UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Departamento de Ciência da Computação

Desenvolvimento de aplicações para TV Digital: Um Estudo de Caso

Ivan de Oliveira Rios

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Prof. Marcelo Lobosco

JUIZ DE FORA - MG
DEZEMBRO, 2009

Desenvolvimento de aplicações para TV Digital: Um Estudo de Caso

Ivan de Oliveira Rios

Monografia submetida ao corpo docente do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora, como parte integrante dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Marcelo Lobosco

JUIZ DE FORA - MG
DEZEMBRO, 2009

Desenvlvimento de Aplicalções para TV Digital: Um Estudo de Caso

Ivan de Oliveira rios

Monografia submetida ao corpo docente do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora, como parte integrante dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 17 de dezembro de 2009.

Comissão Examinadora:

_____

Marcelo Lobosco, DSc (orientador).

_____

Eduardo Barrére, DSC

_____

Eduardo Pagani, MSc

JUIZ DE FORA
DEZEMBRO, 2009

Resumo

Este trabalho tem por objetivo realizar um estudo do padrão brasileiro de TV Digital. Em especial, é apresentado o *middleware* Ginga, desenvolvido pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). O trabalho também apresenta o módulo orientado a eventos NCLua, adaptação da linguagem Lua, também criada pela PUC-Rio. Por fim, uma aplicação que envolve conceitos de Ginga-NCL e NCLua é apresentada.

Palavras Chave: Interatividade, Ginga-NCL e NCLua

Agradecimentos

Primeiramente agradeço a Deus por nos dar forças para vencer os desafios que a vida nos traz e também a minha família, que sempre me apoiou em minha escolhas e nunca foi contra elas. Agradeço também ao meu Orientador Marcelo Lobosco pela paciência e por me ajudar a escolher este tema. Agradeço aos professores que me incentivaram ao longo do curso a não desistir do mesmo e aos meus amigos e namorada que sem eles a vida não teria graça. Um muito obrigado a todos.

Sumário

Lista de Figuras

Capítulo 1 – Introdução

O termo TV Digital foi introduzida há mais de oito anos no Brasil, porém o termo só começou a se popularizar nos últimos dois anos, mais precisamente no dia 02 de dezembro de 2007, quando iniciaram-se, em São Paulo, as transmissões na televisão aberta dos primeiros programas seguindo este novo padrão.

Pode-se dizer que na época em que foi lançada, o impacto da introdução da tecnologia digital na televisão foi, em alguns aspectos, semelhante ao da introdução da televisão em cores no país, realizada no início da década de 1970. Entretanto, diferentemente do que ocorreu na década de 1970, não é só a qualidade das imagens e do som que melhorou com a introdução da tecnologia digital. A interatividade com o telespectador é um dos principais recursos disponibilizados com a introdução desta nova tecnologia.

Assim, podemos dizer que os três principais pilares da TV Digital são: a) a melhoria da qualidade de imagem, b) a melhoria da qualidade do som, e c) a interatividade com o telespectador [2]. Sobre a qualidade de imagem, pode-se dizer que este sistema consegue uma alta fidelidade, reproduzindo assim o conteúdo transmitido de forma muito próxima daquela que foi transmitida. Já sobre o segundo pilar, a qualidade de áudio, pode-se dizer que a TV Digital possibilita transmissões de mais canais de sons: 5 ou até 7 canais podem ser transmitidos simultaneamente, por exemplo. Já a interatividade, num futuro próximo, irá permitir que programas televisivos possam ter a participação direta do telespectador. Além do mais, conteúdos extras sobre o programa em exibição estarão disponíveis ao telespectador.

Este trabalho tem por objetivo realizar um estudo do padrão brasileiro de TV Digital. Em especial, é apresentado o *middleware* Ginga, desenvolvido pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), que tem por objetivos:

- Tornar as aplicações desenvolvidas para TV Digital independentes do sistema operacional da plataforma de hardware utilizados, e
- Oferecer um melhor suporte ao desenvolvimento de aplicações para TV Digital.

E também é apresentado neste trabalho o módulo orientado a eventos NCLua, adaptação da linguagem Lua, também criada pela PUC-Rio. Uma aplicação que envolva os temas Ginga-NCL e NCLua será disposta ao final deste trabalho.

Este trabalho está assim organizado. O capítulo dois apresenta um breve resumo sobre o padrão brasileiro de TV Digital. O capítulo três aborda o *middleware* Ginga, que permite o desenvolvimento de aplicações para a TV Digital. O capítulo quatro apresenta a linguagem NCLua. Já no capítulo cinco são apresentadas três aplicações que utilizam os conceitos descritos neste trabalho. Por fim, o capítulo seis conclui o trabalho.

Capítulo 2 – A TV Digital

A história da TV Digital no Brasil inicia-se em 1994, quando um consórcio formado por emissoras de TV, empresas ligadas à área de telecomunicações e universidades começou a pesquisar os três sistemas de transmissão em formato digital até então existentes. Os primeiros testes de campo só foram realizados entre agosto de 1999 e março de 2000. Em 2003 o Governo Federal resolve tomar a frente da iniciativa para a implantação da TV Digital no Brasil. Com um investimento de oitenta milhões de reais dos cofres públicos [29], foi criado um grupo de estudo para estudar os padrões existentes, e propor a melhor alternativa a ser adotada pelo Brasil.

Após alguns anos de estudo, chegou-se no ano de 2006 a escolha de uma especificação para o padrão brasileiro de TV Digital. Este padrão foi desenvolvido e implantado, tendo as primeiras transmissões sido realizadas cerca de um ano após, em dezembro de 2007 [2]. Porém, nos últimos dois anos, pouca coisa se modificou. Pouco mais de duas dúzias de cidades contam hoje com a transmissão digital. Mesmo nestas cidades, não houve uma forte campanha publicitária explicando as vantagens do sistema de TV Digital. Além do mais, os próprios conversores digitais a venda no comércio brasileiro são caros e não implementam todas as funcionalidades do novo sistema de TV. Este somatório fez com que a população não aderisse em massa à TV Digital. Muitos usuários estão esperando o lançamento de novos produtos, mais baratos e que estejam completamente em conformidade com o novo padrão.

Porém, mesmo com todos os reveses, este é um tema que continua despertando grande interesse, pois é um mercado insipiente e com enorme potencial de crescimento. Em especial, no que tange a Ciência da Computação, muitas são as oportunidades na área de desenvolvimento e porte de aplicações computacionais para a TV Digital.

Como mencionado anteriormente, ainda sobre os padrões, para se chegar ao padrão brasileiro foram estudados três distintos padrões digitais adotados por três países/comunidades [10]:

- ATSC (Norte-Americano),
- DVB (Europeu),
- ISDB (Japonês).

Após realizados alguns testes, o Comitê de Desenvolvimento do Sistema Brasileiro de TV Digital (SBTVD) apresentou os resultados do estudo, com os prós e contras de cada

padrão. A concretização desses estudos foi coordenada pela Finep e pela Fundação Centro de Pesquisa e Desenvolvimento em Tecnologia (CPqD). Um sistema de TV Digital genérico foi dividido em áreas de conhecimento e um consórcio de empresas e universidades propôs formulação do modelo de referência, baseado em uma modificação do padrão Japonês [24].

Nesta modificação do padrão Japonês, nasce o padrão brasileiro, com transmissão de áudio e vídeo em alta qualidade: som digitalizado e vídeo em 720 linhas progressivas e, em alguns casos, 1080 linhas progressivas. Porém, a alta qualidade das imagens e do som é apenas um dos aspectos contemplados pelo padrão brasileiro. A interatividade com o telespectador também tem de ser destacada. Neste contexto, a interatividade é todo o processo de comunicação entre o telespectador e a emissora de televisão que ele assiste. Podemos citar como exemplos de interatividade:

Para programas esportivos:

- Consultar as regras da modalidade esportiva e do campeonato em disputa, a escalação das equipes que disputam um jogo, as estatísticas do jogo, a classificação do campeonato, dentre outras [11];
- Responder a enquetes, fazer perguntas aos apresentadores/comentaristas esportivos;
- Durante uma propaganda televisiva de um campeonato, comprar ingressos para um dos jogos ou mesmo o direito de assisti-los no sistema *pay-per-view*.

Para programas de debate:

- Responder à uma enquete sobre o tema discutido;
- Formular perguntas para os debatedores;
- Ser entrevistado, não somente via voz, mas também via vídeo, caso um microfone e/ou *webcam* estejam acoplados à sua TV ou decodificador.

Para programas jornalísticos:

- Dar sugestão para pautas;
- Consultar mais informações sobre as notícias vinculadas;

- Receber/consultar a previsão do tempo para a região onde está assistindo a programação.

Ou seja, um enorme leque de possibilidades que se abre com a TV Digital.

Porém, tais aplicações digitais devem ser especificadas e codificadas com cuidado, visto que existem diferenças entre as aplicações desenvolvidas para serem executadas em um computador e as desenvolvidas para TV digital. Algumas considerações devem ser observadas [15]:

- A transmissão de grande parte das informações é por difusão e não personalizada;
- Os dispositivos de interação (controle remoto) são ainda pobres em termo de expressividade e usabilidade;
- O vídeo principal é a fonte de sincronismo, incluindo a interação;
- Assistir a um programa é muitas vezes uma atividade coletiva, deve-se se pensar em como atrair a atenção de várias pessoas ao mesmo tempo;
- A TV é usada para lazer e o telespectador não quer nada complexo em seu uso.

Para o desenvolvimento de aplicações para a TV digital, um programador pode optar hoje por duas linguagens distintas, a Linguagem Procedural e a Linguagem Declarativa. A diferença entre as duas é que uma linguagem procedural é aquela que sua entidade inicial é do tipo conteúdo procedural, analogamente, uma aplicação declarativa é aquela que sua entidade inicial é do tipo conteúdo declarativo.

Linguagens declarativas são utilizadas por autores que não possuem conhecimento em linguagens de programação. Possuem um nível maior de abstração e são voltadas a um objeto específico. As linguagens declarativas mais comuns são: NCL (*Nested Conext Language*) [4], SMIL (*Synchronized Multimedia Integration Language*) [33] e XHTML (*eXtensible Hypertext Markup Language*) [34].

Outras linguagens que não são declarativas, buscam uma solução algorítmica do problema. As chamadas Linguagens Procedurais, a despeito do jargão utilizado na TV Digital, seguem alguns paradigmas de programação, como orientação a objetos e orientação a eventos, dentre outras. Porém esta linguagem requer uma maior especialização do programador do que as linguagens declarativas.

Capítulo 3 – Aplicações Digitais

3.1 – Introdução

Todo o programa produzido para a TV Digital deve seguir um padrão bem definido. Segundo Neto[21], este padrão é constituído de uma coleção de nós-de-mídia, formado por vídeo, áudio e imagem, conforme ilustra a Figura 1.

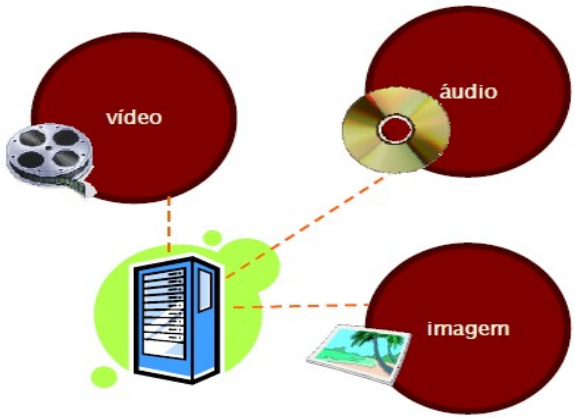

Figura 1 – Diagrama de aplicativos digitais [21]

Neto ainda define que os nós Vídeo e Áudio, como os próprios nomes já sugerem, irão se codificar o vídeo e o áudio da aplicação. Já o nó Imagem codifica pequenas animações ou imagens estáticas. A sincronização destes nós irá produzir a aplicação digital.

Os aplicativos digitais têm o funcionamento baseado em um ambiente cliente / servidor. O servidor para TV Digital é o provedor de conteúdo para o cliente, ou seja, para o telespectador. Para que tudo funcione perfeitamente, faz-se necessário todo um aparato de ferramentas e dispositivos que constituem o chamado *middleware* da TV Digital.

3.2 – Middleware

As aplicações a serem executadas no *middleware* podem ser classificadas em duas categorias, dependendo do tipo de conteúdo processado: declarativa ou procedural. A aplicação declarativa, como o próprio nome indica, apenas declara como conteúdos de vídeo, áudio e

imagens devem ser organizados. Um exemplo de aplicação declarativa é um documento que descreva a organização de conteúdo multimídia. O arquivo pode ser composto de símbolos de marcação, regras de estilo, *scripts*, imagens, áudio e vídeo. No caso de conteúdo procedural, pode-se definir uma aplicação que deverá ser executada na TV ou no S*et-up-box*[1]. É utilizado nas TVs que não possuem sintonizador digital. Um exemplo de aplicação procedural é um programa que agregue outros elementos multimídia, como gráficos, áudio e vídeo. Ambos os *middlewares* diferem portanto na forma de codificação, ou criação, dos aplicativos. No *middleware* procedural é usada a linguagem Java, enquanto no *middleware* declarativo é utilizada a linguagem NCL. Os *middlewares* declarativos são mais fáceis de serem criados, porém são menos flexíveis que os *middlewares* procedurais.

Qualquer que seja o *middleware* utilizado pela aplicação, faz-se necessário realizar a sua integração com a camada de transporte. Existem dois tipos de camadas de transporte: o Carrossel de dados e o Carrossel de objetos [14]. Na prática, pode-se dizer que o Carrossel nada mais é do que uma forma de disponibilizar arquivos, de forma cíclica, através do canal de difusão[14]. O Carrossel de dados contém apenas dados cujo conteúdo não é especificado ou identificado. Assim, cabe ao receptor decodificar os dados que recebe. Já o Carrossel de objetos contém dados que podem ser identificados, como, por exemplo, texto, imagens, ou aplicativos. O Carrossel de dados é, portanto, mais genérico. Ele pode, assim, receber os pacotes gerados pelo Carrossel de objetos, adaptando-os de modo a viabilizar a transmissão conforme ilustra a Figura 2.

