto. A chamada à `canvas:compose()` desenha o corredor em sua posição e quadro corrente sobre o canvas principal. O quarto parâmetro da chamada define o deslocamento feito na imagem para desenhar o quadro correto.

O próximo passo é dar vida à animação, o que deve ocorrer quando o NCLua é iniciado, no tratador de eventos:

```
-- funcao de tratamento de eventos
function handler (evt)
  -- a animacao comeca no *start* e eh realimentada
  --   por eventos da classe *user*
  if (evt.action == 'start') or
     (evt.class == 'user') then
    local now = event.uptime()

    -- movimenta o corredor
    if evt.time then
      local dt = now - evt.time
      runner.x = runner.x + dt*math.random(1,7)/100
    end

    -- muda o frame do corredor a cada 5 pixels
    runner.frame = math.floor(runner.x/5) % 2

    -- caso nao tenha chegado a linha de chegada,
    --   continua dando ciclos a animacao
    if runner.x < DX-runner.dx then
      event.post('in', { class='user', time=now })
    else
      event.post('out', {
        class='ncl', type='presentation',
        area='arrival', action='start'
      })
    end

    redraw()
  end
end
event.register(handler)
```

A animação é iniciada pela chegada de um *start* (linha 5). Em seguida, o script não pode simplesmente entrar em *loop* para realizar os passos da animação, pois isso travaria a aplicação, conforme comentado na Seção 5.2. A realimentação da animação é, então, obtida com eventos da classe *user* (linha 21), postados para o próprio script (`'in'`) e repassados ao tratador assim que o sistema estiver novamente livre (após tratar os outros objetos de mídia, por exemplo).

A posição de cada passo da animação (linhas 10-13) é obtida pela posição anterior, acrescida de um valor aleatório que é multiplicado pelo tempo passado entre dois quadros (ou seja, entre dois eventos *user*). O quadro do corredor muda a cada 5 pixels percorridos (linha 16).

A cada passo, a função `redraw()` atualiza a tela. O processo se repete enquanto o corredor não atingir a linha de chegada (linha 20). Quando isso ocorrer, o script sinaliza esse fato ao documento através da área *arrival* (linhas 23-26).

O exemplo utiliza uma comunicação bidirecional entre o NCLua e o documento: o *start* inicial é acionado pelo documento, já a sinalização de *arrival* é feita pelo NCLua. Essa comunicação é acionada e capturada pelos elos do documento NCL, respectivamente através de ações (`role="start"`) e condições (`role="onBegin"`).

Uma forma alternativa de desenvolver essa aplicação é através do uso de uma propriedade que guarde a posição do corredor, atualizada pelo NCLua a cada passo da animação. O documento NCL deve, então, avaliar quando a posição do corredor ultrapassa a linha de chegada (através de um elo `onEndAttribution`). Essa abordagem, apesar de mais custosa em termos de processamento, leva mais informação para o documento NCL, que pode, por exemplo, avaliar durante a animação quem está liderando a prova.

Uma característica da API NCLua, evidenciada por esse exemplo, é a inexistência de *threads* ou mecanismos de concorrência preemptiva. [3] A complexidade e não-portabilidade das APIs de *threads*, assim como a equivalência de funcionalidade obtida com o uso de eventos da classe *user* ou chamadas à função `event.timer()` [4], são os principais motivos pelos quais *threads* não foram adotados.

7. CONCLUSÃO E TRABALHOS FUTUROS

A integração entre NCL e Lua apresentada neste trabalho difere das adotadas nas linguagens XHTML (e suas derivadas), SVG e SMIL. Nelas, ECMAScripts são comumente misturados com código declarativo, dificultando a separação de tarefas entre equipes com diferentes perfis. Além disso, código ECMAScript têm acesso irrestrito à estrutura dos documentos declarativos, podendo alterar o seu conteúdo indiscriminadamente.

Essa abordagem intrusiva foi evitada a todo custo na integração entre NCL e Lua. Scripts NCLua são encapsulados na abstração de objetos de mídia de NCL, a mesma que engloba textos, imagens, vídeos e outras mídias. Assim, a comunicação com o documento é feita através dos meios convencionais de NCL, por elos ancorados em áreas e propriedades de objetos. Os elos, por sua vez, definem a lógica das aplicações NCL, criando relacionamentos de causa e efeito entre os objetos de mídia.

Com a implementação atual da integração, já existem alguns caminhos a serem explorados:

1. Desenvolvimento de *frameworks*, *game engines*, etc., com utilidades diferentes sobre a simplificada (mas abrangente) API de NCLua.

2. Desenvolvimento de aplicações nativas, mas portáveis entre plataformas de TV Digital.

3. Desenvolvimento de novos adaptadores de mídia escritos puramente em Lua.

O primeiro caminho será seguido naturalmente por desenvolvedores de conteúdo. Cada *software house* procura ter seu próprio conjunto de bibliotecas de maneira a simplificar e padronizar o desenvolvimento de suas aplicações. Um exemplo útil é uma abstração para animações, bastante comuns em jogos.

