# NCLua - Objetos Imperativos Lua na Linguagem Declarativa NCL


- **Francisco Sant'Anna**
- Renato Cerqueira
- Luiz Fernando Gomes Soares


Laboratório Telemídia

PUC-Rio

# Introdução

- NCL – Linguagem Declarativa
- Necessidade de uma linguagem de script auxiliar
- Objetivo: Integração não intrusiva
- Objetos NCLua

# Requisitos

1. As linguagens devem ser alteradas o mínimo possível.

2. Deve ser mantida uma fronteira bem delineada entre os dois modos de programação.

3. A relação entre os dois ambientes deve ser ortogonal.

# Trabalhos Relacionados

- XHTML + ECMAScript
- SMIL

# XHTML + ECMAScript

1. Em XHTML não existe uma abstração única para objetos ECMAScript.

2. Código ECMAScript é escrito dentro de documentos e, até mesmo, atributos XHTML.

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

# XHTML + ECMAScript

- Fronteira tênue:

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

- Efeitos colaterais:

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

# SMIL

- Linguagem com propósitos similares a NCL
- Versão 2.1 - não possui suporte a scripts
- Versão 3.0
  - Módulo *State (expressões em XPath)*
  - *Python* previsto

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

# NCL – Visão Geral

- Separação entre conteúdo e estrutura
- Foco no sincronismo entre mídias: **elos**
  - Definição em separado – sintaxe própria
  - Independente do tipo de mídia

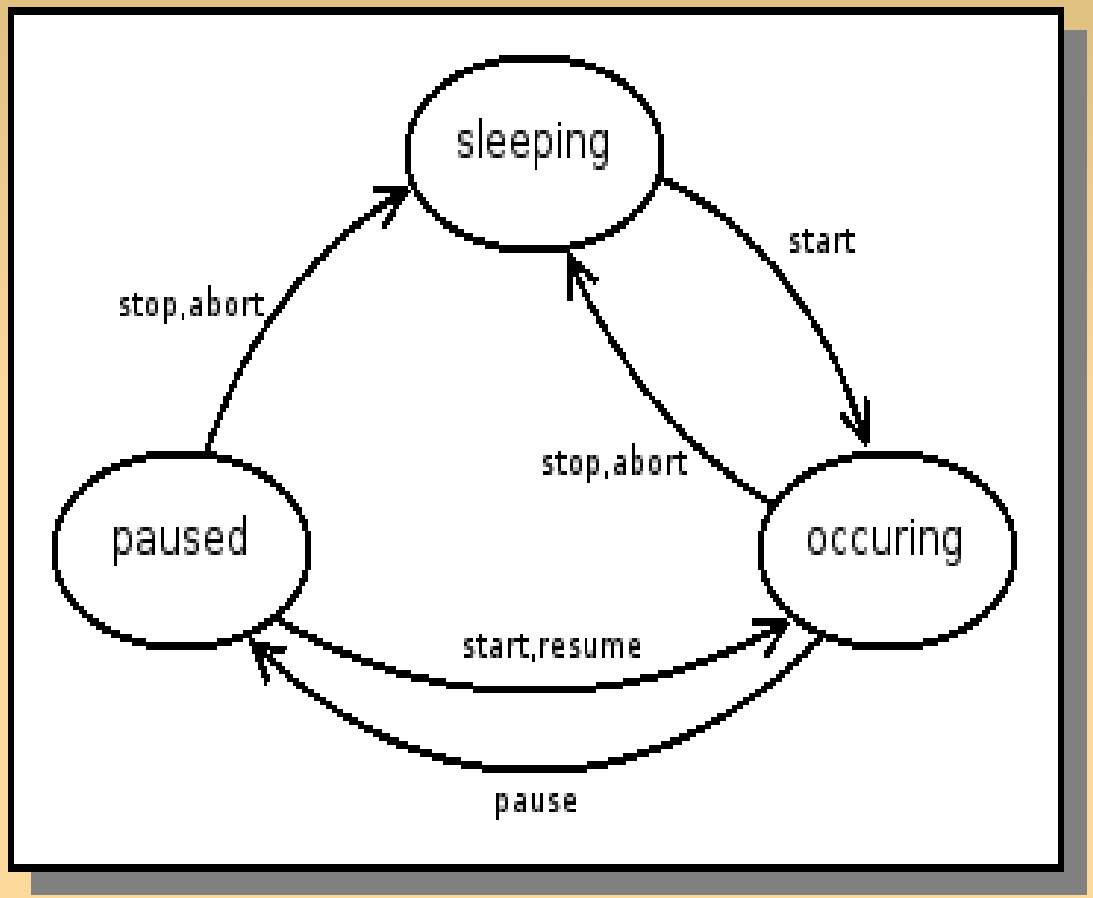
```
<media id="myvideo" src="video.mpg"/>
<media id="mynclua" src="script.lua"/>
<link>
    <bind role="onBegin" component="myvideo"/>
    <bind role="start"   component="mynclua"/>
</link>
```

# NCL – Visão Geral

- Objetos se relacionam por suas âncoras
  - Âncoras de conteúdo e propriedade

```
<media id="myvideo" src="video.mpg">
    <area id="personagem" begin="2s" end="10s"/>
</media>
<media id="mynclua" src="script.lua"/>
    <property name="contador" value="0"/>
</media>
<link>
    <bind role="onBegin" component="myvideo"
                         interface="personagem"/>
    <bind role="set" component="mynclua"
                     interface="contador">
        <bindParam var="1"/>
    </bind>
</link>
```