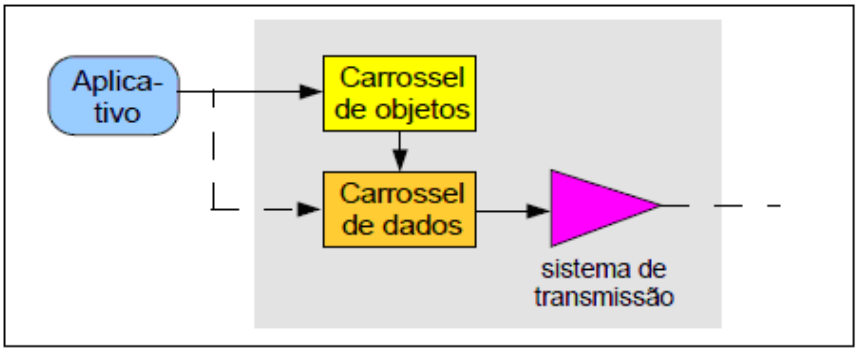

Figura 2 – Relação entre o Carrossel de dados e o de objetos [32]

1 Equipamento usado para conectar o televisor a uma fonte externa de sinal, transformando-o em conteúdo que possa ser apresentado na tela da TV.

Um mecanismo importante no sistema de TV Digital é o canal de retorno. Este canal permite que o telespectador transmita fluxos de dados ao servidor [22]. Através deste mecanismo o usuário consegue a interação com a programação que está sendo transmitida, permitindo-se o envio de uma mensagem do aparelho do cliente para a emissora. A escolha da tecnologia depende de cada radiodifusora. A norma sete do SBTVD prevê várias tecnologias que podem ser utilizadas, mas as tecnologias 3G e banda larga por cabo tem sido as escolhas mais comuns. "Nós não definimos um meio físico da interatividade. Nós definimos a camada de protocolo, a interface de comunicação com os dispositivos que suportarão a interatividade: ADSL, *dial-up*, 2,5G, 3G, e qualquer outra plataforma padronizada internacionalmente." (Roberto Franco, presidente do Fórum SBTVD) [22].

3.2.1 – Ginga

O *middleware* do sistema brasileiro de TV Digital foi desenvolvido pela PUC-Rio e chama-se Ginga. De acordo com site especializado [3]:

> *"Ginga® é constituído por um conjunto de tecnologias padronizadas e inovações brasileiras que o tornam a especificação de middleware mais avançada e a melhor solução para os requisitos do país."*[3]

Esta possui uma máquina de estados bem simples, compostas por 3 estados e 5 eventos. Cada transição é composta por um conjunto de um estado pré-evento, um evento e um estado pós-evento[27]. Esta configuração é apresentada na figura 3.

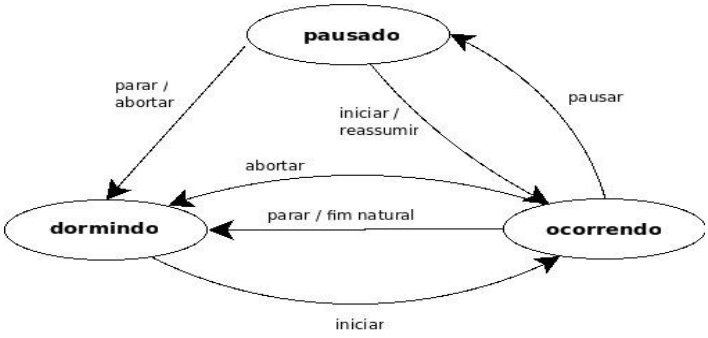

Figura 3 – Ciclo de vida das aplicações Ginga-NCL [27]

A arquitetura Ginga (figura 4) pode ser dividia em três módulos:

- o Ginga-CC (conjunto de exibidores monomídia),
- Ginga-NCL (aplicações declarativas NCL) e
- Ginga-J (aplicações procedurais em Java) [3].

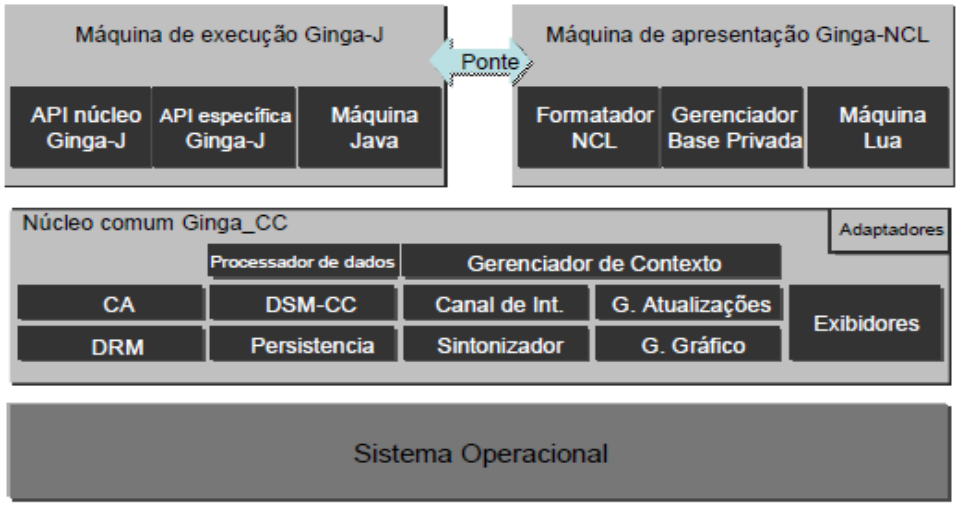

Figura 4 – Arquitetura Ginga [3]

3.2.1.1- Ginga Common Core

Este é o *middleware* de mais baixo nível dentre os três citados. Este concentra serviços necessários, para a máquina de apresentação (declarativo) quanto para a máquina de execução (procedural). Faz a interface com o Sistema Operacional. Neste também é realizado o acesso ao sintonizador de canal, ao sistema de arquivos, ao terminal gráfico, dentro outros. Também fica a seu cargo os exibidores de áudio, vídeo, texto, imagem e ajuda na comunicação das camadas superiores com o canal de retorno [12].

O *Ginga Common Core* é o subsistema lógico provedor de todas as funcionalidades comuns ao suporte dos ambientes declarativo (Ginga-NCL), e imperativo (Ginga-J) [12].

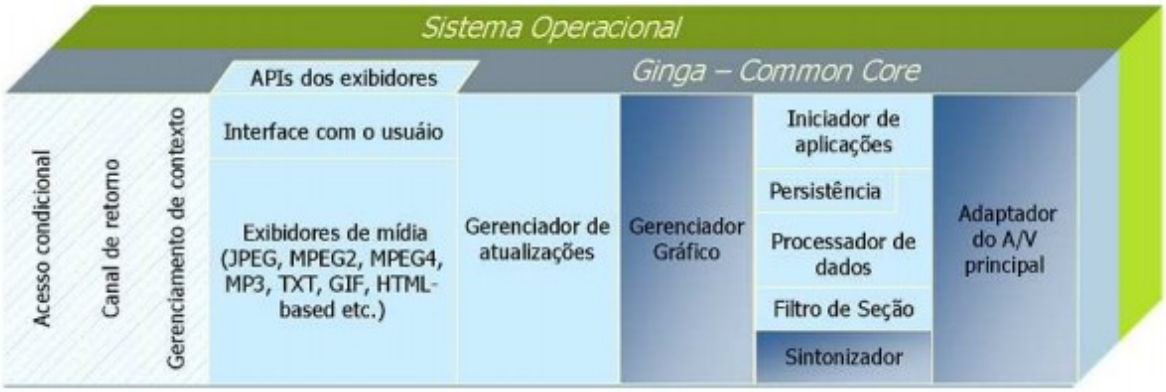

Figura 5 – Ginga-Common Core [18]

3.2.1.2 - Ginga-J

O ambiente imperativo Ginga-J utiliza a linguagem Java e possui uma ampla gama de pacotes que auxiliam o desenvolvimento das aplicações. Este é dividido em três módulos: a) a Máquina Virtual Java, b) APIs verde e c) APIs amarela e vermelha.

- Máquina Virtual Java é o coração do sistema, e abstrai as camadas de hardware e software do sistema hospedeiro. É o responsável pela execução das aplicações;
- APIs Verde: é composta pelo pacote de classes de transmissão de dados *Digital Audio Video Interactive Consortium* (DAVIC), pelo pacote de classes para interconexão de dispositivos *Home Audio/Video Interoperability* (HAVi) e pela API JavaTV da Sun;
- APIs Amarela e Vermelha: a primeira é composta pelo *Java Media Framework* (JMF), que são APIs que podem ser exportadas para outros sistemas. Elas provêem suporte a múltiplos usuários, a múltiplos dispositivos e a múltiplas redes. Já a vermelha é utilizada em aplicações brasileiras, em específico, utilizada em aplicações de inclusão digital.

As APIs amarela e vermelha, e a APIs da ponte entre a máquina de execução e apresentação (Figura 6) são inovações brasileiras do ambiente imperativo Ginga-J, que o distingue dos demais ambientes imperativos dos sistemas europeu e americano [26].

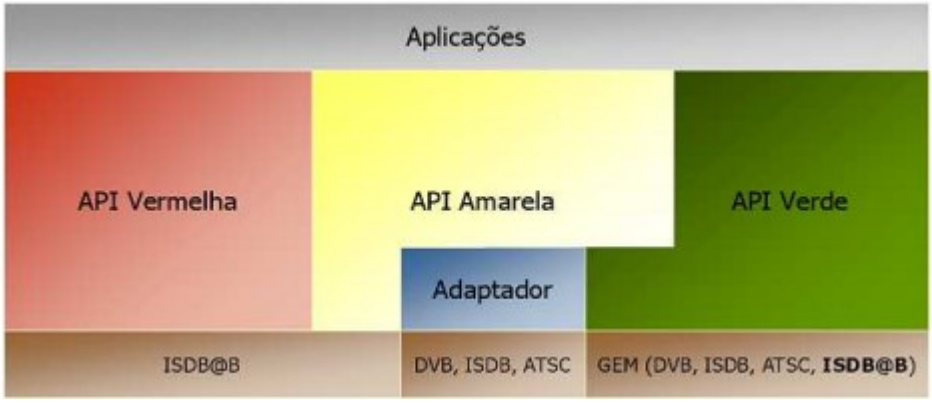

Figura 6 –Ginga-J [18]

3.2.1.3- Ginga-NCL

O Ginga-NCL, ou máquina de apresentação, é um subsistema lógico do Ginga que tem por objetivo exibir documentos no formato NCL. Portanto, a codificação é feita de forma muito parecida com o HTML, linguagem baseada em *tags* utilizada para descrever páginas na web. Apesar deste ambiente declarativo ter por base a linguagem NCL, pode-se utilizar também a linguagem de *script* Lua [31].

O NCL possui um controle não invasivo na ligação de um conteúdo e sua apresentação de leiaute, definindo uma separação bem demarcada entre estes. Com isso esta linguagem não define nenhuma mídia em si, ao contrário, ela define o que irá ligar as mídias em apresentações multimídia, não restringindo ou prescrevendo os tipos de conteúdo[31].

> *"A linguagem NCL está estruturada para organizar mídias. Dessa forma ela gerencia o comportamento de mídias no tempo e no espaço. Não é possível criar nada através da NCL; tudo deve ser desenvolvido em outros ambientes. A linguagem é responsável por colar essas mídias umas nas outras, o que determina o comportamento da aplicação através da atribuição de condições e ações. Assim, uma aplicação para TV digital pode ter apenas elementos de sincronismo das mídias, sem qualquer participação do usuário."* [5]

A manipulação destes nós de mídia é feita por via de descritores. Os descritores definem como serão exibidos e de que forma será conectado cada arquivo com alguma região na tela [5], como exemplificado na Figura 7.

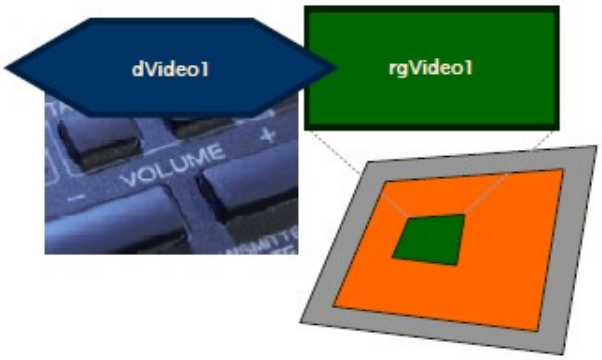

Figura 7 – Diagrama de conexão entre regiões e descritores [21]

Esta ilustra um descritor intitulado “dVideo1”, que conecta-se a algum nó de mídia à região intitulada “rgVideo1” fazendo o aspecto de colagem na tela do telespectador da região definida pelo programador.

O NCL baseia-se no modelo NCM (*Nested Context Model*) para descrever elementos de hipermídia, conforme exemplificado na Figura 8. Neste modelo o documento é organizado com um conjunto estruturado de nós e as ligações entre estes, os chamados elos. Cada nó pode, por sua vez, conter um subconjunto de nós e elos, e assim sucessivamente, formando uma hierarquia [31].

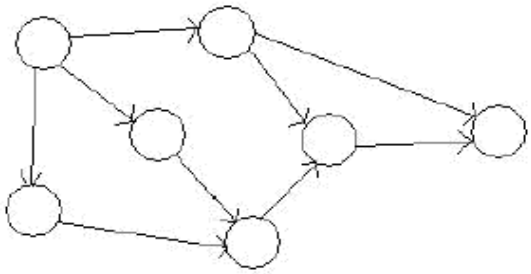

Figura 8 – Estrutura de um documento de Hipermídia [23]

Todos os documentos do tipo NCL possuem uma formatação semelhante ao código demonstrado como exemplo da figura 9 e da figura 10:

```
1:  <?xml version="1.0" encoding="ISO-8859-1"?>
2:
3:  <ncl id="exemplo01"
 xmlns="http://www.telemidia.puc-rio.br/specs/xml/NCL23/profiles"

 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.telemidia.puc-
                   rio.br/specs/xml/NCL23/profiles/NCL23.xsd">
4:  <head>
5:       <regionBase>
6:          <!-- regiões da tela onde as mídias são apresentadas -->
7:       </regionBase>
8:    <descriptorBase>
9:       <!-- descritores que definem como as mídias são apresentadas
 -->
10:   </descriptorBase>
11:   <connectorBase>
12:      <!-- conectores que definem como os elos são ativados e o que
 eles disparam -->
13:   </connectorBase>
```

Figura 9: Exemplo NCL [23]

```
14: </head>
15: <body>
16:   <port id="pInicio" component="ncPrincipal" interface="iInicio"/>
17:      <!-- contextos, nós de mídia, elos e outros elementos -->
18: </body>
19: </ncl>
```

Figura 10: Continuação Exemplo NCL [23]

Neste exemplo, o cabeçalho de um arquivo do tipo NCL é ilustrado pelas linhas de 1 a 3. O cabeçalho do programa se localiza na parte intermediária, entre as linhas 4 e 14 do documento. A *tag* de fechamento é ilustrada na linha 19.