O segundo caminho mostra a versatilidade da API que pode até mesmo substituir a linguagem de implementação do receptor no desenvolvimento de aplicações nativas. Ou seja, o uso de NCL/Lua no desenvolvimento de aplicações residentes. Além de ser mais fácil desenvolver em uma linguagem declarativa com o suporte de scripts, a aplicação passará a executar em qualquer plataforma, uma grande

[3] Cada NCLua pode executar em *threads* diferentes, isso depende da implementação do *middleware* e não é notado pelo programador.

[4] Quando utilizados em conjunto com co-rotinas de Lua tornam-se ainda mais poderosos e flexíveis

vantagem para fabricantes. Esse caminho já vem sendo seguido pelos desenvolvedores de aplicações bancárias.

O terceiro caminho permite criar uma abstração ainda mais poderosa do que um simples conjunto de bibliotecas ou *framework*. Pode-se criar um script NCLua que trate um certo conjunto de âncoras com comportamento pré-definido e documentado, o que nada mais é que um adaptador para uma nova mídia. Com a grande vantagem de poder evoluir com o tempo, já que é portável e empacotado junto da aplicação. Qualquer nova mídia pode ser assim facilmente incorporada ao Ginga, sem necessidade de padronização. A Seção 6 faz exatamente isso com o NCLua do corredor. Apesar de ter uma funcionalidade específica, esse script pode ser usado em outros documentos NCL, bastando que o autor conheça a interface exportada pelo NCLua.

NCL é uma poderosa linguagem de cola com um suporte rico a relacionamentos. O desenvolvimento de novas mídias escritas puramente em NCLua leva a aplicabilidade de NCL a um novo patamar.

Com a API disponível já foram implementadas aplicações de nível razoavelmente complexo como jogos 2D e aplicações de rede, além de diversos scripts de auxílio ao NCL de cunho específico. Isso mostra que os NCLua podem se tornar uma boa alternativa aos Xlets Java (os applets do mundo de TV Digital) e suprem as necessidades imperativas em dispositivos portáteis, onde é a única linguagem imperativa obrigatória.

Finalmente cabe um paralelo com aplicações feitas para *web*. No passado, o senso comum dizia que aplicações complexas seriam *applets* Java, fato que nunca se concretizou. Com o tempo, o uso do par HTML+JavaScript foi sendo disseminado e o recente aparecimento do AJAX e tecnologias relacionadas restringiu ainda mais o uso dos *applets* Java. O mesmo processo pode ocorrer com o uso de linguagens de scripts embutidas em ambientes declarativos de TV.

8. REFERÊNCIAS

[1] ARIB. Data coding and transmission specification for digital broadcasting, 2002.

[2] ATSC. Dtv application software environment - level 1 (dase-1). http://www.atsc.org/standards.html (acessado em 06/2008), march 2003.

[3] W. W. W. Consortium. Scalable vector graphics. http://www.w3.org/Graphics/SVG/ (acessado em 06/2008).

[4] W. W. W. Consortium. Smil 3.0 state. http://www.w3.org/TR/SMIL3/smil-state.html (acessado em 06/2008).

[5] W. W. W. Consortium. Synchronized multimedia. http://www.w3.org/AudioVideo/ (acessado em 06/2008).

[6] G. L. de Souza Filho, L. E. C. Leite, and C. E. C. F. Batista. Ginga-j: The procedural middleware for the brazilian digital tv system. *Journal of the Brazilian Computer Society*, 13(4):47–56, 2007.

[7] ETSI. Multimedia home platform (mhp) specification 1.1.1. http://www.etsi.org/ (acessado em 06/2008), june 2003.

[8] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. Lua — an extensible extension language. *Software Practice and Experience*, 26(6):635–652, 1996.

[9] D. C. Muchaluat-Saade, R. F. Rodrigues, and L. F. G. Soares. Xconnector: extending xlink to provide multimedia synchronization. In *ACM Symposium on Document Engineering*, pages 49–56, 2002.

[10] L. F. S. Romualdo Costa, Marcelo Moreno. Intermedia synchronization management in dtv systems. To be Published.

[11] F. Sant'Anna. Exemplo completo. http://www.telemidia.puc-rio.br/ francisco/nclua/artigo.

[12] L. F. G. Soares et al. *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*. SBTVD Forum, 2008. Norma ABNT NBR 15606-2:2007.

[13] L. F. G. Soares and R. F. Rodrigues. Nested context model 3.0 part 1 - ncm core. Technical report, Departamento de Informática - PUC-Rio, 2005.

[14] L. F. G. Soares and R. F. Rodrigues. Nested context language 3.0 part 8 - ncl digital tv profiles. Technical report, Departamento de Informática - PUC-Rio, october 2006.