# NCL – Visão Geral




Máquina de Estados NCL

- Âncoras de conteúdo:
  - onBegin, onEnd, onPause
  - start, stop, pause
- Âncoras de propriedade:
  - onBeginAttribution, onEndAttribution
  - set

```
<link>
   <bind role="onBegin" component="myvideo"
                        interface="personagem"/>
   <bind role="set" component="mynclua"
                        interface="contador"/>
</link>
```

# NCLua – Objetos Imperativos

- Objetos de mídia onde *src=”*.lua”*
  - `<media id="..." src="myscript.lua">`
- Semântica das âncoras definidas pelo programador
- Ciclo de vida controlado pelo documento NCL
  - *event-driven*
- Bibliotecas extras:
  - Módulo *event*
  - Módulo *canvas*

# NCLua – Ciclo de Vida

1. O NCLua e os elos em que participa são identificados.
2. Em algum momento o NCLua é carregado, entrando no modo orientado a eventos.
3. O NCLua permanece vivo, recebendo os eventos, enquanto pelo menos uma de suas âncoras não estiver no estado *sleeping.*
4. Quando todas suas âncoras estiverem no estado sleeping, o NCLua é destruído.

# NCLua – módulo *event*

- Fundamental para a ponte NCL-Lua
- Sentido NCL → Lua

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

- Sentido Lua → NCL

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

# NCLua – módulo *event*

- Classes de Eventos:
  - ncl
  - edit
  - key
  - tcp
  - sms
  - si
  - user
  - ...

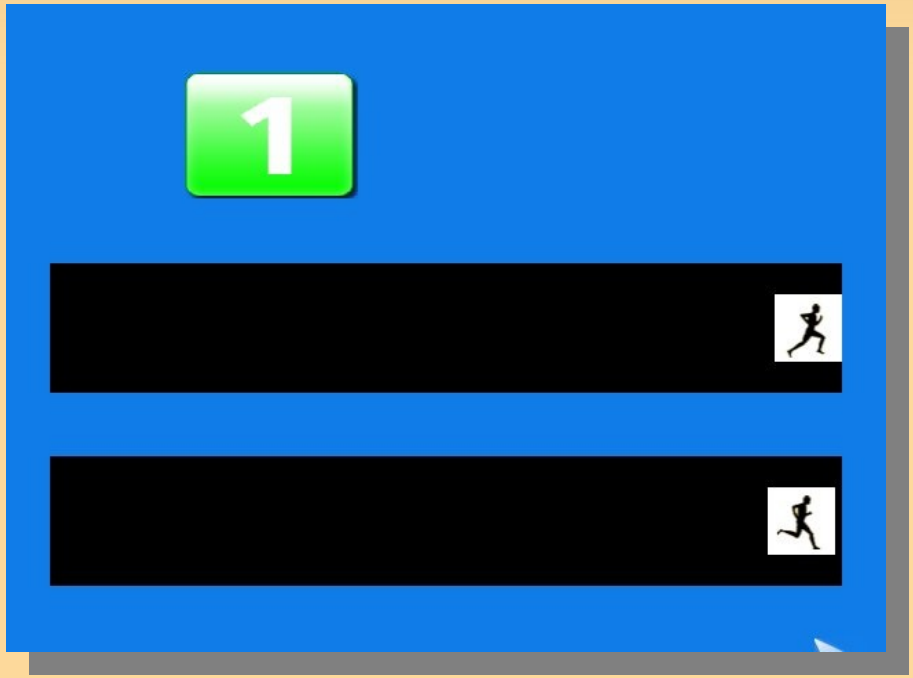
# NCLua – Classe *'ncl'*

- Campos: *type*, *action*, *area/property*, *value*

```
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)
```

# NCLua - Exemplo

- Corrida entre dois atletas.
- Cada um é representado por um NCLua.
- Ao chegar ao final, uma imagem correspondente é mostrada.
- Comunicação nos dois sentidos.



# NCLua - Exemplo

```
<body>
    <port id="entryPoint" component="go"/>

    <media id="go"   src="go.png"   descriptor="dsGo"/>
    <media id="but1" src="but1.png" descriptor="dsBut1"/>
    <media id="but2" src="but2.png" descriptor="dsBut2"/>

    <media id="runner1" src="runner.lua" descriptor="dsRunner1">
        <area id="arrival"/>
    </media>
    <media id="runner2" src="runner.lua" descriptor="dsRunner2">
        <area id="arrival"/>
    </media>

    <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="arrival"/>
        <bind role="start"   component="but1"/>
    </link>
    <link xconnector="onBeginStart">
        <bind role="onBegin" component="runner2" interface="arrival"/>
        <bind role="start"   component="but2"/>
    </link>
</body>
```

# NCLua - Exemplo

```
-- dimensoes da regiao 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 }

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

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

event.register(handler)
```

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

    -- movimenta o corredor caso tempo ja tenha
    -- passado
    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
```

# Trabalhos Futuros

- Desenvolvimento de frameworks, game engines, etc., com utilidades diferentes sobre a API de NCLua.
- Desenvolvimento de aplicações nativas, mas portáveis entre plataformas de TV Digital.
- Desenvolvimento de novos componentes de mídia escritos puramente em Lua.

# Conclusão

- Abordagem intrusiva evitada a todo custo
  - NCLua usa a abstração <media>
  - Comunicação através das tags <link>
  - API de eventos é extensível
  - Separação total de código
- Diversas aplicações desenvolvidas:
  - Jogos 2D, Aplicações de Rede, ”Calculadora”
- Alternativa viável aos *XLets*

# FIM

- Perguntas?
- Obrigado!


Laboratório Telemídia

PUC-Rio