3.3 – Composer

Composer é uma ferramenta de autoria hipermídia que foi desenvolvida pelo Laboratório TeleMídia do Departamento de Informática da PUC-Rio. Esta ferramenta possibilita o desenvolvimento de aplicações audiovisuais interativos sem necessitar de um

grande conhecimento da linguagem NCL. No Composer, é possível visualizar a sua aplicação de quatro formas distintas: [25]

- Visão estrutural: Esta visão permite a criação de aplicações através de elementos gráficos, de forma que podemos fazer edições em nós e elos visualmente;

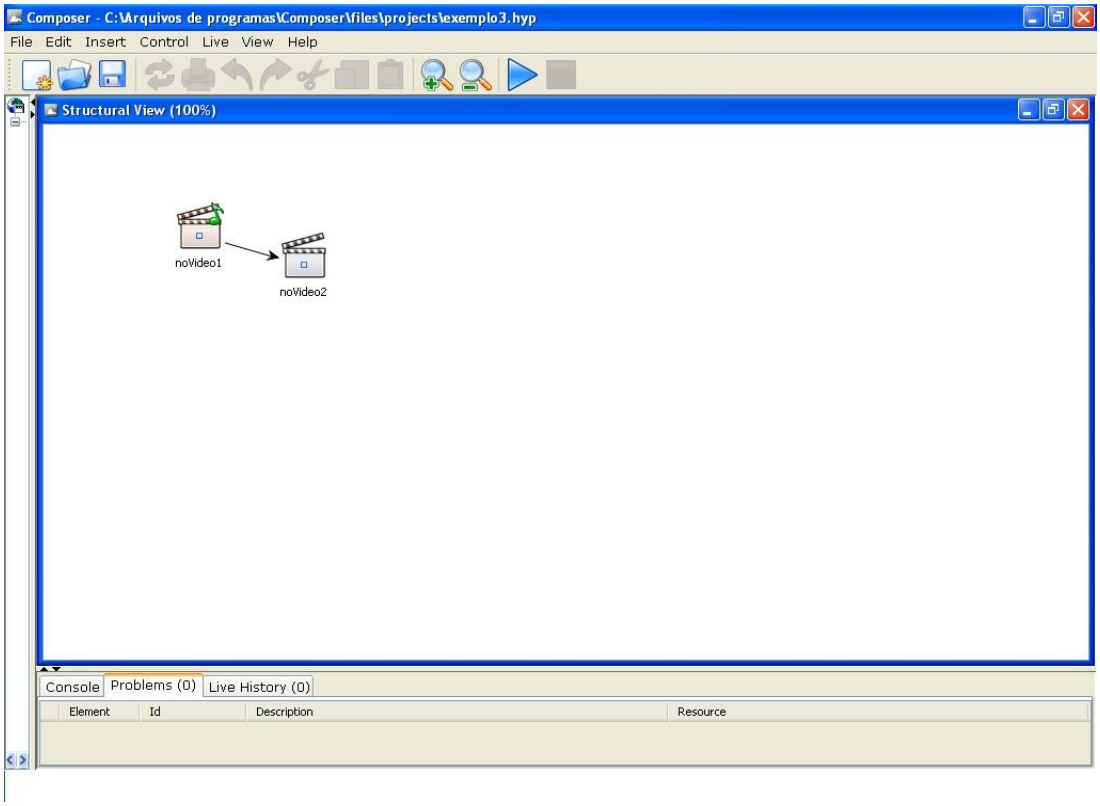

Figura 11 – *Composer*: Visão estrutural

- Visão temporal: Esta visão permite que o usuário passa realizar a sincronização temporal entre os nós de mídia que compõem a sua aplicação;

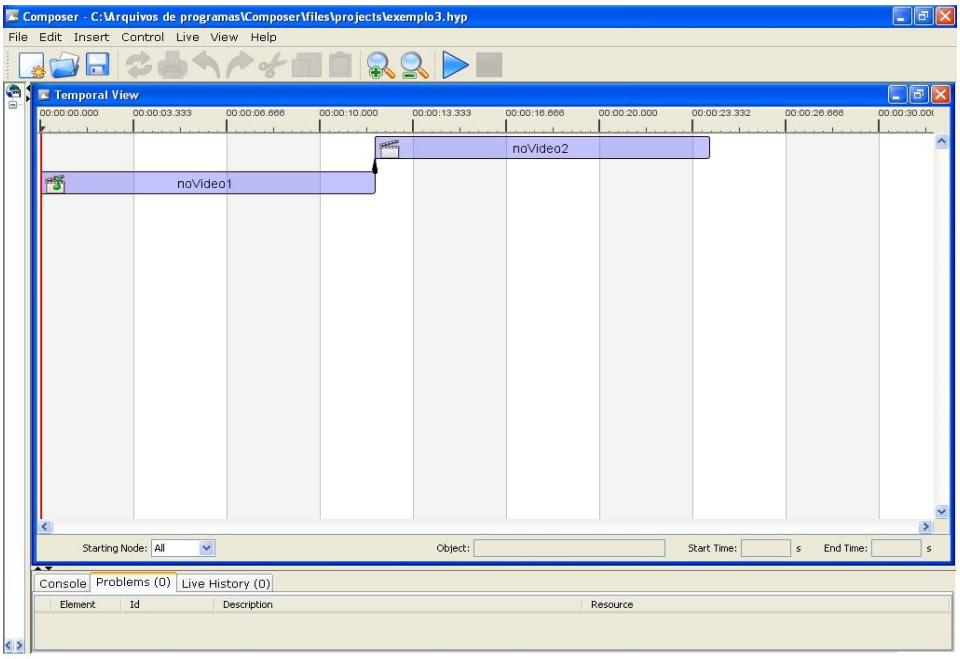

Figura 12 – *Composer*: Visão Temporal

- Visão de leiaute: Permite que o desenvolvedor crie o leiaute de sua aplicação;

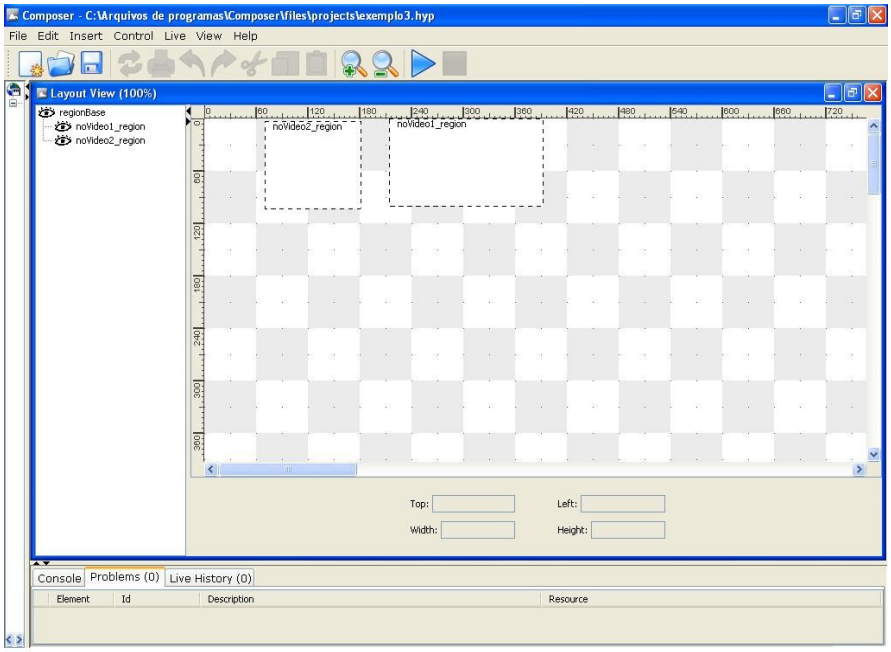

Figura 13 – *Composer*: Visão de leiaute.

- Visão textual: Apresenta o código-fonte NCL, permitindo que o programador faça edições na mesma.

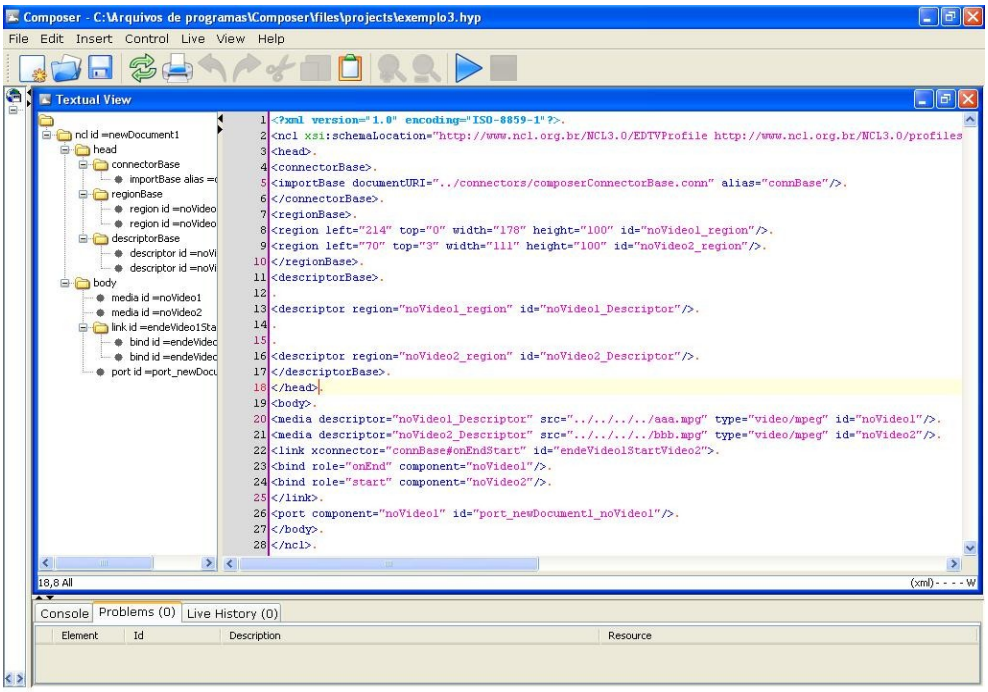

Figura 14 – *Composer*: Visão textual

Capítulo 4 - NCLua

4.1 – Introdução

Lua é uma linguagem de programação que começou a ser desenvolvida nos anos 1990. Foi desenvolvida na PUC-Rio pelo grupo Tecgraf (Tecnologia em computação gráfica) para ser usada com um projeto para a Petrobras. Lua é uma linguagem que possui tipagem dinâmica, sendo interpretada a partir de *bytecodes* para uma máquina virtual que conta com mecanismo de gerenciamento automático de memória. Empresas como a Blizzard Entertainment utilizam linguagem para a configuração de sistemas computacionais de alto desempenho, para atividades de automação (*scripting*), bem como para a prototipagem rápida de aplicações. Devido a sua grande leveza e eficiência, vem sendo empregada no desenvolvimento das mais distintas aplicações na área de computação, podendo ser citadas: controle de robôs, processamento de textos, jogos, dentre outras. Seu nome, Lua, remete a idéia de uma linguagem satélite, utilizada em conjunto com outras no desenvolvimento de aplicações [1].

4.2 – A Linguagem Lua

> "Lua é uma linguagem de *script* amplamente usada nas mais diversas áreas, desde grandes aplicativos para *Desktops*, como *Adobe Photoshop Lightroom*, até software para sistemas embarcados. Lua é a linguagem mais usada atualmente para *scripting* em jogos, e é parte do padrão Ginga para o Sistema Brasileiro de TV Digital. Lua também é muito usada na área de segurança" [17]

Como toda linguagem de *script*, esta não possui um método de entrada principal (*main*). Para executar os códigos que foram programados em Lua, estes devem ser invocados por outros códigos.

Lua é uma linguagem que é considerada simples, eficiente, de boa portabilidade e é livre. A escolha desta linguagem para o desenvolvimento de aplicativos para TV Digital é justificada por cada uma de suas qualidades [16]:

- Simples: trata-se de uma linguagem de fácil entendimento, bem como extremamente maleável;
- Eficiente: a linguagem é leve, excelente para dispositivos com recursos escassos, como os *set-top box* que funcionam como conversores das TVs Digitais;
- Portabilidade: o *middleware* foi desenvolvido para as mais distintas classes de dispositivos, de computadores à celulares e *set-top boxes*;
- Livre: não se faz necessário o pagamento de *royalties* a nenhuma empresa para a sua utilização.

4.3 – Orientação a Eventos

A linguagem Lua utiliza o paradigma de programação orientada a eventos. Neste paradigma o fluxo de execução da aplicação é guiado por atividades externas, chamadas de eventos. O uso deste paradigma é indicado para aplicações que contenham bastante interatividade com o usuário.

Ao contrário de outros paradigmas como, por exemplo, orientação a objetos, o *hardware* só executa um conjunto de instruções quando um determinado evento ocorrer, ou seja, um código é disparado em resposta à ocorrência de um determinado evento.

Os programas que utilizam o paradigma de orientação a eventos (POE) normalmente consistem de um conjunto de tratadores, que são os códigos que processam os eventos para produzir uma resposta, e de disparadores, que invocam os tratadores [9].

4.4 – Lua e a Linguagem NCL

> "Para se adequar ao ambiente de TV Digital e se integrar à NCL, a linguagem Lua foi estendida com novas funcionalidades. Por exemplo, um NCLua precisa se comunicar com o documento NCL para saber quando o seu objeto <media> correspondente é iniciado por um elo. Um NCLua também pode responder a teclas do controle remoto, ou desenhar livremente dentro da região NCL a ele destinada.

> Essas funcionalidades são específicas da linguagem NCL e, obviamente, não fazem parte da biblioteca padrão de Lua. O que diferencia um NCLua de um programa Lua puro é o fato de ser controlado pelo documento NCL no qual está inserido, e utilizar as extensões descritas a seguir.” [30]

O NCL define como os objetos de mídia são estruturados no tempo e no espaço. Como NCL funciona em conjunto com outras linguagens, ela funciona como uma espécie de linguagem de cola. Assim, ela não define nem restringe os tipos de objetos de mídia. Pode-se declarar objetos para execução, objetos de imagem, de vídeo, de áudio ou de texto, dentre outros. Cabe a linguagem definir mecanismos que permitam ao programador gerenciar essas mídias. Como as mídias não possuem qualquer comunicação com outra mídia dentro de um mesmo documento, o escopo de comunicação destas se restringirá aos seus elos, definidos no documento, e aos eventos gerados e que disparam a sua exibição [7].

Os *scripts* Lua recebem todas as regras comuns aos nós dos documentos NCL, como por exemplo, documentos de imagem, texto ou vídeo. Isso quer dizer que todo *script* Lua é comandado por um formatador NCL que define quando o nó irá começar, pausar ou parar. O exemplo a seguir, proposto por [7], ilustra a situação:

```
<media id = "scriptLua" src = "codigoLua.Lua" >
    <area id = "area" />
    <property name = "propriedade" [value = ""] />
</media>
```

Neste exemplo, os id´s referem-se aos nomes de cada nó. O *scriptLua* está ligado diretamente a um código externo denominado “codigoLua.Lua” que será visualizado em uma *area* seguindo as propriedades que foram definidas em *property*.

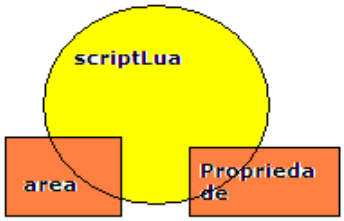

Figura 15 – Script Lua [7]

4.5 – Bibliotecas Lua

Além das bibliotecas convencionais, para que se tenha uma comunicação entre o NCL e os *scripts* Lua, há ainda a necessidade de utilização de quatro módulos responsáveis pela interligação entre as duas linguagens. Cada módulo tem um papel diferente quando utilizado; a escolha e a utilização de cada um depende do tipo de aplicação Lua que será implementada [30]:

- Módulo *canvas*: oferece uma API para desenhar primitivas gráficas e manipular imagens. Ao se inicializar um objeto do tipo NCLua, a região do elemento <media> pode ser acessada pela variável global *canvas* a partir do *script* Lua. Porém, se não for definido ou associado o objeto do tipo *canvas* a uma região, assume-se que este valor será *nil*, ou nulo, em linguagem Lua. Com este módulo pode-se obter a cor associada a um dado *pixel*, obter os pontos que definem a região que um dado texto ocupará no *canvas*, desenhar linhas, polígonos, eclipse ou texto;
- Módulo *settings*: exporta uma tabela com variáveis definidas pelo autor do documento NCL e variáveis de ambiente definidas no nó "*application/x-ginga-settings*". Estas são divididas em sub-tabelas, como, por exemplo, o *user* e suas variáveis: *Age*, *Location* e *Gender*;
- Módulo *persistent*: exporta uma tabela com variáveis persistentes, que estão disponíveis para manipulação apenas por objetos procedurais;
- Módulo *event*: permite que aplicações NCLua comuniquem-se com o *middleware* através de eventos [7].

4.6 – Orientação à Eventos com o NCLua

A Figura 16 ilustra o mecanismo de orientação à eventos em NCLua. Ao centro encontra-se o objeto NCLua e ao seu redor estão as entidades com as quais ele poderá interagir. A fim de interagir com o objeto NCLua, cada entidade externa insere um evento em uma fila do tipo FIFO, que o redireciona para as funções tratadoras de eventos definidas pelo próprio programador do *script* NCLua. Cada tratador irá processar um evento da fila seqüencialmente, deixando os demais em estado de espera. Cabe ao programador escrever tratadores que executem de forma eficiente, ou seja, é de sua responsabilidade evitar o congestionamento da fila. Um NCLua também pode se comunicar com entidades externas postando eventos dentro de seus tratadores, como mostram as setas saindo do NCLua [30].

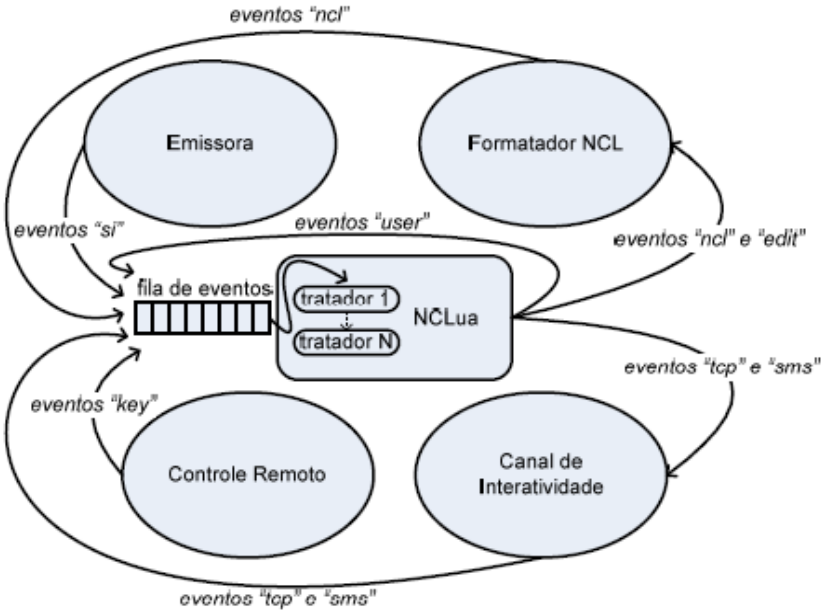

Figura 16 – Paradigma de programação orientado a Eventos [30]

4.6.1 - Módulo *event*

Por utilizar um paradigma orientado à eventos, Lua executa suas chamadas apenas uma vez, sendo assim o *script* denominado de tratador de eventos. Estes eventos são definidos na

linguagem NCL e são tabelas Lua, ou seja, a tabela é gerada para Lua a partir dos eventos definidos pelo programador NCL [30].

```
evt = {
      class = "ncl"
      type = "presentation"
      action = "start"
}
```

> "A tabela chamada *evt* com 3 campos, *class*, *type* e *action*, diz que a classe de eventos é *ncl*, que o tipo de evento foi de apresentação e a ação do formatador que desencadeou a apresentação foi um *start*." [7]

Os elementos da tabela *evt* são:

1. class. São cinco tipos de classes:

- *key*: Esta irá representar o posicionamento das teclas do controle remoto pelo usuário;
- *edit*: Irá permitir que comandos de edição sejam disparados a partir de *scripts* Lua;
- *ncl*: Esta classe é usada na comunicação de um NCLua com o documento NCL que contém o objeto de mídia;
- *user*: Por esta classe, as aplicações podem estender sua funcionalidade criando seus próprio eventos;
- *sms*: Usada apenas para troca de mensagens em dispositivos móveis;
- *tcp*: Utilização do protocolo TCP;
- *si*: Provê acesso a um conjunto de informações multiplexadas em um fluxo de transporte e transmitidas periodicamente por difusão.

2. type. Pode assumir os valores:

- *presentation* : Este tipo de evento controla a exibição de um nó NCLua. Este sempre estará ligado a nós ou âncoras. Âncoras podem ser iniciadas com a diretiva '*start*';
- *attribution*: Faz o papel do controle das propriedades do nó NCLua.

3. action. Pode assumir os seguintes valores: 'start', 'stop', 'abort', 'pause' e 'resume'.

Este *script* já cria a tabela Lua. Entretanto, para que Lua possa acessar a tabela faz-se necessária a declaração de uma função que irá tratar estes eventos, o tratador de eventos. Este tratador captura os eventos gerados a fim de ser informado quando os eventos externos são recebidos [30].

```
function tratadorEventos (evt)

    --código tratador
end

event.register(tratadorEventos)
```

Neste trecho de *script* foi declarada uma função que irá capturar todos os eventos gerados. A função declarada recebe como parâmetro uma tabela do tipo *evt*. Cabe ao código da função manipular a tabela que recebeu, por exemplo, identificando o tipo de evento e a ação que deverão ser executadas.

Todo NCLua tem a obrigação de registrar ao menos uma função de tratamento de eventos em seu corpo. Isto é feito através do *event.register*. É garantido que toda a operação de registro de eventos ocorrerá antes que o documento NCL gere qualquer tipo evento que possa disparar ações para o NCLua.

4.6.2 - Comunicação entre Lua e NCL

Após definido o evento que receberá tratamento, é hora de se definir como o *script* Lua irá se comunicar com um documento NCL.

A função que trata desta comunicação é a *post*:

```
event.post {
            class = "ncl"
            type = "presentation"
            action = "stop"
}
```

Com o uso desta função pode-se especificar, por exemplo, a interação entre objetos Lua, conforme exemplificado na Figura 17:

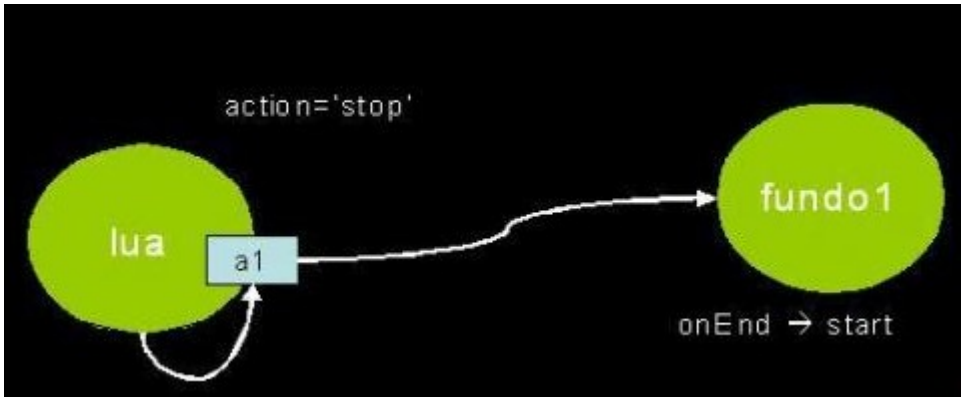

Figura 17 – Interação entre objetos NCLua [7]

O *script* se comunica com a âncora *a1* para que esta pare, *action n = stop*, com isto o formatador verifica que a âncora está parada. Consequentemente, o evento de fim de execução do *script* irá gerar uma ação de início de exibição de uma figura, *fundo1,* de plano de fundo [7].

Capítulo 5 – Construindo aplicações para TV Digital

Neste capítulo são apresentados três códigos exemplo de aplicações desenvolvidas para TV Digital. Os padrões escolhidos para a implementação foram o Ginga-NCL e o NCLua. Um dos códigos foi desenvolvido no escopo deste trabalho de conclusão de curso utilizando apenas Ginga-NCL, ou seja, sem que fosse acrescentada nada além do que a linguagem define por padrão. As duas outras aplicações que serão apresentadas foram desenvolvidas pela PUC-Rio. Ambas utilizam o Ginga-NCL em conjunto com NCLua e ilustram o modelo de iteração de objetos imperativos NCLua em documentos NCL.

5.1 – Primeiro Exemplo: Ciclo de Vida de Objetos NCLua

O objetivo deste exemplo é ilustrar, de modo simples e claro, a facilidade de iteração de objetos do tipo NCLua com imagens. Ela foi desenvolvida pela PUC-Rio.

Ao se iniciar esta aplicação, três objetos NCLua são iniciados:

- o primeiro executa indefinidamente;
- o segundo termina logo após ser iniciado;
- o terceiro finaliza a própria execução 3 segundos após ser iniciado.

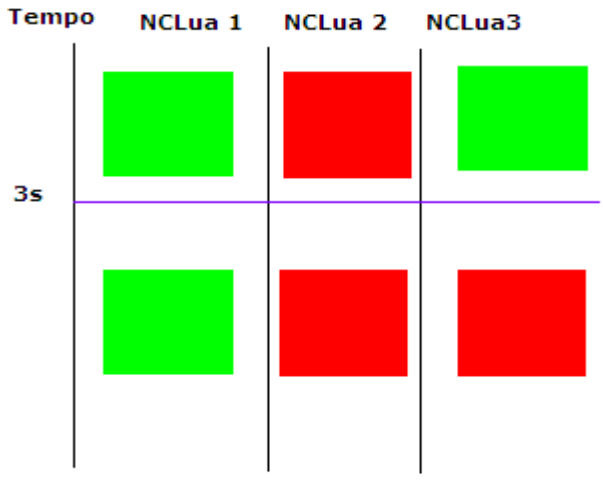

Figura 18 – Primeiro exemplo aplicação NCLua

Todos os objetos NCLua citados estão associados a duas imagens que possuem formato parecido com um botão, uma na cor verde e a outra na cor vermelha. A cor está associada ao estado de execução de um objeto NCLua: a cor verde indica que o objeto está executando, enquanto a vermelha indica que o objeto já encerrou sua execução. A Figura 18 ilustra a situação descrita.

A seguir, o código de cada objeto do tipo NCLua. O primeiro deles é um *script* sem codificação, ou seja, não foi proposto um tratador de eventos para o mesmo. A execução infinita em conjunto com o não tratamento dos eventos justificam o fato da figura verde estar em permanente exibição.

```
-- 1.Lua
-- vazio
```

O segundo código, diferentemente do primeiro, registra um tratador de eventos. Este NCLua finaliza sua execução imediatamente após ter sido iniciado, ou seja, logo após um documento NCL o chamar com o "*start*". Assim, o botão verde estará visível por um tempo extremamente curto, a ponto mesmo de não poder ser visto, mudando quase que imediatamente para a cor vermelha, que ficará em permanente exibição.

```
-- 2.Lua:
function tratador (evt)
  if (evt.class == 'ncl') and (evt.type == 'presentation')
and (evt.action == 'start') then
     event.post {
  class = 'ncl',
  type = 'presentation',
  action = 'stop' }
  end
end
event.register(tratador)
```

O código do terceiro *script* assemelhasse ao do segundo na medida em que também possui um tratador de eventos. Porém este tratador irá criar um temporizador de 3 segundos. Após este intervalo o estado do evento mudará de ativo para inativo, ou seja, indicará o fim do evento. O botão associado ao *script* ilustra essa situação: inicia-se em verde, mas depois de 3 segundos se torna vermelho.

Neste caso, deve-se notar que o evento que trata o temporizador, o *event.timer*, é dado em milissegundos e sua criação é feita no recebimento do evento de início. Nota-se também que a função logo após o temporizador posta um evento idêntico ao do segundo NCLua que indica seu fim natural.

```
-- 3.Lua:
function tratador (evt)
  if (evt.class == 'ncl') and
(evt.type == 'presentation') and
(evt.action == 'start') then


     event.timer(3000,
  function()
    event.post {
class = 'ncl',
type = 'presentation',
action = 'stop'
    }
end)
  end
end
event.register(tratador)
```

5.2 – Segundo Exemplo: NCLua com conectividade ao TCP

5.2.1 – Introdução

Este exemplo, desenvolvido pela PUC-Rio e codificado utilizando objetos NCLua, demonstra o poder de conexão da TV Digital com protocolos de comunicação, no caso específico TCP. Este exemplo foi escolhido pelo fato de que sua funcionalidade, acesso a uma página de pesquisa na Internet, pode abrir um leque enorme de possibilidades de aplicações. Em especial, pode-se vislumbrar situações em que o telespectador seja direcionado para uma página na internet com informações complementares sobre o programa que está sendo assistido, uma página de *chat* com um entrevistado ou uma página que venda produtos que estão sendo exibidos no momento.

5.2.2 – Descrição

Na tela do telespectador, é exibido um campo de entrada de dados e um campo de saída. Quando se preenche o campo de entrada, que pode ser feito pelo controle remoto ou simplesmente pelo teclado, a aplicação estará pronta para realizar uma pesquisa com os dados digitados pelo usuário no *site* Google.com. O resultado da pesquisa é então retornado pela URL "*I´m feeling lucky*", que é carregada no campo de saída.

Esta aplicação é composta por 6 arquivos:

- Google.Lua: Realiza a conexão com a porta TCP;
- Input.Lua: Mapa de caracteres, conjunto de letras e dígitos possíveis e a barra de inclusão de palavras para pesquisa;
- Main.ncl: Arquivo NCL que inicia os objetos do tipo NCLua;
- Output.Lua: Inclusão da barra de saída e faz a inclusão da barra de saída de dados;
- Parser.Lua: Descreve apenas a localidade como rua, telefone, cidade, estado, país e cep.
- Tcp.Lua: Controle das chamas assíncronas da classe TCP.

5.2.3 – Análise

É feita a análise das principais partes do código da aplicação. Dando início pelo nó de entrada da aplicação:

```
<media id="input" src="input.Lua"  descriptor="dsInput">
        <area id="select"/>
        <property name="text"/>
</media>
```

O arquivo main.ncl possui o nó de entrada "*input*", que por sua vez possui a propriedade "*text*", e é neste campo que é guardado o conteúdo digitado pelo usuário. Ainda neste código, nota-se a âncora "*select*" de "*area id*", que sempre será iniciada ao se pressionar a tecla "*ENTER*".

```
<media id="output" src="output.Lua" descriptor="dsOutput">
        <property name="text"/>
</media>
```

Este código descreve o nó de saída. Neste código, não existe mais a âncora pois nele será apresentado o resultado da consulta. Assim, só necessitamos da propriedade "*text*", cujo valor será exibido na tela.

Tendo em mãos o que se deseja pesquisar, pode-se então disparar uma consulta ao *Google*. Esta consulta é realizada pelo nó "google.Lua". Este nó possui as propriedades

"*search*" e "*result*", que como os próprios nomes sugerem, realizam respectivamente a busca e o retorno do resultado desta.

```
<media id="google" src="google.Lua">
        <property name="search"/>
        <property name="result"/>
</media>
```

Quando se altera o valor de "*search*", o *script* estará apto a fazer a consulta utilizando o valor que lhe foi passado. O retorno do resultado da consulta será guardado na propriedade "*result*".

```
<link xconnector="onBeginSet">
        <bind role="onBegin" component="input" interface="select"/>
        <bind role="set" component="google" interface="search">
              <bindParam name="var" value="$get"/>
        </bind>
        <bind role="get" component="input" interface="text"/>
</link>
```

O código acima é o elo responsável pelo início da consulta. Ele é ativado assim que a âncora "*select*" do nó "*google*" for iniciada, ou seja, após a tecla "ENTER" ter sido pressionada. Com isso o valor inserido na entrada corrente é copiado para a propriedade "*search*" do nó "*google*".

```
<link xconnector="onEndAttributionSet">
      <bind role="onEndAttribution" component="google"
      interface="result"/>
      <bind role="set" component="output" interface="text">
            <bindParam name="var" value="$get"/>
      </bind>
      <bind role="get" component="google" interface="result"/>
</link>
```

Assim que a busca for concluída, o nó *google* altera a propriedade *result*, fazendo com que o elo atualize o campo de saída.

A classe de eventos *tcp* é responsável pela comunicação utilizando o protocolo *TCP*. O uso desta é baseada em conexões assíncronas. Por conta disso, o programador deve ter total controle de toda a comunicação, desde a conexão, passando pelo envio e recebimento até a desconexão.

O módulo *tcp.Lua* realiza todo o trabalho de controle das chamadas assíncronas, permitindo que a comunicação seja programada de forma seqüencial. A principal função, *tcp.execute*, recebe uma função que pode conter os seguintes parâmetros:

- *tcp.connect*: receberá a porta de destino e o endereço;
- *tcp.send*: receberá a *string* contendo o valor que será transmitido;
- *tcp.receive*: retorno da *string* com o valor que recebido;
- *tcp.disconnect*: finaliza a conexão.

O código a seguir ilustra a função *tcp.execute*:

```
tcp.execute(
        function ()
            tcp.connect('www.google.com.br', 80)
            tcp.send('get /search?hl=pt-BR&btnI&q='..evt.value..'\n')
            local result = tcp.receive()
            if result then
                    result = string.match(result, '<A HREF="http://(.-)">')
or 'nao encontrado'
              else
                    result = 'error: ' .. evt.error
              end
              local evt = {
                    class    = 'ncl',
                    type     = 'attribution',
                    property = 'result',
                    value    = result,
              }
              evt.action = 'start'; event.post(evt)
              evt.action = 'stop' ; event.post(evt)
            tcp.disconnect()
        end
    )
```

Pode-se observar neste código que *tcp.connect* recebe o endereço www.google.com.br, que será acessado em sua porta 80. Logo após, a *string* com o valor a ser pesquisado é passado para a função *tcp.send*. O resultado recebido por *tcp.receive* é armazenado na variável local *result*. Caso nenhum erro tenha ocorrido, uma função verificará se o mesmo é uma URL válida ou uma resposta padrão do Google que informa que não foram encontradas páginas na Web contendo os termos da consulta. Neste último caso, o resultado receberá o valor “Não encontrado”. Finalmente o *script* estará preparado para criar o evento que trata do ncl, preenchendo a propriedade *result*.

5.3 – Terceiro Exemplo: Programa Esportivo utilizando NCLua

5.3.1 – Introdução

Esta aplicação, desenvolvida como parte dos requisitos necessários para a obtenção de grau de Bacharel em Ciência da Computação, utiliza NCL juntamente com objetos imperativos Lua. Diferentemente das outras aplicações anteriores, trata-se de uma aplicação com maior grau de complexidade.

A aplicação demonstra como a interatividade poderia ser utilizada em um programa de futebol. O telespectador assistindo ao programa poderá:

- Assistir a uma entrevista enquanto os melhores momentos do jogo são exibidos no canto inferior da tela;
- Optar por visualizar o jogo em tela cheia, mantendo a entrevista no canto inferior da tela;
- Manter apenas o jogo na tela, retirando assim a entrevista do canto inferior da tela;
- Consultar um servidor remoto para saber a atual classificação de seu time.

Enquanto o telespectador assiste e navega pelas funções do programa televisivo, o jogo continua a ser exibido em alguma parte da tela. Os menus do controle remoto dão ao telespectador a opção de controlar como o jogo e a entrevista são visualizados.

Esta aplicação necessita, para sua total funcionalidade, do Ginga Virtual Set-top Box 0.11.2.

5.3.2 – Descrição

Neste exemplo, inicia-se o vídeo em tela cheia. O vídeo é uma coletânea de grandes jogadas do futebol, representando o jogo propriamente dito. Neste momento não há qualquer tipo de interatividade, nenhum botão do controle remoto funciona, exceto o botão Verde, que inicia a interatividade da aplicação. O telespectador, ao clicar neste botão, abre o menu que fica à sua esquerda. Este menu possui 3 botões:

- Classificação: Redimensiona o vídeo principal ao lado da classificação atual do campeonato.

- Entrevista: Redimensiona o vídeo principal para o canto inferior direito e ao centro da tela é iniciado um programa de entrevistas estrelado por Galvão Bueno, chamado de *Galvão na estrada,* de propriedade da *Rede Globo*.
- Voltar: Retoma ao jogo, deixando-o em tela cheia e interrompendo a apresentação dos demais itens apresentados na tela.

A outra funcionalidade está no *Botão amarelo* do controle remoto. Este botão tem a finalidade de voltar ao jogo, porém diferentemente do botão *Voltar*, este continua exibindo a entrevista que o telespectador estava vendo enquanto o jogo era transmitido.

5.3.3 – Análise

Neste tópico serão apresentadas as principais partes do código da aplicação. Algumas partes do código não ser abordadas por se tratarem de uma repetição do que já foi mencionado. Todo o código da aplicação se encontra no anexo deste trabalho.

No início da aplicação foram definidos os conectores utilizando a IDE Eclipse e o plugin NCL 1.0.

```
<causalConnector id="onKeySelectionSetComecarParar">
  <connectorParam name="tecla"/>
  <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$tecla"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" max="unbounded" qualifier="par"/>
        <simpleAction role="stop" max="unbounded" qualifier="par"/>
        <simpleAction role="start" max="unbounded" qualifier="par"/>
    </compoundAction>
</causalConnector>
```

Nesta parte do código, o “onKeySelectionSetComecarParar” define ações quando uma tecla for pressionada pressionada. As ações podem ser a) a exibição de mídias, definida por *start*, b) interromper as mídias, através do atributo *stop,* ou c) alterar os valores, através do atributo *set*. O atributo <var> irá definir, em “set”, uma propriedade mapeada e definida pela variável.

```
<causalConnector id="onSelectionSetPararComecar">
  <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
      <compoundAction operator="seq">
          <simpleAction role="set" value="$var" max="unbounded" qualifier="par"/>
          <simpleAction role="start" max="unbounded" qualifier="par"/>
          <simpleAction role="stop" max="unbounded" qualifier="par"/>
      </compoundAction>
</causalConnector>
```

O conector "*onSelectionSetPararComecar*", diferentemente do conector citado anteriormente, não está ligado a qualquer tecla do controle remoto. Este conector foi criado a fim de que, ao se clicar no botão, as mídias recebessem regras temporais tais como iniciar, parar e mudar seus atributos.

Passando pelos conectores, o código prossegue pelas regiões da aplicação, conforme o código a seguir:

```
<regionBase>
<region width="100.0%" height="100.0%" id="rgTV"/>
<region width="100.0%" height="100.0%" id="rgVideo" zIndex="2"/>
<region width="15.0%" height="100.0%" id="rgMenuEsquerdo" />
<region height="7.0%" width="13.0%" left="0.5%" top="35.0%"  id="rgExemplo1" zIndex="2"/>
<region height="7.0%" width="13.0%" left="0.5%" top="43.0%"  id="rgExemplo2" zIndex="2"/>
<region height="7.0%" width="13.0%" left="0.5%" top="52.0%"   id="rgExemplo3" zIndex="2"/>
<region height="60.0%" width="85.0%" right="0.0%" top="20.0%"   id="rgEntrevista" zIndex="3"/>
<region height="60.0%" width="40.0%" right="0.0%" top="20.0%"   id="rgClassificacao" />
<region height="20.0%" width="85.0%" left="14.9%" top="00.0%"   id="rgTop" />
<region height="20.0%" width="65.0%" left="14.9%"  top="80.0%"   id="rgBot" />
<region id="rgLua" width="0%" height="0%" left="0%" top="0%" zIndex="3"/>
</regionBase>
```

Nestas linhas são declaradas todas as regiões utilizadas pela aplicação interativa. Foi escolhido o formato percentual, pois este facilita a definição de regiões com tamanhos distintos. Foram definidas um total de 10 regiões para a aplicação. São elas:

- rgTV: Região principal da aplicação, contempla a toda a tela de visão do usuário. As demais regiões são sub-regiões desta;
- rgVideo: Região correspondente ao vídeo principal, é nesta região onde o primeiro vídeo é exibido e contempla todo o espaço do vídeo;
- rgMenuEsquerdo: Correspondente ao menu a esquerda do vídeo. Esta possui 3 sub-regiões:
    - rgExemplo1: Sub-Região que representa a parte do botão de *Classificação* da aplicação;

- rgExemplo2: Sub-Região que representa a parte do botão de *Entrevista* da aplicação;
- rgExemplo3: Sub-Região que representa a parte do botão de *Voltar* da aplicação;

- rgClassificacao: Corresponde a região que se posiciona no canto direito, onde será visualizada a classificação dos times recuperada pela conexão TCP do nó NCLua, situa-se também entre as regiões rgTop e rgBot;
  - rgTot: Corresponde a região que se posiciona no canto superior, a direita da região correspondente a rgMenuEsquerdo;
  - rgBot: Corresponde a região que se posiciona no canto inferior, a direita da região correspondente a rgMenuEsquerdo;
- a mídia que ocupa a região correspondente a parte superior da aplicação, o nó de mídia “ICE”,
- a mídia que ocupa a região correspondente a parte central da aplicação, o nó de mídia “textoMono”,
- a mídia que ocupa a região correspondente a parte inferior da aplicação, o nó de mídia “textoMono”.

A seguir foi definida a base dos descritores.

```
<descriptorBase>
<descriptor region="rgTV" id="dTV"/>
<descriptor region="rgVideo" id="dVideo"/>
<descriptor region="rgMenuEsquerdo" id="dMenuEsquerdo"/>
<descriptor region="rgExemplo1" id="dExemplo1" focusIndex="1" moveDown="2" moveUp="3" focusBorderWidth="4" focusBorderColor="blue"/>
<descriptor region="rgExemplo2" id="dExemplo2" focusIndex="2" moveDown="3" moveUp="1" focusBorderWidth="4" focusBorderColor="blue"/>
<descriptor region="rgExemplo3" id="dExemplo3" focusIndex="3" moveDown="1" moveUp="2" focusBorderWidth="4" focusBorderColor="blue"/>
<descriptor region="rgEntrevista" id="dEntrevista"/>
<descriptor region="rgClassificacao" id="dClassificacao"/>
<descriptor region="rgBot" id="dBot"/>
<descriptor region="rgTop" id="dTop"/>
<descriptor id="dLua" region="rgLua"/>
</descriptorBase>
```

No código da base dos descritores, cada descritor criado corresponde a uma região da aplicação. Os descritores que indicam os botões da aplicação possuem os atributos:

- focusIndex: Define um índice de navegação para o objeto de mídia associado ao descritor. Sendo que o foco inicial é voltado ao que possui o focusIndex menor, neste caso o descritor dExemplo1.

- moveDown: Índice para a navegação entre os botões. Define qual dos botões deverá receber o foco caso seja pressionado o botão “para baixo” do controle remoto. O número indicado aponta para o focusIndex.
- focusBorderWidth: Largura da borda do botão da aplicação
- focusBorderColor: Cor da borda da aplicação

Logo após foram definidas as regras da aplicação:

```
<ruleBase>
    <rule id="regra1" var="opcao" comparator="eq" value="1"/>
    <rule id="regra2" var="opcao" comparator="eq" value="2"/>
    <rule id="regra3" var="opcao" comparator="eq" value="3"/>

</ruleBase>
```

As regras são utilizadas em conjunto com um nó de mídia do tipo *application/x-ginga-settings*. Isso é importante para o controle da mídia a ser exibida no menu. Todos os atributos de comparação são do tipo *eq* (equal) e comparados a números inteiros referentes à opção escolhida pelo usuário.

Os objetos de mídia são definidos pela tag <media>:

```
<port component="videoInicial" id="port_newDocument1_video"/>

<media descriptor="dVideo" src="media/Dribles.mpg" type="video/mpeg" id="videoInicial">
    <property name="bounds"/>
    <property name="visible"/>
    <property name="soundLevel"/>
</media>

<media descriptor="dEntrevista" src="media/Entrevista.mpg" type="video/mpeg" id="entrevista">
    <property name="bounds"/>
    <property name="visible"/>
    <property name="soundLevel"/>
</media>

<media descriptor="dMenuEsquerdo" src="media/menu.png" type="image/png" id="menuEsquerdo"/>
<media descriptor="dExemplo1" src="media/classificacaoBotao.png" type="image/png" id="exemplo1"/>
<media descriptor="dExemplo2" src="media/entrevistaBotao.png" type="image/png" id="exemplo2"/>
<media descriptor="dExemplo3" src="media/voltarBotao.png" type="image/png" id="exemplo3"/>

<media id="imgTop" src="media/imgTop.png" descriptor="dTop"/>
<media id="menuBot" src="media/menuBot.png" descriptor="dBot"/>

<media id="classificacao" src="imagem.jpg" descriptor="dClassificacao"/>
<media id="loading" src="media/loading.png" descriptor="dClassificacao"/>

<media id="lua" src="classificacao.lua" descriptor="dLua">
    <property name="busca"/>
    <property name="confereRecebimento"/>
</media>
```

Estas mídias serão executadas pela aplicação. Nesta aplicação, foram escolhidas mídias de vídeo nos formatos *MPG* e imagem no formato *PNG e JPG.* A mídia Lua trata do script Lua que será acessado pelo NCL.

Foi definido para cada uma das mídias um *descritor,* que realizará a ligação entre a mídia e a região. Cada mídia possui seu nome com sua propriedade *id,* o tipo em que estamos trabalhando e o *src*, que recebe como valor o local do vídeo a ser exibido.

Os nós cujos vídeos podem sofrer algum tipo de ação, como por exemplo, o *bounds* que trata do redimensionamento da mídia, recebem também um atributo *property*. Esta propriedade recebe quatro valores separados por espaço, que indicam, em ordem: a posição em relação à margem esquerda, a posição em relação ao topo, a largura da mídia e a altura da mídia.

Pode-se notar também outros tipos de *property ,* como o *visible,* que pode receber um valor *true* ou um valor *false*, que dependendo do valor escolhido a mídia será visível ou não na tela. Também podemos notar outra *property* definida como soundLevel, esta poderá receber valores reais de 0 a 1, que definem o volume da mídia exposta. O valor 0 remete-se a sem volume, o valor 1 remete-se a 100% do volume e os valores intermediários e estes as porcentagens que os mesmos representam. Já o nó de mídia Lua possui *property* especiais cujos nomes são definidos pelo programador e estes acessam o *script* fazendo alguma mudança no mesmo [21]. As 2 propriedades definidas visam a comunicação entre o Lua e o NCL, sendo a primeira do NCL para o Lua e a segunda o contrário. Mais a frente estas propriedades serão explicadas quando for tratar do nó Lua.

A *tag* a seguir, apresentada como *tag* <*port*>, simboliza o ponto de entrada da aplicação, indica qual componente e onde o programa inicia. O vídeo principal do jogo de futebol é iniciado assim que o documento for interpretado. Esta *tag* é obrigatória em qualquer documento do tipo NCL, pois sem a mesma o documento não consegue definir qual mídia será executada ao iniciar o aplicativo.

```
<port component="videoInicial" id="port_newDocument1_video"/>
```

Definidas regiões, os descritores e os conectores, é hora de programar visando o que será exibido na tela. Serão então destacados os *elos* da aplicação, estes que controlam toda a

execução da aplicação e a relação entre as mídias. O primeiro elo a ser tratado corresponde ao botão *Classificação* de código a seguir:

```
<!-- Classificação-->
<link xconnector="onSelectionSetPararComecar" id="primeiraOpcao">
      <bind role="onSelection" component="exemplo1"/>
      <bind role="set" component="nodeSettings" interface="opcao">
            <bindParam name="var" value="2"/>
      </bind>

      <bind role="start" component="imgTop"/>
      <bind role="start" component="loading"/>
      <bind role="start" component="menuBot"/>
      <bind role="stop" component="entrevista"/>

      <bind component="Lua" role="set" interface="busca">
            <bindParam name="var" value="envio"/>
      </bind>
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="95.0,95.0,290.0,298.0"/>
      </bind>
</link>
```

Este elo utiliza o conector *onSelectionSetPararComecar*, que já foi definido no cabeçalho do documento. A *id primeiraOpcao* corresponde ao nome da Tag. Nota-se também algumas *tags* dentro deste elo, como por exemplo, a *tag <bind>*. Estas *tags* relacionam as mídias com os papéis definidos no conector. São estas: *imgTop*, *loading*, *menuBot*, *entrevista*, *Lua* e *videoInicial.* Seu funcionamento é como se segue: caso o componente *entrevista* esteja em execução, o mesmo é finalizado. Já o componente *videoInicial* recebe pela *interface bounds,* um redimensionamento baseado em *pixels* para que ao seu lado possa ser visualizada o nó Lua trazendo consigo a classificação. Porém a consulta realizada ao nó Lua demanda um dado tempo, visto que uma consulta é realizada dinamicamente através da web. Isso explica a presença do nó *loading*, que ocupará o espaço vazio enquanto a consulta é realizada. Os nós restantes, *imgTop* e *menuBot*, têm apenas papel estético.

O elo a seguir *aoFinalStart* trata do objeto Lua em si. Quando a *interface confereRecebimento* for preenchida, o nó Lua irá finalizar e, ao término deste, a mídia classificação poderá buscar o arquivo *imagem.jpg*, pois o mesmo já se encontrará no computador local.

```
<link xconnector="aoFinalStart">
```

```
                <bind component="Lua" role="onEndAttribution" inter-
face="confereRecebimento" />
                <bind component="classificacao" role="start" />
</link>
```

O próximo elo a ser tratado é o que faz a funcionalidade do botão entrevista. O papel deste elo é iniciar a entrevista, deixando o jogo em segundo plano no canto inferior direito.

```
<!-- Entrevista-->
<link xconnector="onSelectionSetComecarParar" id="segundaOpcao">
     <bind role="onSelection" component="exemplo2"/>
     <bind role="set" component="nodeSettings" interface="opcao">
          <bindParam name="var" value="2"/>
     </bind>

     <bind role="start" component="imgTop"/>
     <bind role="start" component="menuBot"/>
     <bind role="start" component="entrevista"/>
     <bind role="stop" component="classificacao"/>
     <bind role="stop" component="loading"/>
     <bind role="set" interface="bounds" component="videoInicial">
          <bindParam name="var" value="510.0,390.0,130.0,120.0"/>
     </bind>
     <bind role="set" interface="soundLevel" component="videoIni-
cial">
          <bindParam name="var" value="0"/>
     </bind>

     <bind role="set" interface="bounds" component="entrevista">
          <bindParam name="var" value="95.0,95.0,540.0,298.0"/>
     </bind>
     <bind role="set" interface="soundLevel" component="entrevista">
          <bindParam name="var" value="1"/>
     </bind>
</link>
```

Este elo possui algumas similaridades com o elo Classificação. A diferença deste elo está na mídia que configurará o volume do vídeo principal. A *tag* <*bind*> associada ao vídeo principal da aplicação possui a *property soundLevel*, cuja finalidade já foi definida. Neste caso o volume do som da aplicação principal é levado ao zero enquanto o da entrevista é levado ao máximo, com os valores 0 e 1 sendo respectivamente definidos no parâmetro *value.*

E finalizando os elos associados a botões da aplicação, o último se trata do botão *Voltar*. Este elo tem a finalidade de voltar ao inicial da aplicação, encerrando todas as mídias que foram abertas pelo telespectador e deixando apenas o vídeo principal na tela.

```
<!--Voltar ao Jogo-->
<link xconnector="onSelectionSetParar" id="terceiraOpcao">
<bind role="onSelection" component="exemplo3"/>
<bind role="set" component="nodeSettings" interface="opcao">
<bindParam name="var" value="3"/>
</bind>

      <bind role="stop" component="imgTop"/>
      <bind role="stop" component="menuEsquerdo"/>
      <bind role="stop" component="menuBot"/>
      <bind role="stop" component="entrevista"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="exemplo2"/>
      <bind role="stop" component="exemplo1"/>
      <bind role="stop" component="exemplo3"/>
      <bind role="stop" component="Lua"/>
      <bind role="stop" component="loading"/>

      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="0.0,0.0,1000.0,1000.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="videoIni-
cial">
            <bindParam name="var" value="1"/>
      </bind>
</link>
```

Diferentemente do elo *Entrevista*, desta vez o volume do vídeo principal deve ser levado ao máximo, pois o mesmo poderia estar configurado com o valor 0, caso o telespectador estivesse acessado o botão *Entrevista.*

Passaremos agora a tratar dos elos dos botões do controle remoto, o botão *Verde* e o botão *Amarelo*. No código seguir, está definido o elo que configura o botão Verde do controle remoto.

```
<link xconnector="onKeySelectionSetComecarParar" id="comecaInter-
ativade">
      <bind role="onSelection" component="videoInicial">
            <bindParam name="tecla" value="GREEN"/>
      </bind>
      <bind role="start" component="menuEsquerdo"/>
      <bind role="start" component="exemplo2"/>
      <bind role="start" component="exemplo1"/>
      <bind role="start" component="exemplo3"/>
      <bind role="start" component="Lua"/>
      <bind role="stop" component="menuBot"/>
      <bind role="stop" component="imgTop"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="loading"/>
```

```
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="95.0,0.0,540.0,1000.0"/>
      </bind>
<bind role="set" interface="bounds" component="entrevista">
            <bindParam name="var" value="510.0,390.0,130.0,120.0"/>
      </bind>
</link>
```

Este botão tem a funcionalidade de se iniciar toda a interatividade da aplicação, não há como o telespectador conseguir acessar nada sem antes pressionar o botão *Verde*. Como se pode notar, este elo utiliza-se do conector *onKeySelectionSetComecarParar*, que possui 2 parâmetros: *tecla*, que recebe a tecla acionada no controle remoto e *valor*, que recebe o valor da propriedade do nó a ser modificada. Como visto no código anterior, *tecla* associa-se ao controle remoto enquanto *GREEN* associa-se a tecla Verde do controle remoto.

Após isto, mídias como *videoInicial* e *entrevista* são redimensionadas, enquanto as outras, dependendo do parâmetro *role* passado, ou iniciam, caso seja um *start*, ou terminam caso seja um *stop.*

E, por fim, foi definido o último elo que trata da configuração do botão *Amarelo* da aplicação. Este botão tem a funcionalidade fazer com que, caso o telespectador deseje retornar ao jogo em tela cheia, porém continuar vendo a entrevista, a entrevista não termine e, ao invés disso, seja redimensionada para o canto inferior da tela. A seguir o código deste elo que trata o botão Amarelo:

```
<link xconnector="onKeySelectionSetParar"
id="retornoVideoPrincipal">
      <bind role="onSelection" component="exemplo1">
            <bindParam name="tecla" value="YELLOW"/>
      </bind>
      <bind role="stop" component="menuEsquerdo"/>
      <bind role="stop" component="exemplo1"/>
      <bind role="stop" component="exemplo2"/>
      <bind role="stop" component="exemplo3"/>
      <bind role="stop" component="menuBot"/>
      <bind role="stop" component="imgTop"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="Lua"/>
      <bind role="stop" component="loading"/>
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="0.0,0.0,1000.0,1000.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="videoIni-
cial">
            <bindParam name="var" value="1"/>
```

```
      </bind>

      <bind role="set" interface="bounds" component="entrevista">
            <bindParam name="var" value="510.0,390.0,130.0,120.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="entrevista">
            <bindParam name="var" value="0"/>
      </bind>
</link>
```

A diferença deste elo para os demais elos é que, como o vídeo da entrevista ficará em segundo plano, apenas o áudio do vídeo principal têm seu valor igual a 1, enquanto o da entrevista passa a ter o valor 0. Isso faz com que apenas o áudio do vídeo principal seja ouvido pelo telespectador.

Finalizada a discussão do código da aplicação que trata do NCL baseando-se no Ginga-NCL, falta apenas apresentar o *script* Lua responsável pela pesquisa da classificação do campeonato em um servidor da internet. Deve-se destacar que qualquer dado poderia ser buscado, como imagens, vídeos e páginas. Este módulo do sistema faz uso da classe TCP, já descrita na apresentação do segundo exemplo desta seção.

O script criado chama-se *classificacao.lua*. Sua funcionalidade se restringe a buscar em um servidor o arquivo que contém a classificação dos times em um campeonato qualquer:

```
require 'tcp'

function setPropriedade(propriedade)

local evt = {
        class = 'ncl',
        type  = 'attribution',
        name  = propriedade,
    }

    evt.action = 'start'; event.post(evt)
    evt.action = 'stop' ; event.post(evt)
end


function handler (evt)


if evt.class ~= 'ncl' or evt.type ~= 'attribution'
   or evt.action ~= 'start' or evt.name ~= 'busca' then
   return
end
```

```
local host = "www.universotricolor.com"
tcp.execute(
  function ()
    tcp.connect(host, 80)
    print("Conectado ao servidor Web Remoto")
    tcp.send("GET http://www.universotricolor.com/wp-content/up-
loads/2009/12/classfinal.jpg\n")
    local confereRecebimento = tcp.receive("*a")
    if confereRecebimento then
      local arquivo = "./imagem.jpg"
      file = io.open (arquivo, "w+")
      file:write(confereRecebimento)
      print("Arquivo criado")
      file:close()
    end
    print("Retornando NCL")
    setPropriedade("confereRecebimento", 1)
    tcp.disconnect()
  end
  )
end

event.register(handler)
```

No início do código as funcionalidades da classe TCP são incluídas com o uso da palavra reservada do Lua *require*. Em seguida temos o código do primeiro tratador de eventos Lua, chamado de *setPropriedade*. Este faz a atribuição dos valores passados pelo NCL, e é o cabeçalho que qualquer documento NCLua deve possuir. Basicamente dois eventos são definidos, *start* e *stop*. A definição destes eventos é necessária para que o documento NCL perceba a atribuição de um valor a uma propriedade Lua.

A função seguinte faz o acesso ao arquivo de imagem que contém a classificação atual do campeonato em um servidor. Para que o código não fique continuamente executando, primeiramente foi definida uma condição para que a busca ocorra apenas quando de fato existir uma requisição pendente. Caso contrário, o código não executa. Quando uma requisição é detectada, se inicia a conexão com o servidor que hospeda o arquivo. O método *tcp.send* é utilizado para enviar uma requisição HTTP. O conteúdo da página acessada é obtido através do método *tcp.receive*. Este arquivo é armazenado localmente para que possa ser aberto pelo NCL.

O NCL é avisado que o arquivo já se encontra localmente e poderá ser aberto atribuindo um valor à função *setPropriedade.* Um evento do tipo *attribuition* é gerado para este propósito. Ele atribui um valor qualquer a propriedade *confereRecebimento*.

As funções que tratam dos eventos disparados pelo documento NCL são o *function handler (evt)*, que guarda as propriedades do evento disparado, e por último a *event.register(handler)*, que registra a função *handler*.

Figura 19 – Exemplo aplicação

Figura 20 – Exemplo aplicação parte 2

Figura 21 – Exemplo aplicação parte 3

Figura 22 – Exemplo aplicação parte 4

Figura 23 – Exemplo aplicação parte 5

Figura 24 – Exemplo aplicação parte 6

5.4 – Dificuldades Encontradas

Esta seção descreve as principais dificuldades encontradas durante todo o processo de elaboração da terceira aplicação.

A primeira dificuldade diz respeito ao uso de Ginga-NCL, o não conhecimento das linguagens e as ferramentas oferecidas não deram o suporte ideal a fim de proporcionar um desenvolvimento mais ágil da aplicação. A ferramenta Composer inicialmente foi manipulada a fim de se gerar o código NCL e facilitar a diagramação da aplicação, por ser versátil e de fácil manipulação. Entretanto, quando necessitamos implementar aspectos não triviais, com os conectores de vários vídeos ao mesmo tempo, foi verificado que a ferramenta não daria o suporte necessário não trazendo os resultados esperados. Algumas vezes os vídeos que tinham que simplesmente parar continuava sua exibição. Neste caso foi necessário o desenvolvimento de código manual em sua visão textual e após isso portar o que foi feito para a IDE Eclipse e seu plugin NCL.

A IDE é muito interessante porém não tem um bom sincronizador de erros, estes não tinham qualquer descrição do porque de seus erros e somente com uma analise de todo o código se poderia ter noção do que está errada na aplicação.

E, por fim, NCLua, que também foi utilizada a IDE Eclipse com o plugin Lua para diagramação dos scripts que são chamados pelo NCL. Esta parte também sofre dos mesmos problemas que a parte anterior sofreu, como má sincronização de erros e o Sistema Operacional com o qual estava desenvolvendo a aplicação teve de ser trocado do Windows XP para o Fedora 11 por questões de incompatibilidades de aplicações NCLua não funcionarem como deveriam no Windows XP.

Capítulo 6 – Conclusão

Neste trabalho realizamos um estudo sobre o desenvolvimento de aplicativos para a TV Digital. Os focos deste estudo foram a) o *middleware* declarativo, ou seja, o Ginga-NCL e b) NCLua, e tal foco é justificado pelo fato de que os mesmos são hoje os mais utilizado pela comunidade para o desenvolvimento de aplicações, até em ração dos problemas de *royalties* enfrentados por Ginga-J.

Entretanto, por ser uma tecnologia recente, ainda existem inúmeros problemas que devem ser resolvidos. O primeiro, em relação ao uso de Java, só foi resolvido definitivamente em setembro do corrente ano, quando o Fórum SBTVD definiu que o Java DTV (especificação livre, recém criada pela Sun) seria adotado no Ginga-J, o *middleware* da TV Digital. Outro problema diz respeito a falta de equipamentos no mercado que sejam capazes de executar Java DTV. Poucas ferramentas de desenvolvimento estão disponíveis, e suas funcionalidades ainda são muito limitadas. Em especial, no que diz respeito as ferramentas de desenvolvimento, as seguintes melhorias poderiam ser feitas:

- Melhoramento da utilização dos conectores na ferramenta *Composer*. A primeira vista o *Composer* é um programa com um visual agradável e de fácil utilização. Entretanto, este programa apresenta alguns problemas, principalmente com os conectores. Estes devem ser revistos e melhorados para que a ferramenta se torne referência na implementação de programas para TV Digital.
- Melhoramento do *plugin* de Eclipse para o desenvolvimento NCL. O Eclipse, outra ferramenta utilizada no desenvolvimento de aplicações NCL (utilizando um *plugin*), ainda apresenta alguns problemas. Mais especificamente as informações a respeito dos erros nas aplicações não são exibidos. Com isso, um grande tempo é perdido no desenvolvimento das aplicações.

Capítulo 7 – Anexo: Codigos das Aplicações

main.ncl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl xsi:schemaLocation="http://www.ncl.org.br/NCL3.0/EDTVProfile
http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile" id="newDocument1">
<head>


<connectorBase>
<importBase documentURI="composerConnectorBase.conn" alias="con-
nBase"/>

<causalConnector id="onSelectionSetPararComecar">
  <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
      <compoundAction operator="seq">
          <simpleAction role="set" value="$var" max="unbounded"
qualifier="par"/>
          <simpleAction role="start" max="unbounded"
qualifier="par"/>
          <simpleAction role="stop" max="unbounded"
qualifier="par"/>
      </compoundAction>
</causalConnector>


<causalConnector id="onSelectionSetComecarParar">
  <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
      <compoundAction operator="seq">
           <simpleAction role="set" value="$var" max="unbounded"
qualifier="par"/>
           <simpleAction role="start" max="unbounded"
qualifier="par"/>
           <simpleAction role="stop" max="unbounded"
qualifier="par"/>
      </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetParar">
  <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
      <compoundAction operator="seq">
          <simpleAction role="set" value="$var" max="unbounded"
qualifier="par"/>
          <simpleAction role="stop" max="unbounded"
qualifier="par"/>
      </compoundAction>
</causalConnector>
```

```
<causalConnector id="onKeySelectionSetComecar">
  <connectorParam name="tecla"/>
  <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$tecla"/>
        <compoundAction operator="seq">
          <simpleAction role="set" value="$var" max="unbounded" qual-
ifier="par"/>
           <simpleAction role="start" max="unbounded"
qualifier="par"/>
       </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetParar">
  <connectorParam name="tecla"/>
  <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$tecla"/>
        <compoundAction operator="seq">
          <simpleAction role="set" value="$var" max="unbounded" qual-
ifier="par"/>
           <simpleAction role="stop" max="unbounded"
qualifier="par"/>
       </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetComecarParar">
  <connectorParam name="tecla"/>
  <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$tecla"/>
     <compoundAction operator="seq">
           <simpleAction role="set" value="$var" max="unbounded"
qualifier="par"/>
           <simpleAction role="stop" max="unbounded"
qualifier="par"/>
           <simpleAction role="start" max="unbounded"
qualifier="par"/>
     </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStart">
     <simpleCondition role="onEndAttribution"/>
     <simpleAction role="start"/>
</causalConnector>

</connectorBase>


<regionBase>
<region width="100.0%" height="100.0%" id="rgTV"/>
<region width="100.0%" height="100.0%" id="rgVideo" zIndex="2"/>
<region width="15.0%" height="100.0%" id="rgMenuEsquerdo" />
<region height="7.0%" width="13.0%" left="0.5%" top="35.0%"
id="rgExemplo1" zIndex="2"/>
```

```
<region height="7.0%" width="13.0%" left="0.5%" top="43.0%"
id="rgExemplo2" zIndex="2"/>
<region height="7.0%" width="13.0%" left="0.5%" top="52.0%"
id="rgExemplo3" zIndex="2"/>
<region height="60.0%" width="85.0%" right="0.0%" top="20.0%"
id="rgEntrevista" zIndex="3"/>
<region height="60.0%" width="40.0%" right="0.0%" top="20.0%"
id="rgClassificacao" />
<region height="20.0%" width="85.0%" left="14.9%" top="00.0%"
id="rgTop" />
<region height="20.0%" width="65.0%" left="14.9%"  top="80.0%"
id="rgBot" />

<region id="rgLua" width="0%" height="0%" left="0%" top="0%"
zIndex="3"/>
</regionBase>

<descriptorBase>
<descriptor region="rgTV" id="dTV"/>
<descriptor region="rgVideo" id="dVideo"/>
<descriptor region="rgMenuEsquerdo" id="dMenuEsquerdo"/>
<descriptor region="rgExemplo1" id="dExemplo1" focusIndex="1" move-
Down="2" moveUp="3" focusBorderWidth="4" focusBorderColor="blue"/>
<descriptor region="rgExemplo2" id="dExemplo2" focusIndex="2" move-
Down="3" moveUp="1" focusBorderWidth="4" focusBorderColor="blue"/>
<descriptor region="rgExemplo3" id="dExemplo3" focusIndex="3" move-
Down="1" moveUp="2" focusBorderWidth="4" focusBorderColor="blue"/>
<descriptor region="rgEntrevista" id="dEntrevista"/>
<descriptor region="rgClassificacao" id="dClassificacao"/>
<descriptor region="rgBot" id="dBot"/>

<descriptor region="rgTop" id="dTop"/>
<descriptor id="dLua" region="rgLua"/>
</descriptorBase>

<ruleBase>
     <rule id="regra1" var="opcao" comparator="eq" value="1"/>
     <rule id="regra2" var="opcao" comparator="eq" value="2"/>
     <rule id="regra3" var="opcao" comparator="eq" value="3"/>
</ruleBase>

</head>

<body>

<!-- Mídias-->
<port component="videoInicial" id="port_newDocument1_video"/>

<media descriptor="dVideo" src="media/Dribles.mpg" type="video/mpeg"
id="videoInicial">
     <property name="bounds"/>
     <property name="visible"/>
     <property name="soundLevel"/>
</media>
```

```
<media descriptor="dEntrevista" src="media/Entrevista.mpg" type="vi-
deo/mpeg" id="entrevista">
      <property name="bounds"/>
      <property name="visible"/>
      <property name="soundLevel"/>
</media>

<media descriptor="dMenuEsquerdo" src="media/menu.png"
type="image/png" id="menuEsquerdo"/>
<media descriptor="dExemplo1" src="media/classificacaoBotao.png"
type="image/png" id="exemplo1"/>
<media descriptor="dExemplo2" src="media/entrevistaBotao.png"
type="image/png" id="exemplo2"/>
<media descriptor="dExemplo3" src="media/voltarBotao.png" type="ima-
ge/png" id="exemplo3"/>

<media id="imgTop" src="media/imgTop.png" descriptor="dTop"/>
<media id="menuBot" src="media/menuBot.png" descriptor="dBot"/>

<media id="classificacao" src="imagem.jpg" descriptor="dClassifica-
cao"/>
<media id="loading" src="media/loading.png" descriptor="dClassifica-
cao"/>

<media id="Lua" src="classificacao.Lua" descriptor="dLua">
      <property name="busca"/>
      <property name="confereRecebimento"/>
</media>

<!-- Classificação-->
<link xconnector="onSelectionSetPararComecar" id="primeiraOpcao">
      <bind role="onSelection" component="exemplo1"/>
      <bind role="set" component="nodeSettings" interface="opcao">
            <bindParam name="var" value="2"/>
      </bind>

      <bind role="start" component="imgTop"/>
      <bind role="start" component="loading"/>
      <bind role="start" component="menuBot"/>
      <bind role="stop" component="entrevista"/>

      <bind component="Lua" role="set" interface="busca">
            <bindParam name="var" value="envio"/>
      </bind>
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="95.0,95.0,290.0,298.0"/>
      </bind>
</link>

<link xconnector="onEndAttributionStart">
                <bind component="Lua" role="onEndAttribution" inter-
face="confereRecebimento" />
                <bind component="classificacao" role="start" />
```

```
</link>


<!-- Entrevista-->
<link xconnector="onSelectionSetComecarParar" id="segundaOpcao">
      <bind role="onSelection" component="exemplo2"/>
      <bind role="set" component="nodeSettings" interface="opcao">
            <bindParam name="var" value="2"/>
      </bind>

      <bind role="start" component="imgTop"/>
      <bind role="start" component="menuBot"/>
      <bind role="start" component="entrevista"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="loading"/>
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="510.0,390.0,130.0,120.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="videoIni-
cial">
            <bindParam name="var" value="0"/>
      </bind>

      <bind role="set" interface="bounds" component="entrevista">
            <bindParam name="var" value="95.0,95.0,540.0,298.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="entrevista">
            <bindParam name="var" value="1"/>
      </bind>
</link>

<!--Voltar ao Jogo-->
<link xconnector="onSelectionSetParar" id="terceiraOpcao">
<bind role="onSelection" component="exemplo3"/>
<bind role="set" component="nodeSettings" interface="opcao">
<bindParam name="var" value="3"/>
</bind>

      <bind role="stop" component="imgTop"/>
      <bind role="stop" component="menuEsquerdo"/>
      <bind role="stop" component="menuBot"/>
      <bind role="stop" component="entrevista"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="exemplo2"/>
      <bind role="stop" component="exemplo1"/>
      <bind role="stop" component="exemplo3"/>
      <bind role="stop" component="Lua"/>
      <bind role="stop" component="loading"/>

      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="0.0,0.0,1000.0,1000.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="videoIni-
cial">
```

```
            <bindParam name="var" value="1"/>
      </bind>
</link>
<!-- Botões  GREEN YELLOW RED-->

<link xconnector="onKeySelectionSetComecarParar" id="comecaInter-
ativade">
      <bind role="onSelection" component="videoInicial">
            <bindParam name="tecla" value="GREEN"/>
      </bind>
      <bind role="start" component="menuEsquerdo"/>
      <bind role="start" component="exemplo2"/>
      <bind role="start" component="exemplo1"/>
      <bind role="start" component="exemplo3"/>
      <bind role="start" component="Lua"/>
      <bind role="stop" component="menuBot"/>
      <bind role="stop" component="imgTop"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="loading"/>
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="95.0,0.0,540.0,1000.0"/>
      </bind>
<bind role="set" interface="bounds" component="entrevista">
            <bindParam name="var" value="510.0,390.0,130.0,120.0"/>
      </bind>
</link>

<link xconnector="onKeySelectionSetParar"
id="retornoVideoPrincipal">
      <bind role="onSelection" component="exemplo1">
            <bindParam name="tecla" value="YELLOW"/>
      </bind>
      <bind role="stop" component="menuEsquerdo"/>
      <bind role="stop" component="exemplo1"/>
      <bind role="stop" component="exemplo2"/>
      <bind role="stop" component="exemplo3"/>
      <bind role="stop" component="menuBot"/>
      <bind role="stop" component="imgTop"/>
      <bind role="stop" component="classificacao"/>
      <bind role="stop" component="Lua"/>
      <bind role="stop" component="loading"/>
      <bind role="set" interface="bounds" component="videoInicial">
            <bindParam name="var" value="0.0,0.0,1000.0,1000.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="videoIni-
cial">
            <bindParam name="var" value="1"/>
      </bind>

      <bind role="set" interface="bounds" component="entrevista">
            <bindParam name="var" value="510.0,390.0,130.0,120.0"/>
      </bind>
      <bind role="set" interface="soundLevel" component="entrevista">
            <bindParam name="var" value="0"/>
```

```
     </bind>
</link>


</body>
</ncl>
```

classificacao.Lua

```
require 'tcp'

function setLuaPropertie(propriedade)
print(propriedade)

local evt = {
        class = 'ncl',
        type  = 'attribution',
        name  = propriedade,
    }

    evt.action = 'start'; event.post(evt)
    evt.action = 'stop' ; event.post(evt)
end


function handler (evt)

print(evt.class)
print(evt.type)
print(evt.start)
print(evt.name)

if evt.class ~= 'ncl' or evt.type ~= 'attribution'
   or evt.action ~= 'start' or evt.name ~= 'busca' then
   return
end

local host = "www.universotricolor.com"
print(evt.name)
tcp.execute(
  function ()
    tcp.connect(host, 80)
    print("Conectado ao servidor Web Remoto")
    tcp.send("GET http://www.universotricolor.com/wp-content/up-
loads/2009/12/classfinal.jpg\n")
    local confereRecebimento = tcp.receive("*a")
    if confereRecebimento then
      local arquivo = "./imagem.jpg"
      file = io.open (arquivo, "w+")
      file:write(confereRecebimento)
      print("Arquivo criado")
      file:close()
    end
```

```
    print("Retornando NCL")
    setLuaPropertie("confereRecebimento", 1)
    tcp.disconnect()
  end
  )
end

event.register(handler)
```

tcp.Lua

```
--[[
-- TODO:
-- * nao aceita `tcp.execute` reentrante
--Fonte: http://www.telemidia.puc-rio.br/~francisco/ncLua/index.html
--]]

local _G, coroutine, event, assert, pairs, type
    = _G, coroutine, event, assert, pairs, type
local s_sub = string.sub

module 'tcp'

local CONNECTIONS = {}

local current = function ()
    return assert(CONNECTIONS[assert(coroutine.running())])
end

local resume = function (co, ...)
    assert(coroutine.status(co) == 'suspended')
    assert(coroutine.resume(co, ...))
    if coroutine.status(co) == 'dead' then
        CONNECTIONS[co] = nil
    end
end

function handler (evt)
    if evt.class ~= 'tcp' then return end

    if evt.type == 'connect' then
        for co, t in pairs(CONNECTIONS) do
            if (t.waiting == 'connect') and
               (t.host == evt.host) and (t.port == evt.port) then
                t.connection = evt.connection
                t.waiting = nil
                resume(co)
                break
            end
        end
        return
    end
```

```
    if evt.type == 'disconnect' then
        for co, t in pairs(CONNECTIONS) do
            if t.waiting and
               (t.connection == evt.connection) then
                t.waiting = nil
                resume(co, nil, 'disconnected')
            end
        end
        return
    end

    if evt.type == 'data' then
        for co, t in pairs(CONNECTIONS) do
            if (t.waiting == 'data') and
               (t.connection == evt.connection) then
                resume(co, evt.value)
            end
        end
        return
    end
end
event.register(handler)

function execute (f, ...)
    resume(coroutine.create(f), ...)
end

function connect (host, port)
    local t = {
        host    = host,
        port    = port,
        waiting = 'connect'
    }
    CONNECTIONS[coroutine.running()] = t

    event.post {
        class = 'tcp',
        type  = 'connect',
        host  = host,
        port  = port,
    }

    return coroutine.yield()
end

function disconnect ()
    local t = current()
    event.post {
        class      = 'tcp',
        type       = 'disconnect',
        connection = assert(t.connection),
    }
end
```

```
function send (value)
    local t = current()
    event.post {
        class      = 'tcp',
        type       = 'data',
        connection = assert(t.connection),
        value      = value,
    }
end

function receive (pattern)
    pattern = pattern or '' -- TODO: '*l'/number
    local t = current()
    t.waiting = 'data'
    t.pattern = pattern
    if s_sub(pattern, 1, 2) ~= '*a' then
        return coroutine.yield()
    end
    local all = ''
    while true do
        local ret = coroutine.yield()
        if ret then
            all = all .. ret
        else
            return all
        end
    end
end
```

8 - Referências

[1] A Linguagem de Programação Lua. [S.l.:s.n.]. Manual e referência. Disponível em: <http://www.Lua.org/portugues.html>. Acesso em: 30 nov. 2009

[2] A TV Digital começa por São Paulo no final de 2007. FOLHA ON LINE. São Paulo. Diário. Disponível em < http://www1.folha.uol.com.br/folha/informatica/ult124u20749.shtml > Acesso em: 3 dez 2009

[3] Azevedo, E. G. A.; Teiceira, M.M.; Neto C. S. S. NCL Eclipse: Ambiente Integrado para o Desenvolvimento de Aplicações para TV Digital Interativa em Nested Conext Language. Universidade Federal do Maranhão- Departamento de Informática, São Luis – MA. [2008?]

[4] ABNT – Associação de Normas Técnicas. (2008) – Televisão digital terrestre – Codficaão de dados e especificações de transmissão para radiodifusão digital Parte 2: Ginga-NCL para receptores fixos e móveis – Linagem de aplicação XML para codificação de aplicações. NBR 15606-2

[5] Becker V.; Interatividade, sincronismo e documentos NCL [S.l.] 27 mar 2009. Disponível em <http://imasters.uol.com.br/artigo/12165/tvdigital/interatividade_sincronismo_e_documentos_ncl/imprimir/>, Acesso em: 12 ago. 2009

[6] Converge Comunicações. (2008, Outubro 29). TI INSIDE Online: Sun vai liberar Ginga Java para desenvolvedores até semana que vem. TI INSIDE Online. 10 nov. 2008

[7] Filho, G. H. H., Comunicação entre as linguagens NCL e Lua. [S.l.:s.n.] [2009] Disponível em: < http://imasters.uol.com.br/artigo/12080/tvdigital/comunicacao_entre_as_linguagens_ncl_e_Lua/ >, Acesso em: 10 ago. 2009

[8] Filho, G. H. H. Interface gráfica com NCLua. [S.l.:s.n.] [2009] Disponível em <http://imasters.uol.com.br/artigo/12979/tvdigital/interface_grafica_com_ncLua_/ >, Acesso em: 12 ago. 2009

[9] F. Stephen. Event-Driven Programming: Introduction, Tutorial. History. 2009

[10] GINGA RN: Tutorial NCL. [S.l.:s.n.] [2009]. Disponível em: <http://gingarn.wikidot.com/tutorialncl> Acesso em: 20 nov. 2009

[11] Interatividade na TV Digital. [S.l.:s.n.] Disponível em: <http://www.tvglobodigital.com/telespectadores/interatividade> Acesso em: 13 ago. 2009

[12] Junger, H. S. S. Um Framework para o Módulo de Sincronização do Middleware Ginga, Departamento de Informática, PUC-Rio. 2008

[13] Júnior, C. S. L., Uma análise do middleware Ginga e da TV Digital no Brasil. Monografias em Ciência da Computação, Universidade Federal de Juiz de Fora, Juiz de Fora 2008.

[14] LAVID-UFPB e Telemidia-PUC-Rio. [S.d.]. Ginga Digital TV Middleware Specification. TV interativa se faz com Ginga Disponível em: <http://www.ginga.org.br/> dez 07, Acesso em: nov. 2008

[15] Leite, L. E.C. TV Digital no Brasil e o Middleware Ginga. Fórum SBTVD – mopa Embedded Systems [S.I.] Disponível em: < http://www.dibeg.org/seminar/0902Peru_ISDB-T_seminar/No7Cunha,Luis.pdf> Acesso em: 20 ago. 2009

[16] Leurusalimschy R.; Figueredo L.H.; Celes W. Manual de Referência de Lua 5.1.[2006] Manual. Disponível em: <http://www.Lua.org/manual/5.1/pt/>Acesso em: 20 nov. 2009

[17] Lerusalimschy R., Uma introdução à programação em Lua. [S.I.] [2009] Disponível em: <http://www.scribd.com/doc/19157841/Uma-Introducao-a-Programacao-em-Lua>. Acesso em: 17 set. 2009

[18] Melo, E. L.; Ginga Anlálise do Middleware Ginga para fins de oferecimento de serviços específicos e avaliação de flexibilidade versão 0.1. – Departamento de Ciência da Computação, Universidade Federal de São Carlos. 26 nov. 2007.

[19] Middleware Ginga – TV Interativa se faz com o Ginga!. [S.l.:s.n.] Exemplos. Disponível em: <http://www.ncl.org.br/exemplos/index_30.html>. Acesso em: 11 nov. 2009.

[20] Nassif L. , CPqD - O padrão brasileiro de TV Digital. CPqD - Em Folha de São Paulo. [2004?]

[21] Neto C., Soares L.; Rodrigues R.; Barbosa S. Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0 e a Ferramenta Composer. Manual de Referência. PUC-Rio. Rio de Janeiro, Laboratório de Telemídia. 31 jul. 2007.

[22] Oliveira, L. S., Canal de retorno – Interação e Usabilidade (2008). [S.l.] São Paulo.

[23] Ratamero, E. M. Tutorial sobre a linguagem de programação NCL (Nested Context Language), PET-Tele, Curso de Engenharia de Telecomunicações, Niterói-Rj, 2007

[24] Régis, M. V. O.; Fechine, J. M. INTRODUÇÃO AO SISTEMA DE TV DIGITAL. Monografias em Ciência da Computação, UFCG, Departamento de Ciência da Computação, Campina Grande, PB.

[25] Rodrigues, R. F., & Soares, L. F. (2006). Produção de Conteúdo Declarativo para TV Digital. Rio de Janeiro, RJ.

[26] Soares, L. F. G.; Castro P. H. As múltiplas possibilidades do Middleware Ginga, técnica SBTVD-T - Grupo de Trabalho de Middleware do Brasil, [S.l.] p. 77-82, jun 2008

[27] Sampaio, L. D. H. Implementação de Aplicação para TV Digital utilizando Ginga-NCL, Monografias em Ciência da Computação, Universidade Estadual de Londrina, Londrina 2008

[28] Sant,Anna, F.; Cerqueira R.; Soares L. F. NCLua – Objetos Imperativos Lua na Linguagem Declarativa NCL.

[29] Santana F. Lua como mídia de NCL. Manual de Referência. [S.l.:s.n.] Disponível em: <http://www.telemidia.puc-rio.br/~francisco/ncLua/tutorial/index.html>. Acesso em: 10 out. 2009.

[30] Sant'Anna, F.; Neto, C. S. S.; Barbosa, S. D. J.; Soares, L. F. G. S. Aplicações Declarativas NCL com Objetos NCLua Imperativos Embutidos. Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, Rio de Janeiro, 2009.

[31] Soares, L. F., & Barbosa, S. D. (2008). TV digital interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade. In T. Kowaltowski, & K. Breitman, Atualizações em Informática (pp. 105-174). Rio de Janeiro, RJ: Editora PUC-Rio.

[32] Tome, T. Aspectos Críticos para a integração dos sistemas de Televisão Digital – Nota Técnica. [S.l.] São Paulo, 22 fev. 2006

[33] W3C – World Wide Web Consortium. (2005). Synchronized Multimedia Integration Language (SMIL 2.1). W3C Recommendation.

[34] W3C World Wide Web Consortium (2002). XHTML 1.0 – The Extensible HyperText Markup Language (Second Edition). W3C Recommendation